EPFL

Thèse n° 9802

PRIVACY-PRESERVING FEDERATED NEURAL NETWORK TRAINING AND INFERENCE

Présentée le 4 juillet 2023

Faculté informatique et communications
Laboratoire d'ingénierie de sécurité et privacy
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Sinem SAV

Acceptée sur proposition du jury

Prof. A. Argyraki, présidente du jury
Prof. C. González Troncoso, Prof. J.-P. Hubaux, directeurs de thèse
Prof. Y. Zhang, rapporteur
Prof. B. Berger, rapporteuse
Prof. M. Jaggi, rapporteur

École polytechnique fédérale de Lausanne

2023

"Of course it is happening inside your head, Harry,
but why on earth should that mean that it is not real?"
— Albus Dumbledore

To my family and myself...

Acknowledgements

This thesis would not have been possible without the support, guidance, and contributions of many collaborators, my friends, and my family.

First, I am extremely grateful to my advisors Prof. Jean-Pierre Hubaux and Prof. Carmela Troncoso. I would like to express my gratitude to Jean-Pierre for the opportunity, support, and the patience that he offered me. His vision and guidance were key to this thesis and to my future. I also enjoyed all the years at LDS with the friendly environment that he provided. I would like to thank Carmela for the great environment and feedback that she provided during my last year. I thank her for warm-welcoming me to SPRING. Her input on both my research and career was always inspiring.

I am grateful to my thesis committee members: Prof. Katerina Argyraki, Prof. Martin Jaggi, Prof. Bonnie Berger, and Prof. Yang Zhang for taking the time to read and review my thesis. I would also like to thank Laurent Girod, Sylvain Chatel, and Linus Gasser for helping with the translation of the abstract to french and to german.

I had good luck to have great collaborators. I would like to thank Prof. Manfred Claassen for his valuable input on the medical application part of this thesis. I am very grateful for Dr. Apostolos Pyrgelis' friendly, precious, and patient support throughout my work. It was great working with you and I learned a lot from your critical thinking and reviewing of our papers, and I had a lot of fun, too. This thesis would have not been possible without him and I still would defend that we work on "vertical" federated learning. I thank Dr. Juan R. Troncoso-Pastoriza for his kind support and all the timeless discussions we had about my work, even during extremely busy times. The technical contributions of all my co-authors and collaborators are acknowledged in the relevant chapters of this thesis.

I am extremely lucky to have been able to work with two amazing engineers, Joao Sa Sousa and Jean-Philippe Bossuat, and they became my friends. Without the endless discussions with Jean-Philippe on our works, this thesis would have simply not been possible. I am grateful to Joao for all his support and precious friendship. I thank them both for chalet weekends, holidays, and fun times. I thank also Sylvain Chatel and Christian Mouchet for all the interesting discussions, mental support, coffee breaks, and spicy foods! I was also lucky enough to meet some special colleagues and make friends during this journey: Ludovic Barman, Francesco Marino, Romain Bouyé, Ceyhun Alp, Ahmet Caner Yuzuguler, Mickael Misbach. I would like to also thank all SPRINGers, Sandra Siby, Kasra Edalatnejadkhamene, Laurent Girod, Mathilde Raynal, Bogdan Kulynych, Dario Pasquini, Boya Wang, Wouter Lueks, Theresa Stadler, and Klim Kireev, for welcoming me to SPRING and for fun times at the lounge.

I thank Angela Devenoge, Patricia Hjelt, and Isabelle Coke, for their endless and valuable help with the administrative tasks. I am also grateful to Holly Cogliati-Bauereis for her help in improving all my writing, with patience.
A special word goes to my best friends that I found along the way of Ph.D.: Ozge Orhan, Doruk Oner, and Omer Can Karaman. I would like to thank Can for all the raki nights and fun times! Without Doruk, my Ph.D. would be dull. Ozge, there is no way that I can express my gratitude to her; I will always remember all the fun times, her endless mental support, amazing(!) food, being the best neighbor, and the best friendship. I am extremely lucky to have found her again after several years. I thank them both for being the best partners in crime!
Last, but in no way least, I thank my partner (in life) Ercu, my sister Simay, and the most amazing parents. This thesis would have not been possible without their unconditional love and support. Although I found a second-home through these years, my home will always be where you are.

Lausanne, June 6, 2023 S. S.

Abstract

Training accurate and robust machine learning models requires a large amount of data that is usually scattered across data silos. Sharing, transferring, and centralizing the data from silos, however, is difficult due to current privacy regulations (e.g., HIPAA or GDPR) and due to business competition (e.g., in the finance field). An existing solution for collaborative machine learning is federated learning where several parties collectively train a machine learning model without sharing or transferring their local data. With federated learning, the local data is preserved on the parties premises, the global model is trained via an iterative exchange of cleartext gradients that are computed locally. These gradients have been shown to leak private information about the original training data of the parties through inference attacks. Consequently, any solution that does not incorporate additional security and privacy mechanism to protect these gradients put the training data and their subjects at risk.

In this thesis, we propose, implement, and optimize several neural network algorithms to preserve the privacy of the model and the data during federated learning. Our solutions mitigate federated learning attacks that target the gradients during training in the cross-silo and horizontal federated learning settings with N parties. We also protect the querier's evaluation data sent to prediction-as-a-service (PaaS) systems. To achieve this, we rely on lattice-based multiparty homomorphic encryption (MHE), where all communicated values between the parties remain encrypted and all computations are carried out under encryption. With this, our solutions ensure both the data and the model confidentiality during the training and the prediction under a passive adversary threat model that allows for collusions between up to $N-1$ parties. We (i) propose and implement privacy-preserving federated neural network operations for different neural network architectures (e.g., multilayer perceptrons or recurrent neural networks), (ii) evaluate their performance under cross-silo federated learning settings in terms of model performance, scalability, and efficiency, and (iii) show the maturity of our solutions for real-life use-cases (e.g., medical application). We experimentally show that our solutions' performance is similar to centralized or decentralized non-private approaches and that the communication overhead scales linearly with the number of parties.

First, we propose POSEIDON, a novel system that addresses the problem of privacy-preserving training and the evaluation of multilayer perceptrons (MLPs) and convolutional neural networks (CNNs) in an N-party federated learning setting by relying on MHE. To efficiently and securely execute the backpropagation algorithm, we provide a generic packing approach, *alternating packing*, that enables single instruction, multiple data (SIMD) operations on encrypted data. We also introduce arbitrary linear transformations within the cryptographic

bootstrapping operation. These transformations optimize the costly cryptographic computations over the parties; and we define a constrained optimization problem for choosing the cryptographic parameters.
Second, we propose RHODE. RHODE enables privacy-preserving training of, and prediction on, recurrent neural networks (RNNs) in the federated learning setting. We propose a novel packing scheme, *multi-dimensional packing*, for a better utilization of SIMD operations that is tailored for RNNs and mini-batch training. With multi-dimensional packing, we enable the efficient processing, in parallel, of a batch of samples. Finally, we show that our solutions are applicable to real-life applications and datasets, by replicating the training of a published state-of-the-art CNN architecture in our privacy-preserving federated learning setting tailored for single-cell disease classification tasks.

Keywords: federated learning, multiparty homomorphic encryption, secure collaborations, privacy-preserving machine learning, neural networks, private training.

Zusammenfassung

Das Training präziser und robuster maschinellen Lernmodelle erfordert eine große Menge an Daten, die normalerweise in verschiedenen Datensilos verteilt sind. Das Teilen, Übertragen und Zentralisieren der Daten aus den Silos ist aufgrund aktueller Datenschutzbestimmungen (z.B. HIPAA oder DSGVO(GDPR)) und Geschäftkonkurrenz (z.B. im Finanzbereich) jedoch schwierig. Eine bestehende Lösung für kollaboratives maschinelles Lernen ist das sogenannte föderierte Lernen, bei dem mehrere Parteien ohne das Teilen oder Übertragen ihrer lokalen Daten gemeinsam ein maschinelles Lernmodell trainieren. Obwohl die lokalen Daten nie direkt geteilt werden, wird das globale Modell durch den iterativen Austausch von Klartext-Lokalmodellen (z.B. Gradienten) trainiert. Dies kann zu einem Datenleck von privater Informationen führen. Dank der Anwendung statistischer Methoden auf die Gradienten kann ein Teil oder alle der ursprünglichen Trainingsdaten der Parteien entblößt werden. Folglich setzt jede Lösung, die keine zusätzlichen Sicherheits- und Datenschutzmechanismen zum Schutz dieser Gradienten enthält, die Trainingsdaten und ihre Themen einem Risiko aus.

In dieser Arbeit präsentierenwir mehrere neuronale Netzwerke, um die Privatsphäre des Modells und der Daten zu bewahren. Vor allem um Angriffe auf das föderierte Lernen während des Trainings in den *cross-silo*- und horizontalen föderierten Lernumgebungen mit N Parteien abzumildern. Wir schützen auch die Bewertungsdaten des Abfragers, die an PaaS-Systeme (Prediction-as-a-Service) gesendet werden. Um dies zu erreichen, setzen wir auf lattice-basierte *multiparty homomorphic encryption* (MHE), bei der alle kommunizierten Werte zwischen den Parteien und alle Berechnungen verschlüsselt ausgeführt werden. Auf diese Weise garantieren unsere Lösungen sowohl die Daten- als auch die Modellvertraulichkeit während des Trainings und der Vorhersage unter einem passiven Bedrohungsmodell, bis zu einer Kollaboration von $N-1$ Parteien. Unsere Arbeit beinhaltet (i) privatheitsbewahrende föderierte neuronalen Netzwerkoperationen für verschiedene neuronalen Netzwerkarchitekturen (z.B. *multilayer perceptrons* oder *recurrent neural networks*) und deren Implementation, (ii) Bewertung derer Leistung unter den Bedingungen des *cross-silo federated learning* in Bezug auf Qualität des Models, Skalierbarkeit und Effizienz und (iii) reale Anwendungsfälle, die die Reife unserer Lösungen zeigen (z.B. medizinische Anwendungen). Wir zeigen experimentell, dass sich die Modelleffizienz mit unseren Lösungen ähnlich wie bei zentralen oder dezentralen Ansätzen ohne Einhaltung der Privatsphäre verhält und dass der Kommunikationsaufwand linear mit der Anzahl der Parteien skaliert wird.

Zuerst schlagen wir ein neues System vor, POSEIDON, das erstmals im Bereich des datenschutzfreundlichen Trainings von Neuronalen Netzen eine Lösung für das Training und die Bewer-

tung von *multilayer perceptrons* (MLPs) und *convolutional neural networks* (CNNs) in einem N-Parteien-Federated-Learning-Setting mit MHE vorstellt. Um den sicheren Backpropagation-Algorithmus effizient ausführen zu können, zeigen wir einen generischen Ansatz, das *alternierende Packen*, welches erlaubt, *single instruction, multiple data* (SIMD) Operationen auf verschlüsselten Daten anzuwenden. Wir stellen auch beliebige lineare Transformationen innerhalb der kryptographischen Bootstrapping-Operation vor, optimieren somit die kostspieligen kryptographischen Berechnungen über die Parteien und definieren ein begrenztes Optimierungsproblem für die Wahl der kryptographischen Parameter.
Danach stellen wir RHODE vor. Es ermöglicht eine Datenschutz-gerechtes Training und Vorhersage von *recurrent neural networks* (RNNs) im selben föderierten Lernumfeld. Wir schlagen ein neues Verpackungsschema, Multi-dimensional packing", für eine bessere Nutzung der *single instruction, multiple data* (SIMD) Operationen unter Verschlüsselung vor, das speziell für RNNs und Mini-Batch-Schulung angepasst ist. Mit Multi-dimensional packing können wir die effiziente Verarbeitung eines Batches von Proben parallel durchführen. Schließlich zeigen wir, dass unsere Lösungen auf realen Anwendungen und Datensätzen anwendbar sind, indem wir das Training einer veröffentlichten State-of-the-Art-CNN-Architektur in unserem datenschutzgerechten föderierten Trainingsumfeld für die *single-cell disease* Klassifikation nachbilden.

Schlüsselwörter: Föderiertes Lernen, Mehrparteien-Homomorphe Verschlüsselung, sichere Zusammenarbeit, Datenschutz-gerechtes maschinelles Lernen, Neuronale Netze, private Schulung.

Résumé

L'entraînement de modèles algorithmiques d'apprentissage précis et robustes nécessite un grand volume de données qui sont généralement dispersées dans des silos de données. Cependant, le partage, le transfert et la centralisation des données à partir de ces silos sont toutefois difficiles en raison des réglementations en vigueur concernant la protection des données privées (par exemple, HIPAA ou GDPR) et en raison de la concurrence commerciale (par exemple, dans le domaine de la finance). Une solution existante d'apprentissage machine collaboratif est appelée apprentissage fédéré où, sans partager ni transférer leurs données locales, plusieurs tiers entraînent collectivement le modèle. Bien que les données locales soient conservées *in situ*, le modèle global est entrainé en clair via un échange itératif de modèles locaux (par exemple, des gradients). Ce qui provoque une potentielle fuite d'informations privées des données locales d'entraînement en raison d'inférences attaquant les gradients ou encore le modèle. Par conséquent, toute solution qui n'intègre pas de mécanisme de sécurité et de confidentialité supplémentaire pour protéger ces gradients met en danger les données de formation et leurs sujets.

Dans cette thèse, nous proposons plusieurs algorithmes de réseaux de neurones afin de préserver les données privées du modèle et des données d'entraînement, et pour mitiger les attaques sur l'apprentissage fédéré qui visent la phase d'entraînement dans le contexte de l'apprentissage fédéré avec N tiers et une distribution inter-silos horizontal des données. Nous protégeons également les données d'évaluation du demandeur envoyées aux systèmes de prédiction en tant que service (PaaS). Dans de tel scenario, le modèle entraîné dans le contexte de l'apprentissage fédéré est utilisé et le demandeur envoie les données d'évaluation privées pour le PaaS. Pour y parvenir, nous avons recours au chiffrement homomorphe multipartite (MHE) basé sur un treillis euclidien, où toutes les valeurs communiquées entre les tiers et tous les calculs sont effectués sous encryption. Grâce à cela, nos solutions garantissent à la fois la confidentialité des données d'entraînement et du modèle pendant l'entraînement, et la prédiction pour le modèle de menace d'un adversaire passif permettant des collusions jusqu'entre $N-1$ tiers. Nous (i) proposons et implementons des opérations de réseaux de neurones fédérés préservant la vie privée pour différentes architectures de réseaux de neurones (par exemple, des perceptrons multicouches ou des réseaux de neurones récurrents), (ii) évaluons leurs performances dans des contextes d'apprentissage fédéré inter-silos en termes de performances du modèle, d'évolutivité , et d'efficacité, et (iii) montrons la maturité de nos solutions dans des cas d'utilisation réels (par exemple, pour des applications médicales). Nous montrons expérimentalement que les performances du modèle avec nos solutions

restent similaires aux approches non-privée centralisées ou décentralisées, et que le coût de communication évolue linéairement avec le nombre de tiers.
Tout d'abord, nous proposons un nouveau système, POSEIDON, le premier de son genre dans le régime des réseaux neuronaux préservant les données privées qui aborde le problème de l'entraînement préservant les données privée et l'évaluation des perceptrons multicouches (MLP) et des réseaux de neurones convolutifs (CNN) dans le cadre de l'apprentissage fédéré de N-tiers en s'appuyant sur MHE. Pour exécuter efficacement l'algorithme de rétropropagation sécurisée, nous fournissons une approche d'empaquetage (packing) générique, *empactage alterné* qui permet des opérations à instruction unique, données multiples (SIMD) sur des données chiffrées. Nous introduisons également des transformations linéaires arbitraires dans l'opération de rafraichissement des encryption homomorphes (bootstrapping), optimisant ainsi les calculs cryptographiques coûteux entre les tiers ; et nous définissons un problème d'optimisation sous contrainte pour le choix des paramètres cryptographiques.
Deuxièmement, nous proposons RHODE. Ce systeme permet un entraînement et une prédiction préservant les données privées sur les réseaux de neurones récurrents (RNN) dans le même contexte d'apprentissage fédéré. Nous proposons un nouveau schéma de représentation des donnés (packing), le *packing multidimensionnel*, pour une meilleure utilisation des opérations à instruction unique/données multiples (SIMD) sous chiffrement, adapté aux RNN et à l'entraînement par mini-lot (mini-batch). Avec l'emballage multidimensionnel, nous permettons le traitement efficace, en parallèle, d'un lot d'échantillons.
Enfin, nous montrons que nos solutions sont applicables à des applications et des ensembles de données réels, en reproduisant dans le contexte de notre apprentissage fédéré l'entraînement d'un réseau de neurone de pointe à des fins de classification des maladies unicellulaires.

Mots-clés : apprentissage fédéré, chiffrement homomorphique multi-tiers, collaboration sécurisée, apprentissage machine préservant la vie privée, réseau de neurones, entraînement privé.

Contents

Introduction

In the era of big data and machine learning, neural networks are the state-of-the-art models. They achieve remarkable predictive performance in various domains such as healthcare, finance, and image recognition [19, 196, 267]. However, training an accurate and robust machine learning model requires a large amount of diverse and heterogeneous data [310]. This phenomenon raises the need for data sharing among multiple parties who seek to collectively train a machine learning model in order to extract valuable and generalizable insights.

Nonetheless, collecting large amounts of data from multiple parties remains a challenge due to the sensitive nature of the data, strict privacy regulations, such as HIPAA [3] or GDPR [8], and business competition between the parties [262]. In a medical setting, for example, hospitals need decision-support machine learning systems, e.g., for diagnosis, employing neural networks that require large datasets for training [183, 84]. One approach is to aggregate or pool all the data from different hospitals and to train the model in a centralized server. However, such a solution is not feasible in most real-life applications, due to the aforementioned privacy regulations. The process for data sharing, in particular for cross-border transfers, also needs to interface diverse legislation and ethics committees, which usually hinders the agile process of the research. Hence, building new operational systems that enable several parties to protect and to have ultimate control on their data while being able to collaboratively train machine learning models is more significant than ever. Consequently, solutions that enable privacy-preserving training of machine learning models on the data of multiple parties are highly desirable in many domains [186, 47, 130].

A simple solution for collective training is to outsource the data of multiple parties to a *trusted party* that trains the neural network model on their behalf. In this approach, the data and model's confidentiality relies on established stringent non-disclosure agreements. However, these confidentiality agreements require a significant amount of time to be prepared by legal and technical teams [182] and are very costly [157]. Furthermore, the trusted party becomes a single point of failure, hence both data and model privacy could be compromised by data breaches, hacking, leaks, etc. Therefore, solutions originating from the cryptographic community replace and emulate the trusted party with a group of computing servers. In particular, to enable privacy-preserving training of neural networks, several studies employ secure multiparty computation (MPC) techniques and operate on the two [205, 60], three [204, 278, 279], or four [53, 57] server models. Such approaches, however, limit the number of parties

among which the trust is split, often assume an honest majority among the computing servers, and require parties to communicate their data outside their premises. This might not be acceptable due to the privacy and confidentiality requirements and the strict data protection regulations. Furthermore, the trusted computing servers do not operate on their own data or benefit from the model training; hence, their only incentive is preventing the reputation harm if they are compromised, which increases the possibility of malicious behavior.

A recently proposed algorithm for collective training of neural networks – without data outsourcing – is *federated learning*. Instead of bringing the data to the model, the model is brought (via a coordinating server) to the clients who perform model updates on their local data. The updated models from the parties are averaged to obtain the global neural network model [194, 166]. As such, federated learning eliminates the need for storing huge amounts of data collected from clients in a centralized server. Although federated learning retains the sensitive input (training) data locally and eliminates the need for data outsourcing, the model might also be sensitive, e.g., due to proprietary reasons. This model becomes available to the coordinating server, thus placing the latter in a position of power with respect to the remaining parties. Recent research demonstrates that sharing intermediate model updates among the parties or with the server might lead to various privacy attacks, such as extracting parties' inputs or membership inference attacks [128, 283, 309, 199, 212, 304, 110, 145, 280, 85, 87]. In addition, the model becomes also prone to general inference attacks (that usually target inference frameworks) such as membership inference [254, 241, 299, 189] or model inversion [98, 302, 294] by any party upon receiving the global model. Hence, during the training process, any distributed learning approach requires the protection of the intermediate model updates from any party. Consequently, several works employ differential privacy to enable privacy-preserving exchanges of intermediate values and to obtain models that are free from adversarial inferences [177, 253, 17, 195]. Although differentially private techniques limit privacy attacks, they decrease the utility of the resulting machine learning model [229]. Furthermore, training robust and accurate models is challenging and requires less noise to be added, and as such, the level of privacy achieved in practice remains unclear [141]. Therefore, a distributed privacy-preserving deep learning approach requires a strong cryptographic protection of the intermediate model updates during the training and of the final model weights.

Recent cryptographic approaches for private distributed learning, e.g., [307, 104], not only have limited machine learning functionalities, i.e., regularized or generalized linear models, but also employ traditional encryption schemes that make them vulnerable to post-quantum attacks. This should be cautiously considered, as recent advances in quantum computing [114, 213, 266, 295], increase the need for deploying plausible quantum-resilient cryptographic schemes that eliminate potential risks for applications with long-term sensitive data. We proposed SPINDLE [102] (covered in the thesis by Froelicher [105]), a generic approach for privacy-preserving training of machine learning models in an N-party setting that employs multiparty lattice-based cryptography to achieve plausible post-quantum security guarantees. However, SPINDLE demonstrates the applicability of the approach only for generalized linear

models; and it lacks the necessary protocols and functions to support the training of more complex machine learning models, such as neural networks.

In this thesis, we propose, implement, and evaluate novel systems for privacy-preserving, quantum-resistant, federated learning-based training of and inference on neural networks with N parties. Our systems rely on multiparty homomorphic encryption (MHE) and ensure model and data confidentiality of both the parties and the querier, under a passive-adversarial model and collusions between up to $N-1$ parties [287]. We encrypt the full pipeline of federated learning end-to-end, such that all intermediate values that are communicated through the network or global/local models always remain under encryption. Each party in the collaborative training of neural network retains their data locally and trusts only themselves for the confidentiality of their data. Our systems do not require the collection of data in encrypted form, as opposed to prior works [126, 211], thus reducing the computational complexity.

Our main contributions are as follows:

- **Chapter 4 - Federated Multilayer Perceptron and Convolutional Neural Network Learning.** We present POSEIDON that enables the private training and evaluation of multilayer perceptron (MLPs) and convolutional neural networks (CNNs) in the federated learning setting with N parties, under encryption. We propose an alternating packing (AP) approach for the efficient use of single instruction, multiple data (SIMD) operations. Our results demonstrate that POSEIDON trains a 2-layer neural network model on a dataset with 23 features and 30,000 samples distributed among 10 parties, in 8.7 minutes.
- **Chapter 5 - Federated Recurrent Neural Network Learning.** We present RHODE that enables private training and evaluation of recurrent neural networks (RNNs) in the federated learning setting with N parties, under encryption. We address several challenges introduced by training RNNs under encryption, e.g., exploding gradients. We propose a multi-dimensional packing scheme that enables the efficient execution of mini-batch training over RNNs, through SIMD operations. We show that RHODE can process 256K samples distributed among 10 parties with an RNN of 32 hidden units and 4 timesteps, with 100 global training iterations, in $\sim$ 1.5 hours.
- **Chapter 6 - Federated Neural Network Learning for Disease-Associated Cell Classification.** We present PRICELL for evaluating our systems, within the medical framework of single-cell analysis. We efficiently replicate the training of a convolutional neural network (CellCnn) architecture, designed by Arvaniti and Claassen [32], in a decentralized and privacy-preserving manner. We enable private CellCnn training for single-cell analysis within our framework for a disease classification task. We design new packing strategies that are tailored for the execution of CellCnn architecture. Our solution converges comparably to the training with centralized data, and we improve on POSEIDON in terms of training time. In a setting with 10 parties, we improve POSEIDON's execution time by at least one order of magnitude. PRICELL trains, in less than 20 minutes, a

CellCnn model on a training set of 200 cells per sample, 1,000 samples per party, and 32 features across 10 parties.

Here, we summarize the key contributions and features of all the systems that we present throughout this thesis:

- **Data Locality.** Our solutions eliminate the need for communicating the parties' confidential data outside their premises, which might not be always feasible due to privacy regulations [3, 8]. This is in contrast to MPC-based solutions that require parties to distribute their data among several servers, hence fall under the cloud-outsourcing model.
- **Security.** In our solutions, data holders trust only themselves and no other party. As such we guarantee the data and model confidentiality properties under a passive-adversarial model and collusions between up to $N-1$ parties. On the contrary, MPC-based solutions limit the number of parties among which the trust is split (typically, 2, 3, or 4 servers) and assume an honest majority among them for efficiency.
- **Scalability.** The communication and the computation overhead in POSEIDON, RHODE, and PRICELL is linear in the number of parties, when all other parameters are kept the same.
- **Accuracy/Model Performance.** We evaluate our systems on several real-world datasets and various neural network architectures/parameters. We show that in all cases it achieves training accuracy or model performance levels on par with the centralized or decentralized non-private approaches.

Publications. Chapter 4 contains the contributions of the paper [244] that was published at NDSS'21 [12]. Then, Chapter 5 contains the contributions of the paper [243] that is accepted at PETS'23. Finally, in Chapter 6, we describe the findings of [242] that was published in the journal Patterns (Volume 3, Issue 5) [13]. We also compile our contributions on privacy-preserving predictions on neural networks in the paper [136] that is accepted at Cloud S&P 2023 and which is not covered in this thesis. We enable the privacy-preserving federated training of generalized linear models in the paper [102] that published in PETS'21 [11] and that is not covered in this thesis.

Patents/Impact. The publication of Chapter 4 has received the best prize in the CSAW'21 Applied Research Competition in Europe [10]. To evaluate our contributions in the field of applied homomorphic encryption, we participated in IDASH'21 for the "Homomorphic Encryption-based Secure Viral Strain Classification" track and ranked 2nd in the competition [9].

Both [102] and the contributions in POSEIDON [244] resulted in the filing of patents. The first patent filed is *"System and Method for Privacy-Preserving Distributed Training of Machine Learning models on Distributed Datasets"* [101] and the second is *"System and Method for*

Privacy-Preserving Distributed Training of Neural Network Models on Distributed Datasets" [245]. These patents and the several contributions of this thesis are at the core of and used by the startup company Tune Insight SA [16] that enables secure data collaborations for real-life scenarios.

Software. We developed and implemented the prototypes for evaluating our systems described in this thesis. The implementations of POSEIDON (Chapter 4) and RHODE (Chapter 5) are licensed by Tune Insight SA [16]. Our implementation for the prototype for PRICELL (Chapter 6) is available at [15] and is deposited at Zenodo [14].

Thesis Outline. In Chapter 1, we introduce the background on the building blocks that are used throughout this thesis: neural networks, federated learning, multiparty homomorphic encryption, and the frequently used symbols and notations. In Chapter 2, we describe the state-of-the-art privacy-preserving machine learning frameworks, and we position our contributions in this thesis. We present the common baseline, i.e., the system and threat model, problem statement, and the system overview, in Chapter 3; this facilitates the presentation in all remaining chapters. We describe our systems for privacy-preserving federated multilayer perceptron and convolutional neural networks and recurrent neural network learning in Chapters 4 and 5, respectively. We show the maturity of our solutions in a medical setting for collaborative disease-associated cell classification tasks in Chapter 6. Finally, we summarize the contributions of this thesis in the Conclusion (Chapter 7).

1 Background

We introduce the main building blocks that we rely on for the remaining chapters of the thesis. We first introduce the background information about neural networks (NNs), federated learning, and the multiparty homomorphic encryption (MHE) [97, 33] scheme which are used in all chapters.

1.1 Neural Networks

Neural networks are machine learning algorithms that extract complex non-linear relationships between the input and output data. They are used in a wide range of fields such as pattern recognition, data/image analysis, face recognition, forecasting, and data validation in the medicine, banking, finance, marketing, and health industries [19]. Typical neural networks are composed of a pipeline of layers, where feed-forward and backpropagation steps for linear and non-linear transformations (activations) are applied to the input data, iteratively [115]. Each training iteration is composed of one forward pass and one backward pass, and the term epoch refers to processing once all the samples in a dataset. Below, we provide a brief explanation for multilayer perceptrons, convolutional neural networks, and recurrent neural networks which are used throughout this thesis. We summarize their pipeline in Figure 1.1.

1.1.1 Multilayer Perceptrons (MLPs)

MLPs are fully-connected neural network structures that are widely used in the industry. MLPs are composed of an input layer, one or more hidden layer(s), and an output layer; each neuron is connected to all the neurons in the following layer. At iteration k, the weights between layers j and $j+1$, are denoted by a matrix W_j^k, whereas the matrix L_j represents the activation of the neurons in the j^{th} layer. The forward pass first requires the linear combination of each layer's weights with the activation values of the previous layer, i.e., $U_j = W_{j-1}^k \times L_{j-1}$. Then, an activation function is applied to calculate the values of each layer as $L_j = \varphi(U_j)$.

Backpropagation, a method based on gradient descent, is then used to update the weights

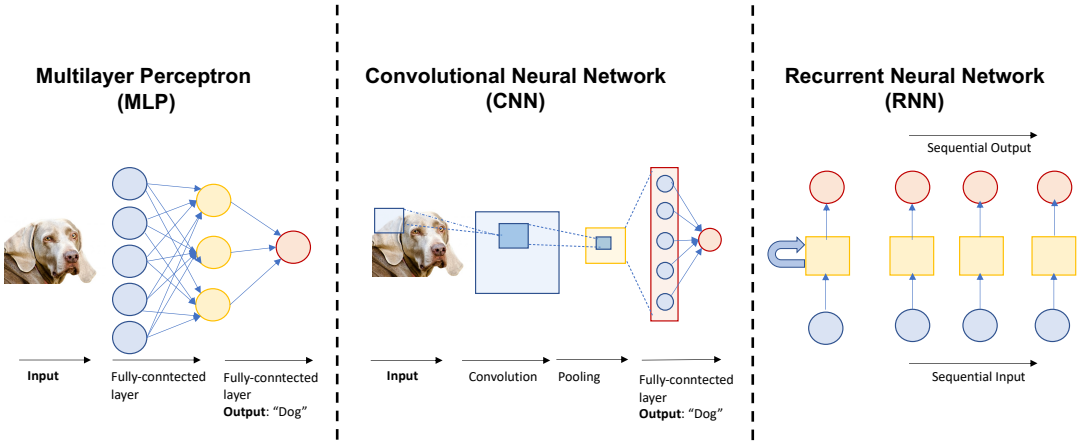


Figure 1.1: *Typical architecture for various neural networks.*

during the backward pass. Here, we describe the update rules for mini-batch gradient descent, where a random batch of sample inputs of size B is used in each iteration. The aim is to minimize each iteration's error based on a cost function E (e.g., mean squared error) and to update the weights, accordingly. The update rule is $W_j^{k+1} = W_j^k - \frac{\eta}{B}\nabla W_j^k$, where η is the learning rate and ∇W_j^k denotes the gradient of the cost function with respect to the weights and is calculated as $\nabla W_j^k = \frac{\partial E}{\partial W_j^k}$. We note that backpropagation requires several transpose operations applied to matrices/vectors, and we denote the transpose of a matrix/vector as W^T. Given a cost function, e.g., the mean squared error or the mean absolute error, E_l denotes the error with respect to the the true class labels Y. Then, the error term, δ_j for each layer j, is calculated iteratively. Starting from the last layer $\delta_l = \frac{\partial E}{\partial L_j} \odot \frac{\partial L_j}{\partial U_j}$, where $\frac{\partial L_j}{\partial U_j}$ is the derivative of the activation function (φ') and $\odot$ denotes element-wise multiplication. Following the chain rule[1] $\delta_j^k = (\delta_{j+1}^k \times (W_j^k)^T) \odot \varphi'(U_j)$ for each hidden layer j, $\nabla W_j^k = \delta_j^k L_j$.

1.1.2 Convolutional neural networks (CNNs)

CNNs follow a very similar sequence of operations, i.e., forward and backpropagation passes, and typically consist of convolutional (CV), pooling, and fully connected (FC) layers. It is worth mentioning that CV layer operations can be expressed as FC layer operations by representing them as matrix multiplications; in our protocols for Chapter 4, we simplify CV layer operations by employing this representation [278, 4]. Finally, pooling layers are downsampling layers where a kernel, i.e., a matrix that moves over the input matrix with a stride of a, is convoluted with the current sub-matrix. For a kernel of size $k \times k$, the minimum, maximum, or average (depending on the pooling type) of each $k \times k$ sub-matrix of the layer's input is computed.

[1]The chain rule is used to differentiate composite functions. It states that the derivative of $f(g(x))$ is $f'(g(x))g'(x)$.

1.1.3 Recurrent Neural Networks (RNNs)

RNNs are a special type of model that enables the learning on sequential data (e.g., time-series) [239, 90]. 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 hidden nodes that form a directed graph. The RNN input is a sequence where each item in the sequence is called a timestep. The hidden layer consists of several hidden units (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 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 [90], takes a sequential input ($X = x_1, x_2, \cdots, x_T$) of T timesteps then outputs the prediction y_t through the hidden units and an activation function by sequentially computing the hidden state at time t as

$$\begin{aligned} h_t &= \varphi(h_{t-1} \times W + x_t \times U + b_h) \\ y_t &= h_t \times V + b_y \end{aligned} \tag{1.1}$$

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 as the hidden and output biases. φ is the activation function that is usually chosen to be Tanh. Note that the input at each timestep x_t can itself be 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 thesis, 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. We here note that a Jordan network [146] changes the first line of this equation to $h_t = \varphi(y_{t-1} \times W + x_t \times U + b_h)$, and consequently the size of the weight matrix W to $o \times h$. Jordan networks [146] 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 $o \times h$.

Similarly to feed-forward neural networks, the training of RNNs is an iterative process with a series of forward and backward passes. The forward pass comprises the prediction by using the formulas in Eq. 1.1. However, backpropagation is computed via the backpropagation through time (BPTT) algorithm that takes into account the error propagated through the hidden units and computes the gradients by following the chain rule [239]. We describe in detail the operations of this process under Chapter 5, Section 5.2, Algorithm 6.

Finally, we note that there are various RNN structures, i.e., one-to-one, one-to-many, many-to-one, and many-to-many, depending on the input and output layer size (e.g., a many-to-many structure yields outputs for κ timesteps). We differentiate these RNN structures from RNN variants in terms of architecture, such as Gated Recurrent Unit (GRU) or Long Short-Term

Memory (LSTM).

1.2 Federated Learning

Federated Learning is an emerging collaborative machine learning approach that is proposed by Google and enables multiple parties to train a model, without sharing their local training data [194, 167, 166]. With federated learning, each party 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. [194] in Algorithm 1. W_i^k denotes the model weights of the i^{th} party during the k^{th} 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 and sends them to a random subset of parties S that execute the local gradient descent algorithm (Local Gradient Descent$(\cdot)$; see Chapters 4, 5, and 6 for the details under different systems) to update their local model. The model updates from each party are aggregated to the global model via weighted averaging (n_i denotes the number of local samples at the i^{th} party 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 party's side. We denote the local batch size as b and global batch size as B, where for a number of local iterations is 1 and N parties, $B = b \times N$.

Cross-silo vs. cross-device federated learning. Federated learning can be instantiated as a cross-device or cross-silo setting. Cross-device refers to the setting where the participating parties are the devices such as smartphones and hence the number of parties can be in the order of millions (e.g., Google's applications such as mobile keyboard prediction [121]). Cross-silo, on the other hand, refers to the setting where the parties are organizations where the data is siloed. Thus, the number of parties is small and usually below 200 [149, 133]. In this thesis, we tackle privacy-preserving federated learning in the cross-silo setting and assume that the parties are available to participate in cryptographic operations. Note that an extension to a cross-device setting is possible without changing the protocols, but remains unpractical due to the assumption on availability and the underlying cryptographic operations that would result in unrealistic runtimes for millions of devices.

1.3 Multiparty Homomorphic Encryption (MHE)

In our system, we rely on a Cheon-Kim-Kim-Song (CKKS) [62] variant of the MHE scheme proposed by Mouchet et al. [209]. In this scheme, a public collective key is known by all parties while the corresponding secret key is distributed among them. As such, decryption is possible only with the participation of *all* parties. Our motivations for choosing this scheme are as follows: (i) It is well suited for floating point arithmetic. (ii) It is a lattice-based scheme which relies on the ring learning with errors (RLWE) problem [193], thus making our system plausibly

Algorithm 1 Federated Averaging Algorithm (FedAvg)

Initialize the global model W^0 ▷ Server executes
for $k = 0 \rightarrow e-1$ **do** ▷ Global iterations
  Choose P random subset of N parties
  Send W^k to each party in S
  Each party P_i in P executes (in parallel):
  for $\ell = 0 \rightarrow l-1$ **do** ▷ Local iterations
    $W_i^{k+1} \leftarrow$ Local Gradient Descent$(\cdot)$
  end for
  $W^{k+1} \leftarrow \sum_{i=1}^{N} \frac{n_i}{n} W_i^{k+1}$ ▷ Server executes
end for

secure against post-quantum attacks [20]. (iii) It enables secure and flexible collaborative computations between parties without sharing their respective secret key. And (iv) it enables a secure collective key-switch functionality, that is to say, changing the encryption key of a ciphertext without decryption. Here, we provide a brief description of the cryptographic scheme's functionalities that we use throughout our protocols. The cyclotomic polynomial ring of dimension $\mathcal{N}$, where $\mathcal{N}$ is a power-of-two integer, defines the plaintext and ciphertext space as $R_{Q_L} = \mathbb{Z}_{Q_L}[X]/(X^{\mathcal{N}}+1)$, with $Q_L = \prod_0^L q_i$ in our case. Each q_i is a unique prime, and Q_L is the ciphertext modulus at an initial level L. Note that a plaintext encodes a vector of up to $\mathcal{N}/2$ values. Below, we introduce the main functions that we use in our system. Note that non-linear operations are not supported by this scheme (only addition and multiplication operations can be performed).

We denote by $\boldsymbol{c} = (c_0, c_1) \in R_{Q_L}^2$ and $p \in R_{Q_L}$, a ciphertext (indicated as boldface) and a plaintext, respectively. $\bar{p}$ denotes an encoded (packed) plaintext. We denote by $L_{\boldsymbol{c}}$, $S_{\boldsymbol{c}}$, L, and S, the current level of a ciphertext $\boldsymbol{c}$, the current scale of $\boldsymbol{c}$, the initial level, and the initial scale (precision) of a fresh ciphertext, respectively; we use the equivalent notations for plaintexts. The functions listed below that begin with 'D' are distributed, and executed among all the secret-key-holders, whereas the others can be executed locally by anyone with the public key.

- SecKeyGen(1^λ): Returns the set of secret keys $\{sk_i\}$, i.e., sk_i for each party P_i, for a security parameter λ.
- DKeyGen$(\{sk_i\})$: Returns the collective public key pk.
- Encode(msg) : Returns a plaintext $\bar{p} \in R_{Q_L}$ with scale S, encoding msg.
- Decode$(\bar{p})$: For $\bar{p} \in R_{Q_{L_p}}$ and scale S_p, returns the decoding of p.
- DDecrypt$(\boldsymbol{c}, \{sk_i\})$: For $\boldsymbol{c} \in R_{Q_{L_{\boldsymbol{c}}}}^2$ and scale $S_{\boldsymbol{c}}$, returns the plaintext $p \in R_{Q_{L_{\boldsymbol{c}}}}$ with scale $S_{\boldsymbol{c}}$.
- Enc$(pk, \bar{p})$: Returns $\boldsymbol{c}_{pk} \in R_{Q_L}^2$ with scale S such that DDecrypt$(\boldsymbol{c}_{pk}, \{sk_i\}) \approx \bar{p}$.

- $\mathsf{Add}(\boldsymbol{c}_{pk}, \boldsymbol{c'}_{pk})$: Returns $(\boldsymbol{c}+\boldsymbol{c'})_{pk}$ at level $\min(L_{\boldsymbol{c}}, L_{\boldsymbol{c'}})$ and scale $\max(S_{\boldsymbol{c}}, S_{\boldsymbol{c'}})$.
- $\mathsf{Sub}(\boldsymbol{c}_{pk}, \boldsymbol{c'}_{pk})$: Returns $(\boldsymbol{c}-\boldsymbol{c'})_{pk}$ at level $\min(L_{\boldsymbol{c}}, L_{\boldsymbol{c'}})$ and scale $\max(S_{\boldsymbol{c}}, S_{\boldsymbol{c'}})$.
- $\mathsf{Mul}_{\mathsf{pt}}(\boldsymbol{c}_{pk}, \bar{p})$: Returns $(\boldsymbol{c}p)_{pk}$ at level $\min(L_{\boldsymbol{c}}, L_p)$ with scale $(S_{\boldsymbol{c}} \times S_p)$.
- $\mathsf{Mul}_{\mathsf{ct}}(\boldsymbol{c}_{pk}, \boldsymbol{c'}_{pk})$: Returns $(\boldsymbol{c}\boldsymbol{c'})_{pk}$ at level $\min(L_{\boldsymbol{c}}, L_{\boldsymbol{c'}})$ with scale $(S_{\boldsymbol{c}} \times S_{\boldsymbol{c'}})$.
- $\mathsf{RotL/R}(\boldsymbol{c}_{pk}, k)$: Homomorphically rotates $\boldsymbol{c}_{pk}$ to the left/right by k positions.
- $\mathsf{Res}(\boldsymbol{c}_{pk})$: Returns $\boldsymbol{c}_{pk}$ with scale $S_{\boldsymbol{c}}/q_{L_c}$ at level $L_{\boldsymbol{c}}-1$.
- $\mathsf{SetScale}(\boldsymbol{c}_{pk}, S)$: Returns $\boldsymbol{c}_{pk}$ with scale S at level $L_{\boldsymbol{c}}-1$.
- $\mathsf{KS}(\boldsymbol{c}_{pk} \in R^2)$: Returns $\boldsymbol{c}_{pk} \in R^2$.
- $\mathsf{DKeySwitch}(\boldsymbol{c}_{pk}, pk', \{sk_i\})$: Returns $\boldsymbol{c}_{pk'}$.
- $\mathsf{DBootstrap}(\boldsymbol{c}_{pk}, L_{\boldsymbol{c}}, S_{\boldsymbol{c}}, \{sk_i\})$: Returns $\boldsymbol{c}_{pk}$ with initial level L and scale S.

Below, we summarize the most important operations of this scheme:

- **Key Generation:** Each party i locally generates a *secret* key (sk_i). The *public* key (pk) and evaluation keys ($\{ek\}$) are collectively generated from the parties' local secret keys ($\mathsf{DKeyGen}(\{sk_i\})$). $\{ek\}$ is a set of special public keys that are required for the execution of homomorphic multiplications and rotations. With this scheme, any party can encrypt and locally perform homomorphic operations by using pk and $\{ek\}$, but decrypting a ciphertext requires the collaboration among all parties ($\mathsf{DDecrypt}(\mathbf{c}_{pk}, \{sk_i\})$).
- **Arithmetic Operations and Parallelization:** The CKKS homomorphic encryption scheme enables approximate arithmetic over a vector of complex numbers ($\mathbb{C}^{\mathcal{N}/2}$) for a polynomial ring of dimension $\mathcal{N}$. Encrypted operations over $\mathbb{C}^{\mathcal{N}/2}$ are carried out in a single instruction, multiple data (SIMD) manner, which enables parallelization over the $\mathcal{N}/2$ plaintext vector slots.
- **Arithmetic Circuit Depth and Distributed Bootstrapping:** A fresh ciphertext has an initial level of L and at most L multiplications (i.e., an L-depth circuit) can be evaluated on it. When the levels are exhausted, the ciphertext is refreshed with a collective bootstrapping operation ($\mathsf{DBootstrap}(\mathbf{c}_{pk}, \{sk_i\})$) that enables further operations. We note here that $\mathsf{DBootstrap}(\cdot)$ is a costly operation with an overhead approximately 2 orders of magnitude higher than a homomorphic addition/multiplication.
- **Slot Rotations:** The slots of a ciphertext vector can be re-arranged via rotations to the left or right, i.e., $\mathsf{RotL/R}(\mathbf{c}_{pk}, k)$ homomorphically rotates $\mathbf{c}_{pk}$ to the left/right by k positions, using $\{ek\}$.

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

We note that $\mathsf{Res}(\cdot)$ is applied to a resulting ciphertext after each multiplication. Furthermore, for a ciphertext at an initial level L, at most an L-depth circuit can be evaluated. To enable more homomorphic operations to be carried on, the ciphertext must be re-encrypted to its original level L. This is done by the bootstrapping functionality ($\mathsf{DBootstrap}(\cdot)$). $\mathsf{Encode}(\cdot)$ enables us to pack several values into one ciphertext and operate on them in parallel.

For the sake of clarity, we differentiate between the functionality of the collective key-switch ($\mathsf{DKeySwitch}(\cdot)$) that requires interaction between all the parties, and a local key-switch ($\mathsf{KS}(\cdot)$) that uses a special public-key. The former is used to decrypt the results or change the encryption key of a ciphertext. The latter, which does not require interactivity, is used during the local computation for slot rotations or relinearization after each multiplication. Lastly, we provide the cryptography glossary that explains several terms that are used throughout this thesis in Appendix A.

1.4 Notations

Table 1.1 summarizes the commonly used symbols and notations used throughout this thesis. Table 1.2 summarizes the additional frequently used symbols and notations for the Chapters 4 and 5. Finally, we present the notations that are only used in Chapter 6 in the Table 1.3

Notation	Description
P_i	i^{th} Party
Q	Querier
X_i $X_i[n]$	Input matrix of P_i n^{th} row of the input matrix
y_i	True labels of P_i
N	Total number of parties
η	Learning rate
$\varphi(\cdot)$	Activation function
$\varphi'(\cdot)$	Derivative of the activation function
E_j^k	Error propagated in layer j, at k^{th} iteration
$\nabla W_{j,i}^k$	Gradient computed in P_i for a layer j, at k^{th} iteration, k, j, and i are omitted if no ambiguity
$\odot$	Element-wise multiplication
$\times$	Matrix or vector multiplication
$\|\|$	Concatenation
	Cryptographic Parameters and Operations
$\mathcal{N}$	Ring dimension
λ	Security level
$\boldsymbol{W}$	Encryption of W (bold-face)
R_Q	The ring $\mathbb{Z}_Q[X]/(X^{\mathcal{N}}+1)$, with $\mathcal{N}=2^d$
$\mathsf{RotL}(\mathbf{c}_{pk}, k)$	Homomorphically rotates $\mathbf{c}_{pk}$ to the left by k positions, pk is omitted if no ambiguity.
$\mathsf{RotR}(\mathbf{c}_{pk}, k)$	Homomorphically rotates $\mathbf{c}_{pk}$ to the right by k positions, pk is omitted if no ambiguity.

Table 1.1: *Frequently Used Symbols and Notations for Chapters 4, 5, and 6*

Notation	Description
$W_{j,i}^{k}$	Weight matrix in P_i for a layer j, at k^{th} iteration, k, j, and i are omitted if no ambiguity
n	Number of data samples
d	Number of features
d_a	Degree of an approximated polynomial
ℓ	Total number of layers
h_j	Number of neurons in j^{th} layer for MLPs
h_ℓ	Number of output labels for MLPs
h	Hidden dimension for RNNs
b	Local batch size
B	Global batch size
e	Number of global iterations
l	Number of local iterations
Cryptographic Parameters and Operations	
$\bar{msg}$	Encoded (packed) plaintext vector msg
L	Initial level of a ciphertext
S	Initial scale of a ciphertext
L_c	Current level of a ciphertext c
S_c	Current scale of a ciphertext c
$\mathsf{RIS}(\boldsymbol{c}, p, s)$	RotateInnerSum with $log_2(s)$ number of rotations.
$\mathsf{RR}(\boldsymbol{c}, p, s)$	RotateReplication with $log_2(s)$ number of rotations.

Table 1.2: *Additional Frequently Used Symbols and Notations for Chapters 4 and 5*

Notation	Description
s	Total number of samples over parties
$L_{n\times c\times m}$	Batch of multi-cell samples
$C_{m\times h}$	Convolution filters (weights)
$W_{h\times o}$	Weight matrix of dense layer
n	Number of multi-cell samples in a batch (batch size)
c	Number of cells in multi-cell input
m	Number of features (markers) per cell
h	Number of filters for the convolutional layer
o	Number of labels (phenotype)
μ	Momentum
Cryptographic Parameters and Operations	
$\boldsymbol{A}^{\ell}$	Encryption of A (bold-face) at level ℓ
$\mathsf{MultImag}(\cdot)$	Multiply the slots by the imaginary unit i
$\mathsf{Conjugate}(\cdot)$	Complex conjugate of the slots
$\mathsf{InnerSum}_{i,j}(\cdot)$	Sum j batches of i slots
$\mathsf{Replicate}_{i,j}(\cdot)$	Replicate batches of i slots j times

Table 1.3: *Additional Frequently Used Symbols and Notations used for Chapter 6*

2 Related Work

In this chapter, we discuss state-of-the-art privacy-preserving machine learning (PPML). We first describe the literature on PPML that focuses on machine learning models other than neural networks in Section 2.1. We then discuss privacy-preserving time-series solutions in Section 2.2 that facilitate the contributions of Chapter 5. We discuss the state-of-the-art private solutions for the inference and training of neural networks, including MLP, CNN, and RNN models, in Sections 2.3 and 2.4, respectively. Finally, we describe some biomedical applications of privacy-preserving tools that are related to the contributions of Chapter 6. Note that we only review software or algorithm based solutions and exclude hardware-based solutions such as Intel Software Guard Extension, SGX that lacks the theoretical privacy guarantees [38].

Our contributions in Chapters 4, 5, and 6 enable the training of neural networks in a cross-silo federated learning while protecting the confidentiality of the training data of each party, the local gradients, and the global model. By design, our work mitigates passive federated learning inference attacks during training, because all intermediate values that are exchanged between parties remain encrypted and as it enables all the computations of the federated learning pipeline under encryption.

2.1 Privacy-Preserving Machine Learning (PPML)

Several PPML works focus exclusively on the training of (generalized) linear models [31, 144, 44, 76, 159, 162]. They rely on *centralized* solutions where the learning task is securely outsourced to a server, notably using homomorphic encryption (HE) techniques. As such, these works do not solve the problem of privacy-preserving *distributed* machine learning, where multiple parties collaboratively train a machine learning model on their data. As a result, the data records of individual parties have to be transferred to another server, which might be difficult due to the data protection regulations. To address the problem of privacy-preserving distributed machine learning, several works propose secure multi-party computation (MPC) solutions where several tasks, such as clustering and regression, are distributed among two or three servers [139, 52, 216, 108, 111, 25, 248, 41, 67]. Although such solutions enable

multiple parties to collaboratively learn on their data, the trust distribution is limited to the number of computing servers that train the model, and they rely on assumptions, such as non-collusion or an honest majority among the servers. There exist only a few works that extend the distribution of machine learning computations to N parties ($N \geq 4$) and that remove the need for outsourcing [75, 307, 104, 102]. A recent work also enables the principal component analysis (PCA) in the federated learning setting by relying on HE and MPC [99]. For the privacy-preserving learning of linear models, Zheng et al. propose a system, Helen, that combines HE with MPC techniques [307]. However, the use of the Paillier additive HE scheme [218] makes their system vulnerable to post-quantum attacks. To address this issue, we introduced SPINDLE [102], that constitutes the framework for MHE-based decentralized training of generalized linear models but is not covered in this thesis. These works have paved the way for PPML computations in the N-party setting, but none of them addresses the challenges associated with the privacy-preserving training of and inference on neural networks (NNs).

2.2 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 [169, 198, 227], whereas others combined secure aggregation with differential privacy to bound the leakage stemming from the computations [251, 49]. More recently, Liu et al. [185] employed secure multi-party computation (MPC) techniques for privacy-preserving collaborative medical time-series analysis based on dynamic time warping. Dauterman et al. [81] use function secret-sharing (FSS) to support various functionalities, e.g., multi-predicate filtering, on private time-series databases. All these authors focus on simple collaborative time-series tasks. Whereas, in Chapter 5, we focus mainly on the privacy-preserving training of machine learning models on time-series data.

2.3 Privacy-Preserving Inference on Neural Networks

In this research direction, the majority of works operate on the following setting: a central server holds a trained neural network model and clients communicate their *secret* evaluation data to privacy-preserving obtain predictions-as-a-service (PaaS) [112, 54, 184, 148, 223, 237, 202, 134, 39, 126, 233, 50, 65, 197, 188, 48, 64, 174, 274, 173]. Their aim is to protect both the confidentiality of the server's model and the clients' data. The first work on this line, CryptoNets, proposes the use of a ring-based leveled HE scheme to enable the inference phase on encrypted data [112]. Cryptonets is later improved in terms of latency [50]. Some works rely on hybrid approaches by employing two-party computation (2PC) and HE [148, 184, 134, 233], or secret sharing and garbled circuits to enable privacy-preserving inference on neural networks [223, 237, 202]. For instance, Liu et al. propose MiniONN [184] that relies on leveled HE, in which the non-linear activation functions are enabled by secure

two-party computation (2PC) - garbled circuits, whereas Juvekar et al. propose Gazelle that employs HE supporting basic SIMD operations, and garbled circuits [148]. Riazi et al. use garbled circuits to achieve constant round communication complexity during the evaluation of binary neural networks [237], whereas Mishra et al. propose a similar hybrid solution that outperforms previous works in terms of efficiency, by tolerating a small decrease in the model's accuracy [202]. Other works attempt to increase the efficiency through matrix operations [192, 143] or focus on different machine learning models. For example, Ran et al. recently propose a system that enables HE-based inference on graph convolutional networks [231]. More recently, Kim et al. adopted the well-known private inference framework and applied it to more realistic applications, such as human action recognition, and achieved better latency than previous works [160].

Some works developed a deep learning graph compiler for multiple HE cryptographic libraries, such as SEAL [1], HElib [118], and Palisade [238]. Boemer et al. enables the deployment of a model that is trained with well-known frameworks (e.g., Tensorflow [18], PyTorch [222]) and that enables predictions on encrypted data [39, 40]. Dalskov et al. use quantization techniques to enable efficient privacy-preserving inference on models trained with Tensorflow [18] by using MP-SPDZ [6] and they demonstrate benchmarks for a wide range of adversarial models [77]. CHET, on the other hand, supports domain-specific language for specifying and compiling neural networks for different domains to inference protocols that rely on FHE [80].

Only a few recent studies focus on privacy-preserving prediction on RNNs and its variants. Bakshi and Last propose CryptoRNN; it employs HE but requires interaction between the client and the server to refresh the ciphertexts [35]. Rathee et al. propose SiRNN that relies on novel two-party computation (2PC) protocols and lookup tables, combined with an iterative algorithm, to approximate non-linear math functions [232]. Feng et al. tackle privacy-preserving natural language processing [95] by relying on MPC. And Feng et al. propose a hybrid approach that combines HE and garbled circuits for evaluating GRU networks on text analysis tasks [94]. More recently, Jang et al. extended the CKKS-HE scheme to the multivariate ring-learning with errors (RLWE) problem in order to support efficient matrix operations that are required for evaluating GRUs on sequence modeling, regression, and classification tasks [140].

Similar to these works, our solutions protect the confidentiality of the model and the client's evaluation data during the prediction phase. However, our work is broader, as it focuses on both the privacy-preserving training and the inference, protecting the training data, the resulting model, and the evaluation data by relying on MHE.

2.4 Privacy-Preserving Training of Neural Networks

In this section, we review the state-of-the-art solutions that address the privacy-preserving training of neural networks relying on various techniques or algorithms. Below, we differentiate the solutions that operate on a centralized setting and further discuss the other techniques and algorithms that operate on the decentralized settings.

Centralized Solutions. A number of works focus on *centralized* solutions to enable privacy-preserving learning of [259, 17, 293, 276, 211, 126]. Some of them [259, 17, 293], in order to derive models that are protected from inference attacks, employ differentially private techniques to execute the stochastic gradient descent while training a neural network. However, they assume that the training data is available to a *trusted* party that applies the noise required during the training steps. Other works [276, 211, 126] rely on HE to outsource the training of multi-layer perceptrons to a central server.

HE-Based Centralized Solutions. Along the direction of HE-based centralized solutions, Vizitu et al. propose using MORE (Matrix Operation for Randomization or Encryption) [163], a noiseless fully homomorphic encryption (FHE) scheme to enable deep neural networks over encrypted data for medical applications. The underlying cryptographic scheme, however, achieves very poor security guarantees as it is based on strong assumptions on the distribution of the plaintext, does not provide IND-CPA security [277], and is vulnerable to post-quantum attacks [163]. Nandakumar et al. [211] propose training multi-layer perceptrons over homomorphically encrypted data by using the FHE library HElib [118]. The choice of the cryptographic parameters remains far from the realistic use of the underlying scheme for deep learning tasks. Besides, they show that applying HE to deep neural networks is not yet practical, because the proposed packing solution, the parallelization capabilities, and the use of local bootstrapping slow down the overall training significantly. For the same neural network structure, our system in Chapter 4 (POSEIDON) is several orders of magnitude more efficient than the proposed solution, achieves better security guarantees, and enables learning tasks over N parties. A similar approach, also employing HElib [118], to train multi-layer perceptrons is proposed by Hesamifard et al. [126]. Their solution avoids the use of local bootstrapping by allowing interactions between the client and the server. These solutions either employ cryptographic parameters that are far from realistic [276, 211] or yield impractical performance [126]. Furthermore, they do not support the training of neural networks in the N-party setting, whereas this is the main focus of our work. Several works also employ HE to either enable secure aggregation in decentralized learning [224, 225] or to encrypt the clients' data, before outsourcing it to a server that performs the model training [300]. Note that secure aggregation solutions still leak information and prone to several inference attacks upon decryption on the client side. Below, we summarize the techniques for privacy-preserving training of neural networks in a decentralized or federated manner:

Secure Multiparty Computation (MPC). A number of works that enable privacy-preserving *distributed* learning of neural networks employ MPC approaches where the parties' confidential data is distributed among two [205, 22, 82], three [204, 278, 279, 127, 60, 264, 308], and four servers [53, 57] (2PC, 3PC, and 4PC, resp.). For instance, in the 2PC setting, to train various machine learning models, Mohassel and Zhang describe a system where data owners process and secret-share their data among two non-colluding servers [205], and Agrawal et al. propose a framework that supports discretized training of neural networks by ternarizing the weights [22]. Then, Mohassel and Rindal extend [205] to the 3PC setting and introduce

new fixed-point multiplication protocols for shared decimal numbers [204]. Wagh et al. further improve the efficiency of privacy-preserving neural network training on secret-shared data [278] and provide security against malicious adversaries, assuming an honest majority among three servers [279]. Hie et al. demonstrate the application of privacy-preserving neural network training with 3PC on a predictive model for drug-target interactions [127], and Chen and Zhong enable the privacy-preserving backpropagation algorithm by using ElGamal encryption [89] for neural network training over vertically partitioned data in 2PC setting [60]. Their work enables the privacy-preserving neural network training over vertically partitioned data among two servers, in a semi-honest setting. More recently, 4PC honest-majority malicious frameworks [53, 57] or arbitrary-number semi-honest frameworks [165] for PPML were proposed. These works split the trust between more servers and achieve round complexities better than previous ones, yet they do not address neural network training among N-parties. Note that 2PC, 3PC, and 4PC solutions fall under the *cloud outsourcing* model, as the data of the parties has to be transferred to several servers among which the majority has to be trusted. Our work, however, focuses on a distributed setting, where the data owners maintain their data locally and iteratively update the collective model yet data and model confidentiality is ensured in the existence of a dishonest majority in a semi-honest setting, thus withstanding passive adversaries and up to $N-1$ collusions between them. We provide a comparison between our contributions in Chapter 4 and aforementioned works in Section 4.5.6.

Overall, all aforementioned MPC solutions enable the training of regression models and feed-forward neural networks, whereas RNNs or its variants have not been studied. As a result, several algorithms, e.g., back-propagation through time, and crucial operations, e.g., gradient clipping, have not been addressed. One recent work, which relies on additive secret-sharing, investigates the training of RNNs as a case study [301], but without evaluating runtime performance or scalability. Privacy-preserving training of RNNs and its variants in the federated learning setting is our main contribution in Chapter 5.

Federated Learning. Federated learning (FL), proposed by Google, enables different parties to perform the collective training of a machine learning model while maintaining the data in their local premises and communicating only the intermediate values/local model updates to an aggregation server [194, 167, 166]. The main idea is to train a global model on data that is distributed across multiple clients, with the assistance of a server that coordinates model updates on each client and averages them. Similarly to tabular or image data, federated learning was applied on many time-series applications such as load forecasting [93], natural language processing [59, 121], traffic flow prediction, [187], and healthcare systems [107]. However, recent research has shown that sharing gradients and local model updates in federated learning lead to severe privacy leakage because membership inference and data/label reconstruction attacks are effective [128, 283, 309, 199, 212, 304, 110, 145, 280, 85, 87]. To counter this, some works focus on secure aggregation techniques for distributed neural networks where the aggregation of the local updates is done in a secure way by using HE [224, 225, 297] or MPC [42]. Although encrypting the gradient values prevents the leakage of parties' confidential data to the central server, these solutions do not account for potential leakage from the *aggregate* val-

ues themselves. In particular, parties that decrypt the received model before the next iteration are able to infer information about other parties' data from its parameters [128, 199, 212, 309].

Differential Privacy (DP). One common approach for privacy-preserving *decentralized* or *federated* machine learning is to integrate differential privacy (DP) into the learning phase, such that the intermediate values exchanged among the parties and the global model meet the DP requirements [253, 177, 17, 195, 285, 288]. Shokri and Shmatikov [253] apply DP to the parameter update stages, and Li et al. design a privacy-preserving federated learning system for medical image analysis where the parties exchange differentially private gradients [177]. McMahan et al. propose differentially private federated learning for RNNs [195], by employing the moments accountant method [17], to protect the privacy of all the records belonging to a user. Wen et al. applied local differential privacy to an federated learning framework for energy-theft detection [286]. Finally, other works combine MPC with DP techniques to achieve better privacy guarantees [142, 272]. Although DP-based learning aims to mitigate inference attacks, it significantly degrades model utility, as training accurate neural network models requires high privacy budgets [229]. As such, DP introduces a privacy vs. utility trade-off that is hard to parameterize; and it is hard to quantify the level of privacy protection that can be achieved with these approaches [141]. To account for these issues, our work employs multiparty homomorphic encryption techniques for achieving private training of neural networks in a distributed setting, where the parties' intermediate updates and the final model remain under encryption. Moreover, it was recently shown that RNNs are particularly vulnerable to inference attacks (e.g., membership inference) compared to traditional neural networks [289].

None of these works, however, address the learning of sequential patterns over the data (our contributions in Chapter 5). A recent study employs secure aggregation with HE for federated traffic-flow prediction [187], but the study remains vulnerable to privacy attacks due to the shared global model.

Private Aggregation of Teacher Ensembles (PATE). A different line of research is focused on the private aggregation of teacher ensembles (PATE) [219, 220], and its variants [190, 292]; the focus is mainly on knowledge transfer by combining differential privacy and generative adversarial networks (GANs). PATE enables the private training of a student model from several teacher models, where the knowledge is transferred in a noisy fashion. However, PATE requires a trusted aggregator, and although it achieves record-level differential privacy guarantees, it remains vulnerable to property inference attacks [199]. Our line of research diverges from these works due to their strong assumption of publicly available datasets and their different designs that predominantly target knowledge transfer. Choquette-Choo et al. recently proposed CaPC Learning that enables confidential and private collaborative learning by relying on MPC, HE, and DP in order to increase the utility of each party's local machine learning model [69]. Their approach works on non-iid settings with heterogeneous model architectures across parties. 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 and

higher privacy guarantees. Our work decouples privacy from the amount of data and the number of parties by relying on MHE.

In summary, each of the aforementioned techniques introduces a different trade-off: DP solutions decrease the utility of the machine learning model — in particular for RNNS [289] — whereas their level of practical privacy protection remains unclear [142]; MPC solutions lack scalability in terms of circuit complexity and number of parties due to high communication overheads [116, 156]; and HE solutions do not scale with the model complexity due to high computational overhead.

2.5 Privacy-Preserving Biomedical Applications

As in PPML solutions, privacy-preserving biomedical applications or genomic data sharing also rely on DP, MPC, or HE [38]. Several solutions in the medical domain rely on DP [70, 177, 161, 21] as an obfuscation technique. Other works also rely on DP for enabling privacy-preserving Genome-Wide Association Studies (GWAS) [257] or survival analysis [43]. DP-based solutions introduce a privacy-accuracy trade-off by perturbing the model parameters which might be unacceptable in the medical domain, whereas our contributions in Chapter 6 decouple the accuracy from the privacy and achieves privacy-by-design with reasonable overhead.

MPC-based solutions [138, 67, 74, 151, 127] often require the parties to communicate their data outside their premises to a limited number of computing nodes; and to protect the data and/or model confidentiality, these solutions assume an honest majority among the computing nodes. Comparatively, our solution in Chapter 6 does not require communicating the data and permits parties to keep their data on their premises, withstanding collusions of up to $N-1$ parties. Smajlović et al. recently propose a framework for developing efficient applications (optimized computation and communication) for biomedical data with honest-but-curious threat model with arbitrary collusions between parties as long as at least one party is honest [258] (similar to our threat model and assumptions). Their solution does not necessarily require parties to communicate their data but suffers from MPC-communication overheads; our solution in Chapter 6 scales efficiently with N parties in terms of communication.

A recent work relies on a combination of HE and MPC to enable secure and federated GWAS by enabling linear mixed models (LMMs) [58]. Similarly, this combination is later used also for secure GWAS by enabling privacy-preserving principal component analysis (PCA) and LMMs in the federated learning scenario [66]. HE-based solutions for privacy-preserving analytics in distributed medical settings [103, 230] also enable functionalities (e.g., basic statistics, counting, or linear regression) different than those in Chapter 6, as such they do not enable the efficient execution of neural networks in a federated learning setting.

3 System Overview

Here, we abstract the system and threat model for POSEIDON, RHODE, and PRICELL, as well as their objectives (Sections 3.1 and 3.2). We provide a high-level description of their functionality (Section 3.3). The model-specific system overviews for each chapter are then described in their respective chapters to emphasize the changes that are introduced to the high-level system model introduced in this section (e.g., aggregation server operations might include additional evaluations such as gradient clipping).

3.1 System and Threat Model

We introduce the system and threat model that are used throughout this thesis below.

System Model. We consider a cross-silo federated learning (FL) setting with a moderate number of parties (N) (typically in the range of 2 to 200 [149]). Each party i has its input data X^i and the corresponding output Y^i ($Y^i \in \{0, 1, \cdots, n\}$ for n-class classification or $Y^i \in \mathbb{R}$ for regression tasks) that they contribute for training a collective training of a neural network model (horizontal federated learning). The structure of X^i depends on the setting, e.g., X^i is time-series data in Chapter 5 for the training of RNNs.

At the end of the training process, the parties use the model to enable predictions-as-a-service (PaaS). A querier q – which can be one of the N parties or an external entity – queries the model and obtains prediction results y_q on its evaluation data X_q. The parties involved in the training process are interested in preserving the privacy of their local data, of the intermediate model updates, and of the resulting model. The querier obtains prediction results on the trained model and keeps its evaluation data confidential. We assume that the parties are interconnected and organized in a tree-structured topology for efficient communication. However, our system is fully distributed and does not assume any hierarchy, thus remaining agnostic of the network topology, e.g., we can consider a fully-connected network.

Threat Model. We consider a *passive-adversary model* (common assumption for cross-silo

settings) with collusions of up to $N-1$ parties (including the querier for PaaS): The parties follow the protocol, but up to $N-1$ parties might share among them their inputs and observations to extract information about the other parties' inputs. Our systems aim at preserving the confidentiality of both the input data and the model during the training and prediction pipelines. We note here that our focus is on mitigating federated learning inference attacks that target the model during training [199, 212, 128, 309, 283]. Inference attacks targeting the system's outputs during prediction, such as model stealing [270], membership [254], or model inversion [98], are out of the scope of this work. We discuss complementary security mechanisms that can be used orthogonally to our systems that can limit the information a querier infers from the prediction results and an extension to the active-adversary model in Appendix B.1.

3.2 Objectives

POSEIDON, RHODE, and PRICELL's main objectives are to enable the privacy-preserving training of and the inference on neural networks in the above system and threat model. Our objective is to protect the parties' and querier's **data confidentiality**, as well as the trained **model confidentiality**, as defined below:

- **Data Confidentiality.** During training and prediction, no party P_i (including the querier P_q) should learn more information about the input data X_j of any other honest party P_j ($j \neq i$, including the querier P_q), other than what can be deduced from its own input data X_i, y_i (or the input X_q and output y_q, for the querier).

- **Model Confidentiality.** During training and prediction, no party P_i (including the querier P_q) should gain more information about the trained model weights, other than what can be deduced from its own input data X_i and the output labels y_i (or X_q, y_q for the querier).

Based on these definitions, we summarize the objectives below:

Training. The parties aim at collaboratively training a neural network model *without* relying on a trusted party and *without* revealing their data to any other party. The objective is to enable this and to mitigate passive federated learning attacks [128, 199, 212, 309] — that target the model or local gradients and that can potentially leak information about the parties' input data — by protecting the confidentiality of the parties' inputs, the model, and any intermediate values, e.g., local or global models, that are communicated in the system. As we optimize the communication via a tree-structured topology, the root data-holder plays the role of the aggregation server in traditional federated learning; and for clarity, throughout the paper, we refer to this node as the *aggregation server* and denote it as P_1.

Prediction. Our objective is to protect the confidentiality of both the querier's data and the

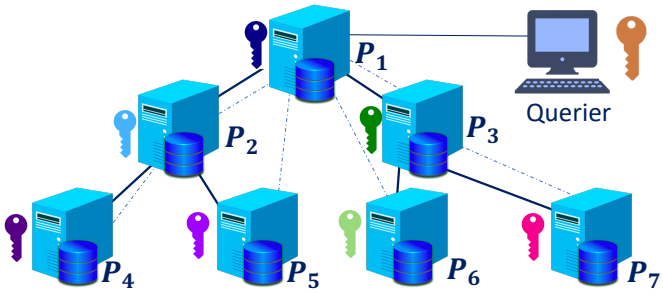


Figure 3.1: *System Model.*

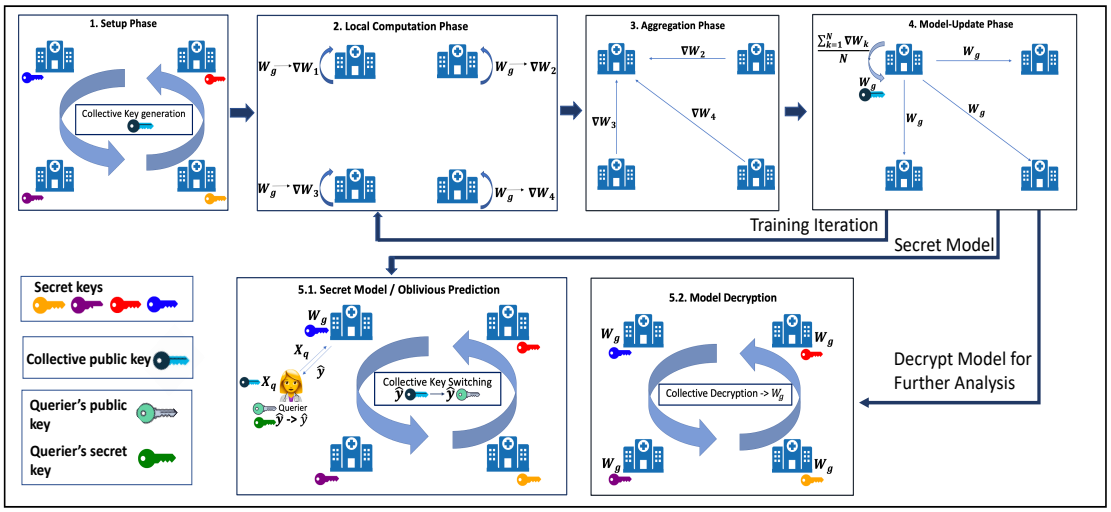


Figure 3.2: POSEIDON, RHODE, *and* PRICELL*'s Training and Prediction Pipelines. Training encapsulates the generation of cryptographic keys and the federated learning iterations on an encrypted model with multiple parties, e.g., healthcare institutions. After training, the model is either kept encrypted or decrypted for further analysis.*

trained neural network model: The parties 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.

3.3 Solution Overview

To fulfill the training and prediction objectives under the given threat model, we rely on federated learning and MHE. To protect data confidentiality and to mitigate the passive federated learning inference attacks during training, we retain the model and all the intermediate values communicated through the network in encrypted form (under the public collective key). However, keeping the model protected implies that all the local neural network operations performed by the parties need to be carried out under homomorphic encryption. To efficiently enable this, we use novel packing strategies for each system, several cryptographic optimizations, and approximations that are detailed in Chapters 4, 5, and 6. We represent the

high-level training and prediction workflows that are used throughout this thesis in Figure 3.2 for a scenario with four parties (healthcare institutions for the example) and describe it in detail in this section.

The operations required for the communication-efficient training of neural networks are enabled by the scheme's computation homomorphic properties, which enables the parties to perform operations between their local data and the encrypted model weights. To enable oblivious prediction on the trained model, we utilize the scheme's key-switching functionality (see Section 1.3 that allows the parties to collectively re-encrypt the prediction results with the querier's public key. We employ several packing schemes to enable SIMD operations on the weights of various neural network layers and use approximations that enable the evaluation of activation functions (e.g., Sigmoid, Softmax, ReLU) under encryption.

To refresh the noise accumulated by the homomorphic operations, we perform bootstrapping ($\mathsf{DBootstrap}(\cdot)$) when required (depending on the network and security parameters) and on appropriate ciphertexts. For clarity, we refer to Algorithm 1, Section 1.2 for a high-level federated learning algorithm. Note that we enable, under encryption, both FedAvg and FedSGD with every party included in each iteration (i.e., $|S| = N$) as we consider a cross-silo setting and given that all parties are available to participate in cryptographic operations (see Appendix B for a possible extension to the asynchronous learning case).

3.3.1 Private Training

We provide a high-level solution for the aforementioned problem and the objectives in Algorithm 2. The chapters of this thesis are different instantiations of this high-level algorithm that are tailored for specific NNs. The bold terms denote encrypted values and $\boldsymbol{W}_{j,i}^{k}$ represents the weight matrix of the j^{th} layer, at iteration k, of the party P_i. When there is no ambiguity or when we refer to the global model, we replace the sub-index i with $\cdot$ and denote weights by $\boldsymbol{W}_{j,\cdot}^{k}$. Similarly, we denote the local gradients at party P_i by $\boldsymbol{\nabla W}_{j,i}^{k}$, for each network layer j and iteration k. Throughout the thesis, the n^{th} row of a matrix that belongs to the i^{th} party is represented by $X_i[n]$ and its encoded (packed) version as $\bar{X}_i[n]$.

The training protocol operates in four phases: **Setup Phase**, **Local Computation Phase**, **Aggregation Phase**, and **Model-Update Phase**. We detail these phases hereunder:
Setup Phase: In this phase, the necessary cryptographic and learning parameters are decided

by the parties, i.e., the minimum security guarantees, the neural network architecture, and the learning parameters, e.g., the learning rate (η), the batch size (b), the number of hidden layers (ℓ), and the number of global (e) and local iterations (l). Then, the parties generate their secret keys (sk_i, for the i^{th} party) and collectively generate the corresponding public key (pk) with the forenamed MHE scheme (Section 1.3). Subsequently, they collectively normalize or standardize their input data with the secure aggregation protocol described in [104]. Each P_i encodes (packs) its input data samples X_i and output labels y_i (see Section 4.3.1) as $\bar{X}_i, \bar{y}_i$.

For the training execution to start, the aggregation server (P_1) initializes the global model, encrypts it with pk, and broadcasts it to all parties.

Local Computation Phase: Each party receives the encrypted global model weights and begins the local gradient descent ($\mathsf{Local\ Gradient\ Descent}(\cdot)$, Line 6, Algorithm 1), i.e., the execution of the forward and backward pass on its local data as detailed in subsequent chapters of this thesis, for an MLP, CNN, and RNN local computations. Note that for different neural network architectures, the local gradient descent algorithm can be altered which is our main contribution in this thesis: to enable different neural network algorithms under encryption.

Aggregate Phase: Parties collectively aggregate their locally computed models (or gradients) in a tree manner. In other words, each party sends its encrypted local model to its parent, and each parent homomorphically sums the received child models with its own encrypted one. As such, the aggregation server receives the sum of all the local models.

Model-Update Phase: The aggregation server updates the global model by averaging the local models of the parties (Line 8, Algorithm 1). As 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 parties for the next Local Computation Phase.

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

Training Termination: In our systems, we stop the learning process, after a predefined number of epochs. However, we note that several early-stop techniques [226] for the neural network training termination can be straightforwardly integrated into our systems (see Appendix B.2).

3.3.2 Predictions-as-a-Service (PaaS)

After the collective model training, the parties can (i) use the encrypted model for oblivious PaaS *without* decryption, or (ii) if required by the application, they can collectively decrypt the model and reveal it to all the parties and/or an external party for semi-oblivious PaaS. In both cases, the querier's evaluation data is encrypted with the parties' collective public key, and we rely on the collective key-switch functionality of the underlying MHE scheme (Section 1.3). In particular, for (i), the querier encrypts X^q with the parties' collective key pk, the parties then compute the prediction on the encrypted data using the encrypted model, and switch the encryption key of the result with $\mathsf{DKeySwitch}(\mathbf{X}^q_{pk}, qk, \{sk_i\})$, where qk is the querier's key. As such, only the querier can decrypt the prediction result. For (ii), the collective decryption is simply a special case of the collective key-switch operation with the encrypted model as input and no target public key. The rest of the evaluation is same as oblivious PaaS.

Algorithm 2 Collective Training

Inputs: X_i, y_i for $i \in \{1,2,...,N\}$

Outputs: $\boldsymbol{W}^e_{1,\cdot}, \boldsymbol{W}^e_{2,\cdot}, \ldots, \boldsymbol{W}^e_{\ell,\cdot}$

Setup Phase:

Parties collectively agree on $\ell, h_1, ..., h_\ell, \eta, \varphi(\cdot), e, b$

Each P_i generates $sk_i \leftarrow \mathsf{SecKeyGen}(1^\lambda)$

Parties collectively generate $pk \leftarrow \mathsf{DKeyGen}(\{sk_i\})$

Each P_i encodes its local data as $\bar{X}_i$, $\bar{y}_i$

P_1 initializes $\boldsymbol{W}^0_{1,\cdot}, \boldsymbol{W}^0_{2,\cdot}, ..., \boldsymbol{W}^0_{\ell,\cdot}$

for $k = 0 \rightarrow e-1$ **do**

Local Computation Phase:

P_1 sends $\boldsymbol{W}^k_{1,\cdot}, \boldsymbol{W}^k_{2\cdot}, ..., \boldsymbol{W}^k_{\ell,\cdot}$ down the tree

Each P_i does:

Local Gradient Descent Computation:

$\nabla \boldsymbol{W}^k_{1,i}, \nabla \boldsymbol{W}^k_{2,i}, \ldots, \nabla \boldsymbol{W}^k_{\ell,i}$

Aggregate Phase:

Parties collectively aggregate: $\nabla \boldsymbol{W}^k_{1,\cdot}, \ldots, \nabla \boldsymbol{W}^k_{\ell,\cdot} \leftarrow \sum_{i=1}^N \nabla \boldsymbol{W}^k_{1,i}, \ldots, \nabla \boldsymbol{W}^k_{\ell,i}$

P_1 obtains $\nabla \boldsymbol{W}^k_{1,\cdot}, \nabla \boldsymbol{W}^k_{2,\cdot}, \ldots, \nabla \boldsymbol{W}^k_{\ell,\cdot}$

Model-Update Phase (performed by P_1) :

for $j = 1 \rightarrow \ell$ **do**

$\boldsymbol{W}^{k+1}_{j,\cdot} += \eta \frac{\nabla \boldsymbol{W}^k_{j,\cdot}}{b \times N}$

end for

end for

4 Federated Multilayer Perceptron and Convolutional Neural Network Learning

4.1 Introduction

In this chapter, we address the privacy-preserving training of and evaluation on MLPs and CNNs in an N-party federated learning setting by relying on MHE to enable the executions under encryption. MLPs and CNNs are crucial for many fields in the industry, e.g., multilayer perceptrons (MLPs) constitute 61% of Tensor Processing Units' workload in Google's datacenters [147]. Consequently, the privacy-preserving solutions that enable their execution are highly desirable. For this, we extend the approach of SPINDLE [102], that was for the generalized linear models in the framework represented in Chapter 3, Section 3.3, and build POSEIDON, a novel system that enables the encrypted training and evaluation of MLPs and CNNs and provides end-to-end protection of the parties' training data, the resulting model, and the evaluation data. Using MHE [209], POSEIDON enables neural network executions with different types of layers, such as fully connected, convolution, and pooling, on a dataset that is distributed among N parties, e.g., a consortium of tens of hospitals, that trust only themselves for the confidentiality of their data and of the resulting model. POSEIDON relies on stochastic gradient descent and protects, from any party, the intermediate updates of the neural network model by maintaining the weights and gradients encrypted throughout the training phase. POSEIDON also enables the resulting encrypted model to be used for privacy-preserving inference on encrypted evaluation data. It provides privacy-preserving inference on the encrypted model using the encrypted data, by leveraging on the key-switching functionality of the underlying crypto scheme that allows for changing the encryption key of a ciphertext.

We evaluate POSEIDON on several real-world datasets and various network architectures and observe that it achieves model performance (test set accuracy levels) on par with centralized or decentralized non-private approaches. Regarding its execution time, we find that POSEIDON trains a 2-layer MLP on a dataset with 23 features and 30,000 samples distributed among 10 parties, in 8.7 minutes. Moreover, POSEIDON trains a 3-layer neural network with 64 neurons per hidden-layer on the MNIST dataset [172] with 784 features and 60K samples

shared between 10 parties, in 1.4 hours, and a neural network with convolutional and pooling layers on the CIFAR-10 [168] dataset (60K samples and 3,072 features) distributed among 50 parties, in 175 hours. Finally, our scalability analysis shows that POSEIDON's computation and communication overhead scales linearly with the number of parties and logarithmically with the number of features and the number of neurons in each layer.

In this chapter, we make the following contributions:

- We present POSEIDON, a novel system for privacy-preserving, quantum-resistant, federated learning-based training of and inference on neural networks with N parties, that relies on multiparty homomorphic encryption and respects the confidentiality of the training data, the model, and the evaluation data.
- We propose an alternating packing approach for the efficient use of single instruction, multiple data (SIMD) operations on encrypted data, and we provide a generic protocol for executing MLPs under encryption, depending on the size of the dataset and the structure of the network.
- We improve the distributed bootstrapping protocol of [209] by introducing arbitrary linear transformations for optimizing computationally heavy operations, such as pooling or a large number of consecutive rotations on ciphertexts.
- We formulate a constrained optimization problem for choosing the cryptographic parameters and for balancing the number of costly cryptographic operations required for training and evaluating neural networks in a distributed setting.
- We evaluate POSEIDON against centralized and decentralized non-private solutions using various datasets and evaluate its scalability in terms of the number of parties and different neural network parameters, e.g., the number of neurons in each layer.

To the best of our knowledge, POSEIDON is the first system that enables quantum-resistant distributed learning on neural networks with N parties in a federated learning setting, and that preserves the privacy of the parties' confidential data, the intermediate model updates, and the final model weights.

Acknowledgements. This work is a result of a collaboration with Dr. Apostolos Pyrgelis, Dr. Juan R. Troncoso- Pastoriza, Dr. David Froelicher, Jean-Phillipe Bossuat, Joao Sa Sousa, and Prof. Jean-Pierre Hubaux.

4.2 POSEIDON Design

The system and threat model of POSEIDON is described in Chapter 3, Section 3.1. We also present the high-level federated learning algorithm in Chapter 1, Algorithm 1 for the collective training and our encrypted version of it in Chapter 3, Section 3.3 in Algorithm 2. As introduced

in Chapter 3, POSEIDON's training pipeline includes four phases: **Setup Phase**, **Local Computation Phase**, **Aggregation Phase**, and **Model-Update Phase**; and PaaS after the training is used for evaluating the predictions to the querier. We remind here that POSEIDON enables the computations under encryption by keeping the model in encrypted form throughout the execution pipeline. Below, we define the system-specific **Local Gradient Descent Computation** that refers to Line 9 of the Algorithm 2 and the weight initialization.

Local Gradient Descent (LGD) Computation: Each P_i performs b forward and backward passes to compute and aggregate the local gradients, by processing each sample of its respective batch. Algorithm 3 describes the LGD steps performed by each party P_i for an MLP with l layers, at iteration k; $\odot$ represents the element-wise product and $\varphi'(\cdot)$ the derivative of an activation function. As the algorithm refers to one local iteration for a specific party, we omit k and i from the weight and gradient indices. This algorithm describes the typical operations for the forward and backward pass using gradient descent with the $L2$ loss. The computation for each layer j in the forward pass comprises a multiplication of current layer values with the weight matrix ($\boldsymbol{U}_j = \boldsymbol{L}_{j-1} \times \boldsymbol{W}_j$) and the activation function that is computed on this result ($\varphi(\boldsymbol{U}_j)$). Then the backpropagation calculates the accumulated error for each layer. We note that the operations in this algorithm are performed over encrypted data.

Weight Initialization. To avoid exploding or vanishing gradients, we rely on commonly used techniques: (i) Xavier initialization [113] for the sigmoid or tanh activated layers: $W_j = U(\frac{-1}{\sqrt{h_{j-1}}}, \frac{1}{\sqrt{h_{j-1}}})$ where U is the uniform distribution and h_{j-1} is the size of the previous layer, and (ii) He initialization [123] for ReLU activated layers, where the Xavier-initialized weights are multiplied twice by their variance.

Algorithm 3 Local Gradient Descent (LGD) Computation (MLP)

Inputs: $\boldsymbol{W}^k_{1,\cdot}, \boldsymbol{W}^k_{2,\cdot}, \ldots, \boldsymbol{W}^k_{\ell,\cdot}$
Outputs: $\nabla\boldsymbol{W}^k_{1,i}, \nabla\boldsymbol{W}^k_{2,i}, \ldots, \nabla\boldsymbol{W}^k_{\ell,i}$. Note that i and k indices are omitted in this protocol.

1: **for** $t = 1 \rightarrow b$ **do** ▷ Forward Pass
2: $L_0 = \bar{X}[t]$
3: **for** $j = 1 \rightarrow \ell$ **do**
4: $\boldsymbol{U}_j = \boldsymbol{L}_{j-1} \times \boldsymbol{W}_j$
5: $\boldsymbol{L}_j = \varphi(\boldsymbol{U}_j)$
6: **end for**
7: $\boldsymbol{E}_\ell = \bar{y}[t] - \boldsymbol{L}_\ell$ ▷ Backpropagation
8: $\boldsymbol{E}_\ell = \varphi'(\boldsymbol{U}_\ell) \odot \boldsymbol{E}_\ell$
9: $\boldsymbol{\nabla W}_\ell += \boldsymbol{L}^{\boldsymbol{T}}_{\ell-1} \times \boldsymbol{E}_\ell$
10: **for** $j = \ell - 1 \rightarrow 1$ **do**
11: $\boldsymbol{E}_j = \boldsymbol{E}_{j+1} \times \boldsymbol{W}^{\boldsymbol{T}}_{j+1}$
12: $\boldsymbol{E}_j = \varphi'(\boldsymbol{U}_j) \odot \boldsymbol{E}_j$
13: $\boldsymbol{\nabla W}_j += \boldsymbol{L}^{\boldsymbol{T}}_{j-1} \times \boldsymbol{E}_j$
14: **end for**
15: **end for**

4.3 Cryptographic Operations and Optimizations

We first present the alternating packing (AP) approach that we use for packing the weight matrices (Section 4.3.1). We then explain how we enable activation functions on encrypted values (Section 4.3.2) and introduce the cryptographic building blocks and functions employed in POSEIDON (Section 4.3.3), together with their execution pipeline and their complexity (Sections 4.3.4 and 4.3.5). Finally, we formulate a constrained optimization problem that depends on a cost function for choosing the parameters of the cryptoscheme (Section 4.3.6).

4.3.1 Alternating Packing (AP) Approach

For the efficient computation of the forward pass and backpropagation described in Algorithm 3, we rely on the packing capabilities of the cryptoscheme that enables Single Instruction, Multiple Data (SIMD) operations on ciphertexts. Packing enables coding a vector of values in a ciphertext and to parallelize the computations across its different slots, thus significantly improving the overall performance.

We observe that the existing packing strategies for enabling secure distributed machine learning in SPINDLE [102], which are row-based [159] and diagonal approach [119], require a high number of rotations for the execution of the matrix-matrix multiplications and matrix transpose operations, performed during the forward and backward pass of the local gradient descent computation (see Algorithm 3). We here remark that the number of rotations has a significant effect on the overall training time of a neural network on encrypted data, as they require costly key-switch operations (see Section 4.3.5). As an example, the diagonal approach scales linearly with the size of the weight matrices, when it is used for batch-learning of neural networks, due to the matrix transpose operations in the backpropagation. We follow a different packing approach and process each batch sample one by one, making the execution embarrassingly parallelizable. This enables us to optimize the number of rotations, to eliminate the transpose operation applied to matrices in the backpropagation, and to scale logarithmically with the dimension and number of neurons in each layer.

We propose an "alternating packing (AP) approach" that combines row-based and column-based packing, i.e., rows or columns of the matrix are vectorized and packed into one ciphertext. In particular, the weight matrix of every FC layer in the network is packed following the opposite approach from that used to pack the weights of the previous layer. With the AP approach, the number of rotations scales logarithmically with the dimension of the matrices, i.e., the number of features (d), and the number of hidden neurons in each layer (h_i). To enable this, we pad the matrices with zeros to get power-of-two dimensions. In addition, the AP approach reduces the cost of transforming the packing between two consecutive layers.

Algorithm 4 describes a generic way for the initialization of encrypted weights for an ℓ-layer MLP by P_1 and for the encoding of the input matrix (X_i) and labels (y_i) of each party P_i. It takes as inputs the neural network parameters: the dimension of the data (d) that describes

the shape of the input layer, the number of hidden neurons in the j^{th} layer (h_j), and the number of outputs (h_ℓ). We denote by gap a vector of zeros, and by $|\cdot|$ the size of a vector or the number of rows of a matrix. $\mathsf{Replicate}(v, k, gap)$ returns a vector that replicates v, k times with a gap in between each replica. $\mathsf{Flatten}(W, gap, dim)$, flattens the rows or columns of a matrix W into a vector and introduces gap in between each row/column. If a vector is given as input to this function, it places gap in between all of its indices. The argument dim indicates flattening of rows ('r') or columns ('c') and $dim = '\cdot'$ for the case of vector inputs.

We observe that the rows (or columns) packed into one ciphertext, must be aligned with the rows (or columns) of the following layer for the next layer multiplications in the forward pass and for the alignment of multiplication operations in the backpropagation, as depicted in Table 4.1 (e.g., see steps F1, F6, B3, B5, B6). We enable this alignment by adding gap between rows or columns and using rotations, described in the next section. Note that these steps correspond to the weight initialization and to the input preparation steps of the **Setup phase**.

Convolutional Layer Packing. To optimize the SIMD operations for convolutional (CV) layers, we decompose the n^{th} input sample $X_i[n]$ into t smaller matrices according to the kernel size $h = f \times f$. We pack these decomposed flattened matrices into one ciphertext, with a gap in between each matrix that is defined based on the number of neurons in the next layer ($h_2 - h_1$), similarly to the AP approach. The weight matrix is then replicated t times with the same gap between each replica. If the next layer is another convolutional or downsampling layer, the gap is not needed and the values in the slots are re-arranged during the training execution (see Section 4.3.3). Lastly, we introduce the average-pooling operation to our bootstrapping function ($\mathsf{DBootstrapALT}(\cdot)$, see Section 4.3.3), and we re-arrange almost for free the slots for any CV layer that comes after average-pooling.

We note that high-depth kernels, i.e., layers with a large number of kernels, require a different packing optimization. In this case, we alternate row and column-based packing (similar to the AP approach), replicate the decomposed matrices, and pack all kernels in one ciphertext. This approach introduces k multiplications in the **Local Computation Phase**, where k is the number of kernels in that layer, and comes with reduced communication overhead; the latter would be k times larger for **Aggregate Phase**, **Local Computation Phase**, and $\mathsf{DBootstrap}(\cdot)$, if the packing described in the previous paragraph was employed.

Downsampling (Pooling) Layers. As there is no weight matrix for downsampling layers, they are not included in the offline packing phase. The cryptographic operations for pooling are described in Section 4.3.4.

4.3.2 Approximated Activation Functions

For the encrypted evaluation of non-linear activation functions, such as Sigmoid or Softmax, we use least-squares approximations and rely on the optimized polynomial evaluation that, as described in [102], consumes $\lceil \log(d_a + 1) \rceil$ levels for an approximation degree d_a. For

Algorithm 4 Alternating Packing (AP) Protocol

Inputs: $X_i, y_i, d, \{h_1, h_2, ..., h_\ell\}, \ell$
Outputs: $\boldsymbol{W}^0_{1,\cdot}, \boldsymbol{W}^0_{2,\cdot}, ..., \boldsymbol{W}^0_{\ell,\cdot}, \bar{X}_i, \bar{y}_i$

for $i = 1 \rightarrow N$ each P_i **do**
  Initialize $|gap| = max(h_1 - d, 0)$ ▷ Input Preparation
  for $n = 1 \rightarrow |X_i|$ **do**
    $X_i[n] = \mathsf{Replicate}(X_i[n], h_1, gap)$
    $\bar{X}_i[n] = \mathsf{Encode}(X_i[n])$
  end for
  if $\ell\%2! = 0$ **then** ▷ Labels Preparation
    Initialize $|gap| = h_\ell$
    $y_i = \mathsf{Flatten}(y_i, gap, '\cdot')$
  end if
  $\bar{y}_i = \mathsf{Encode}(y_i)$
  if i==1 **then** ▷ P_1 performs Weight Initialization:
    Initialize $W^0_{1,\cdot}, W^0_{2,\cdot}, ..., W^0_{\ell,\cdot}$
    for $j = 1 \rightarrow \ell$ **do**
      if $j\%2 == 0$ **then** ▷ Row Packing
        if $h_{j-2} > h_j$ **then**
          Initialize $|gap| = h_{j-2} - h_j$
        end if
        $W^0_{j,\cdot} = \mathsf{Flatten}(W^0_{j,\cdot}, gap, 'r')$
        $\boldsymbol{W}^0_{j,\cdot} = \mathsf{Enc}(pk, W^0_{j,\cdot})$
      else ▷ Column Packing
        if $h_{j+1} > h_{j-1}$ **then**
          Initialize $|gap| = h_{j+1} - h_{j-1}$
        end if
        $W^0_{j,\cdot} = \mathsf{Flatten}(W^0_{j,\cdot}, gap, 'c')$
        $\boldsymbol{W}^0_{j,\cdot} = \mathsf{Enc}(pk, W^0_{j,\cdot})$
      end if
    end for
  end if
end for

the piece-wise function ReLU, we approximate the smooth approximation of ReLU, softplus (SmoothReLU), $\varphi(x) = \ln(1 + e^x)$ with least-squares. Lastly, we use derivatives of the approximated functions. We discuss possible alternatives to these approximations in Appendix C.2.

To achieve better approximation with the lowest possible degree, we apply two approaches to keep the input range of the activation function as small as possible, by using (i) different weight initialization techniques for different layers (i.e., Xavier or He initialization), and (ii) collective normalization of the data by sharing and collectively aggregating statistics on each party's local data in a privacy-preserving way [104]. Finally, the interval and the degree of the approximations are chosen based on the heuristics on the data distribution in a privacy-preserving way, as described in [126].

	AP Approach	Representation
PREPARE:		
1. Each P_i prepares $X_i[n], y_i[n]$	Encode $X_i[n]$, $y_i[n] \rightarrow \tilde{X}_i[n]$, $\tilde{y}_i[n]$ $\tilde{L}_0 = \tilde{X}_i[n]$	
2. P_1 initializes $W_{1,\cdot}$	Vectorize columns, pack with $\lvert gap\rvert = 0$ $\boldsymbol{W}^0_{1,\cdot} = \mathsf{Flatten}(W^0_{1,\cdot}, gap, 'c')$	
3. P_1 initializes $W_{2,\cdot}$	Vectorize rows, pack with $\lvert gap\rvert = d - h_\ell$ $\boldsymbol{W}^0_{2,\cdot} = \mathsf{Flatten}(W^0_{2,\cdot}, gap, 'r')$	
4. Each P_i generates masks $\tilde{m}_1, \tilde{m}_2$	$\tilde{m}_1 = [1,0,0,0,1,0,0,0,1,0,0,0,1,...]$ $\tilde{m}_2 = [1,1,0,0,0,0,0,0,0,0,0,0,0,...]$	
Forward Pass (Each P_i):		
1. $\boldsymbol{U}_1 = \tilde{L}_0 \times \boldsymbol{W}_{1,\cdot}$ 2. $\boldsymbol{L}_1 = \varphi(\boldsymbol{U}_1)$	F1. $\boldsymbol{U}_1 = \mathsf{Mul}_{\mathsf{pt}}(\tilde{L}_0, \boldsymbol{W}_{1,\cdot})$, $\mathsf{Res}(\boldsymbol{U}_1)$ F2. $\boldsymbol{U}_1 = \mathsf{RIS}(\boldsymbol{U}_1, 1, d)$ F3. $\boldsymbol{U}_1 = \mathsf{Mul}_{\mathsf{pt}}(\boldsymbol{U}_1, \tilde{m}_1)$, $\mathsf{Res}(\boldsymbol{U}_1)$ F4. $\boldsymbol{U}_1 = \mathsf{RR}(\boldsymbol{U}_1, 1, h_\ell)$ F5. $\boldsymbol{L}_1 = \varphi(\boldsymbol{U}_1)$	
3. $\boldsymbol{U}_2 = \boldsymbol{L}_1 \times \boldsymbol{W}_{2,\cdot}$ 4. $\boldsymbol{L}_2 = \varphi(\boldsymbol{U}_2)$	F6. $\boldsymbol{U}_2 = \mathsf{Mul}_{\mathsf{ct}}(\boldsymbol{L}_1, \boldsymbol{W}_{2,\cdot})$, $\mathsf{Res}(\boldsymbol{L}_2)$ F7. $\boldsymbol{U}_2 = \mathsf{RIS}(\boldsymbol{U}_1, d, h_1)$ F8. $\boldsymbol{L}_2 = \mathsf{Mul}_{\mathsf{pt}}(\boldsymbol{L}_2, \tilde{m}_2)$, $\mathsf{Res}(\boldsymbol{L}_2)$ F9. $\mathsf{DBootstrap}(\boldsymbol{U}_2)$ F10. $\boldsymbol{L}_2 = \varphi(\boldsymbol{U}_2)$	
Backpropagation (Each P_i):		
1. $\boldsymbol{E}_2 = \tilde{y}_i[n] - \boldsymbol{L}_2$	B1. $\boldsymbol{E}_2 = \mathsf{Sub}(\tilde{y}_i[n], \boldsymbol{L}_2)$	
2. $\boldsymbol{E}_2 = (\varphi'(\boldsymbol{U}_2)) \odot \boldsymbol{E}_2$	B2. $\boldsymbol{d} = \varphi'(\boldsymbol{U}_2)$ B3. $\boldsymbol{E}_2 = \mathsf{Mul}_{\mathsf{ct}}(\boldsymbol{E}_2, \boldsymbol{d})$, $\mathsf{Res}(\boldsymbol{E}_2)$ B4. $\boldsymbol{E_2} = \mathsf{RR}(\boldsymbol{E_2}, d, h_1)$	
3. $\nabla \boldsymbol{W}_{2,i} = \boldsymbol{L}_1^T \times \boldsymbol{E}_l$	B5. $\nabla \boldsymbol{W}_{2,i} = \mathsf{Mul}_{\mathsf{ct}}(\boldsymbol{L}_1, \boldsymbol{E}_2)$, $\mathsf{Res}(\nabla \boldsymbol{W}_{2,i})$	
4. $\boldsymbol{E}_1 = \boldsymbol{E}_2 \times \boldsymbol{W}^{\boldsymbol{T}}_{2,\cdot}$	B6. $\boldsymbol{E}_1 = \mathsf{Mul}_{\mathsf{ct}}(\boldsymbol{E}_2, \boldsymbol{W}_{2,i})$, $\mathsf{Res}(\boldsymbol{E}_1)$ B7. $\boldsymbol{E}_1 = \mathsf{RIS}(\boldsymbol{E}_1, 1, h_\ell)$	
5. $\boldsymbol{E}_1 = (\varphi'(\boldsymbol{U}_1) \odot \boldsymbol{E}_1)$	B8. $\boldsymbol{d} = \varphi'(\boldsymbol{U}_1)$ B9. $\boldsymbol{d} = \mathsf{Mul}_{\mathsf{pt}}(\boldsymbol{d}, \tilde{m}_1)$ B10. $\boldsymbol{E}_1 = \mathsf{Mul}_{\mathsf{ct}}(\boldsymbol{E}_1, \boldsymbol{d})$, $\mathsf{Res}(\boldsymbol{E}_1)$ B11. $\mathsf{DBootstrapALT}(\boldsymbol{E}_1)$	
6. $\nabla \boldsymbol{W}_{1,i} = \tilde{L}_0^T \times \boldsymbol{E}_1$	B12. $\boldsymbol{E}_1 = \mathsf{Mul}_{\mathsf{pt}}(\boldsymbol{E}_1, \tilde{m}_1)$, $\mathsf{Res}(\boldsymbol{E_1})$ B13. $\boldsymbol{E_1} = \mathsf{RR}(\boldsymbol{E}_1, 1, d)$ B14. $\nabla \boldsymbol{W}_{1,i} = \mathsf{Mul}_{\mathsf{pt}}(\tilde{L}_0, \boldsymbol{E}_1)$, $\mathsf{Res}(\nabla \boldsymbol{W}_{1,i})$	
Update (at P_1): 1. $\boldsymbol{W}_{j,\cdot} += \eta \frac{\nabla \boldsymbol{W}_{j,\cdot}}{b \times N}$ $\forall\, j \in \{1,2,..,l\}$	U1. $\mathsf{SetScale}(\nabla \boldsymbol{W}_{j,\cdot}, S_{\nabla \boldsymbol{W}_{j,\cdot}} \times (b \times N))/\eta)$ U2. $\boldsymbol{W}_{j,\cdot} = \mathsf{Add}(\boldsymbol{W}_{j,\cdot}, \nabla \boldsymbol{W}_{j,\cdot})$ U3. $\mathsf{DBootstrap}(\boldsymbol{W}_{j,\cdot})$	

Table 4.1: *Execution pipeline of a toy example with 2-layer MLP network. The left-most column shows the operations for each step, the middle column present the AP approach and the respective homomorphic operations for each MLP step. The right-most column presents the effect of each operation on the slots of the ciphertext. Orange steps indicate the operations introduced to DBootstrapALT(·).*

4.3.3 Cryptographic Building Blocks

We present each cryptographic function that we employ to enable the privacy-preserving training of neural networks with N parties. We also discuss the optimizations employed to avoid costly transpose operations in the encrypted domain.
Rotations. As we rely on packing capabilities, computation of the inner-sum of vector-matrix

multiplications and transpose operation implies a restructuring of the vectors, that can only be achieved by applying slot rotations. Throughout this chapter, we use two types of rotation functions: (i) Rotate For Inner Sum ($\mathsf{RIS}(\boldsymbol{c}, p, s)$) is used to compute the inner-sum of a packed vector $\boldsymbol{c}$ by homomorphically rotating it to the left with $\mathsf{RotL}(\boldsymbol{c}, p)$ and by adding it to itself iteratively $\log_2(s)$ times, and (ii) Rotate For Replication ($\mathsf{RR}(\boldsymbol{c}, p, s)$) replicates the values in the slots of a ciphertext by rotating the ciphertext to the right with $\mathsf{RotR}(\boldsymbol{c}, p)$ and by adding to itself, iteratively $\log_2(s)$ times. For both functions, p is multiplied by two at each iteration, thus both yield $\log_2(s)$ rotations. As rotations are costly cryptographic functions (see Table 4.2), and the matrix operations required for neural network training require a considerable amount of rotations, we minimize the number of executed rotations by leveraging a modified bootstrapping operation, that automatically performs some of the required rotations.

Distributed Bootstrapping with Arbitrary Linear Transformations. To execute the high-depth homomorphic operations required for training neural networks, bootstrapping is required several times to refresh a ciphertext, depending on the initial level L. In POSEIDON, we use a distributed version of bootstrapping [209], as it is several orders of magnitude more efficient than the traditional centralized bootstrapping. Then we modify it, to leverage on the interaction to automatically perform some of the rotations, or pooling operations, embedded as transforms in the bootstrapping.

Mouchet et al. replace the expensive bootstrap circuit by a one-round protocol where the parties collectively switch a Brakerski/Fan-Vercauteren (BFV) [92] ciphertext to secret-shares in $\mathbb{Z}_t^{\mathcal{N}}$. Since the BFV encoding and decoding algorithms are linear transformations, they can be performed without interaction on a secret-shared plaintext. Despite its properties, the protocol that Mouchet et al. propose for the BFV scheme cannot be directly applied to CKKS, as CKKS is a *leveled* scheme): The re-encryption process extends the residue number system (RNS) basis from Q_ℓ to Q_L. Modular reduction of the masks in Q_ℓ will result in an incorrect encryption. Our solution to this limitation is to collectively switch the ciphertext to a secret-shared plaintext with statistical indistinguishability.

We define this protocol as $\mathsf{DBootstrapALT}(\cdot)$ (Algorithm 5) that takes as inputs a ciphertext $\boldsymbol{c}_{pk}$ at level ℓ encrypting a message msg and returns a ciphertext $\boldsymbol{c}'_{pk}$ at level L encrypting $\phi(msg)$, where $\phi(\cdot)$ is a linear transformation over the field of complex numbers. We denote by $||a||$ the infinity norm of the vector or polynomial a. As the security of the RLWE is based on computational indistinguishability, switching to the secret-shared domain does not hinder security. We refer to Appendix C.4 for technical details and the security proof of our protocol.

Algorithm 5 DBootstrapALT$(\cdot)$

Inputs: $\boldsymbol{c}_{pk} = (c_0, c_1) \in R^2_{Q_\ell}$ encrypting msg, λ a security parameter, $\phi(\cdot)$ a linear transformation over the field of complex numbers, a a common reference polynomial, s_i the secret-key of each party P_i, χ_{err} a distribution over R, where each coefficient is independently sampled from Gaussian distribution with the standard deviation $\sigma = 3.2$, and bound $\lfloor 6\sigma \rfloor$.

Constraints: $Q_\ell > (N+1) \cdot ||msg|| \cdot 2^\lambda$.

Outputs: $\boldsymbol{c}'_{pk} = (c'_0, c'_1) \in R^2_{Q_L}$

1: **for all** P_i **do**
2: $M_i \leftarrow R_{||msg||\cdot 2^\lambda}$, $e_{0,i}, e_{1,i} \leftarrow \chi_{err}$
3: $M'_i \leftarrow \mathsf{Encode}(\phi(\mathsf{Decode}(M_i)))$
4: $h_{0,i} \leftarrow s_i c_1 + M_i + e_{0,i} \mod Q_\ell$
5: $h_{1,i} \leftarrow -s_i a - M_i + e_{1,i} \mod Q_L$
6: **end for**
7: $h_0 \leftarrow \sum h_{0,i}, h_1 \leftarrow \sum h_{1,i}$
8: $c'_0 \leftarrow \mathsf{Encode}(\phi(\mathsf{Decode}(c_0 + h_0 \mod Q_\ell)))$
9: **return** $\boldsymbol{c}'_{pk} = (c'_0 + h_1 \mod Q_L, a) \in R^2_{Q_L}$

Optimization of the Vector-Transpose Matrix Product. The backpropagation step of the local gradient computation at each party requires several multiplications of a vector (or matrix) with the transposed vector (or matrix) (see Lines 11-13 of Algorithm 3). The naïve multiplication of a vector $\boldsymbol{v}$ with a transposed weight matrix $\boldsymbol{W}^T$ that is fully packed in one ciphertext, requires converting $\boldsymbol{W}$ of size $g \times k$, from column-packed to row-packed. This is equivalent to applying a permutation of the plaintext slots, that can be expressed with a plaintext matrix $W_{gk \times gk}$ and homomorphically computed by doing a matrix-vector multiplication. As a result, a naïve multiplication requires $\sqrt{g \times k}$ rotations followed by $\log_2(k)$ rotations to obtain the inner sum from the matrix-vector multiplication. We propose several approaches to reduce the number of rotations when computing the multiplication of a packed matrix (to be transposed) and a vector: **(i)** For the mini-batch gradient descent, we do not perform operations on the batch matrix. Instead, we process each batch sample in parallel, because having separate vectors (instead of a matrix that is packed into one ciphertext) enables us to reorder them at a lower cost. This approach translates ℓ matrix transpose operations to be transposes in vectors (the transpose of the vectors representing each layer activations in the backpropagation, see Line 13, Algorithm 3), **(ii)** Instead of taking the transpose of $\boldsymbol{W}$, we replicate the values in the vector that will be multiplied with the transposed matrix (for the operation in Line 11, Algorithm 3), leveraging the gaps between slots with the AP approach. That is, for a vector $\boldsymbol{v}$ of size k and the column-packed matrix $\boldsymbol{W}$ of size $g \times k$, $\boldsymbol{v}$ has the form $[a, 0, 0, 0 \ldots, b, 0, 0, 0, \ldots, c, 0, 0, 0, \ldots]$ with at least k zeros in between values (due to Algorithm 4). Hence, any resulting ciphertext requiring the transpose of the matrix that will be subsequently multiplied, will also include gaps in between values. We apply $\mathsf{RR}(\boldsymbol{v}, 1, k)$ that consumes $\log_2(k)$ rotations to generate $[a, a, a, \ldots 0 \ldots, b, b, b, .., 0 \ldots, c, c, c, \ldots, 0, \ldots]$. Finally, we compute the product $\mathscr{P} = \mathsf{Mul}_{\mathsf{ct}}(\boldsymbol{v}, \boldsymbol{W})$ and apply $\mathsf{RIS}(\mathscr{P}, 1, g)$ to get the inner sum with $\log_2(g)$ rotations, and **(iii)** We further optimize the performance by using DBootstrapALT$(\cdot)$ (Algorithm 5): If the ciphertext before the multiplication must be bootstrapped, we embed the $\log_2(k)$ rotations as a linear transformation

performed during the bootstrapping.

4.3.4 Execution Pipeline

Table 4.1 depicts the pipeline of the operations for processing one sample in LGD computation for a 2-layer MLP. These steps can be extended to an ℓ-layer MLP by following the same operations for multiple layers. The weights are encoded and encrypted using the AP approach, and the shape of the packed ciphertext for each step is shown in the representation column. Each forward and backward pass on a layer in the pipeline consumes one Rotate For Inner Sum ($\mathsf{RIS}(\cdot)$) and one Rotate For Replication ($\mathsf{RR}(\cdot)$) operation, except for the last layer, as the labels are prepared according to the shape of the ℓ^{th} layer output. In Table 4.1, we assume that the initial level $L = 7$, which is a typical value that is frequently used in our experimental evaluations. When a bootstrapping function is followed by a masking (that is used to eliminate unnecessary values during multiplications) and/or several rotations, we perform these operations embedded as part of the distributed bootstrapping ($\mathsf{DBootstrapALT}(\cdot)$) to minimize their computational cost. The steps highlighted in orange are the operations embedded in the $\mathsf{DBootstrapALT}(\cdot)$. The complexity of each cryptographic function is analyzed in Section 4.3.5.

Convolutional Layers. As we flatten, replicate, and pack the kernel in one ciphertext, a CV layer follows the exact same execution pipeline as a FC layer. However, the number of $\mathsf{RIS}(\cdot)$ operations for a CV layer is smaller than for a FC layer. That is because the kernel size is usually smaller than the number of neurons in a FC layer. For a kernel of size $h = f \times f$, the inner sum is calculated by $log_2(f)$ rotations. Note that when a CV layer is followed by a FC layer, the output of the i^{th} CV layer (L_i) already gives the flattened version of the matrix in one ciphertext. We apply $\mathsf{RR}(\boldsymbol{L}_i, 1, h_{i+1})$ for the preparation of the next layer multiplication. When a CV layer is followed by a pooling layer, however, the $\mathsf{RR}(\cdot)$ operation is not needed, as the pooling layer requires a new arrangement of the slots of $\boldsymbol{L}_i$. We avoid this costly operation by passing $\boldsymbol{L}_i$ to $\mathsf{DBootstrapALT}(\cdot)$, and by embedding both the pooling and its derivative in $\mathsf{DBootstrapALT}(\cdot)$.

Pooling Layers. In POSEIDON, we evaluate our system based on average pooling as it is the most efficient type of pooling that can be evaluated under encryption [112]. To do so, we exploit our modified collective bootstrapping to perform arbitrary linear transformations. Indeed, the average pooling is a linear function, and so is its derivative (note that this is not the case for the max pooling). Therefore, in the case of a CV layer followed by a pooling layer, we apply $\mathsf{DBootstrapALT}(\cdot)$ and use it both to rearrange the slots and to compute the convolution of the average pooling in the forward pass and its derivative, that is used later in the backward pass. For a $h = f \times f$ kernel size, this saves $\log_2(h)$ rotations and additions ($\mathsf{RIS}(\cdot)$) and one level if masking is needed. For max/min pooling, which are non-linear functions, we refer the reader to Appendix C.3 and highlight that evaluating these functions by using encrypted arithmetic remains impractical due to the need of high-precision approximations.

	Computational Complexity	#Levels Used	Communication	Rounds
FORWARD P. (FP)	$(\log_2(h_{i-1})+\log_2(h_{i+1}))\cdot \mathsf{KS}+\mathsf{Mul}_{\mathsf{ct}}+\mathsf{Mul}_{\mathsf{pt}}+\varphi$	$2+\lceil\log_2(d_a+1)\rceil$	–	–
BACKWARD P. (BP)	$(\log_2(h_{i-1})+\log_2(h_{i+1}))\cdot \mathsf{KS}+2\mathsf{Mul}_{\mathsf{ct}}+\mathsf{Mul}_{\mathsf{pt}}+\varphi'$	$3+\lceil\log_2(d_a)\rceil$	–	–
Local Computation Phase	$\ell(\mathsf{FP}+\mathsf{BP})-2\log_2(h_\ell)$	$\ell(5+\lceil\log_2(d_a+1)+\lceil\log_2(d_a)\rceil)$	$z(N-1)\lvert c\rvert$	1/2
Aggregate Phase	–	–	$z(N-1)\lvert c\rvert$	1/2
Model-Update Phase	$\ell(\mathsf{Mul}_{\mathsf{pt}}+\mathsf{DB})$	–	–	–
DBootstrap (DB)	$N\log_2(N)(L+1)+N\log_2(N)(L_{\boldsymbol{c}}+1)$	–	$(N-1)\lvert c\rvert$	1
Mul Plaintext ($\mathsf{Mul}_{\mathsf{pt}}$)	$2N(L_{\boldsymbol{c}}+1)$	1	–	–
Mul Ciphertext ($\mathsf{Mul}_{\mathsf{ct}}$)	$4N(L_{\boldsymbol{c}}+1)+\mathsf{KS}$	1	–	–
Approx. Activation Function (φ)	$(2^\kappa+\mathrm{m}-\kappa-3+\lceil(d_a+1)/2^\kappa\rceil)\cdot\mathsf{Mul}_{\mathsf{ct}}$	$\lceil\log_2(d_a+1)\rceil$	–	–
$\mathsf{RIS}(\boldsymbol{c},p,s)$, $\mathsf{RR}(\boldsymbol{c},p,s)$	$\log_2(s)\cdot\mathsf{KS}$	–	–	–
Key-switch (KS)	$\tilde{\mathcal{O}}(\mathcal{N}\log_2(\mathcal{N})L_{\boldsymbol{c}}\beta)$	–	–	–

Table 4.2: *Complexity analysis of* POSEIDON*'s building blocks.* $\mathcal{N},\alpha,L,L_{\boldsymbol{c}},d_a$ *stand for the cyclotomic ring size, the number of secondary moduli used during the key-switching, maximum level, current level, and the approximation degree, respectively.* $\beta=\lceil L_{\boldsymbol{c}}+1/\alpha\rceil$, $m=\lceil\log(d_a+1)\rceil$, $\kappa=\lfloor m/2\rfloor$.

4.3.5 Complexity Analysis

Table 4.2 displays the communication and *worst-case* computational complexity of POSEIDON's building blocks. This includes the MHE primitives, thus facilitating the discussion on the parameter selection in the following section. We define the complexity in terms of key-switch $\mathsf{KS}(\cdot)$ operations and recall that this is a different operation than $\mathsf{DKeySwitch}(\cdot)$, as explained in Section 1.3. We note that $\mathsf{KS}(\cdot)$ and $\mathsf{DBootstrap}(\cdot)$ are 2 orders of magnitude slower than an addition operation, rendering the complexity of an addition negligible.

We observe that POSEIDON's communication complexity depends solely on the number of parties (N), the number of total ciphertexts sent in each global iteration (z), and the size of one ciphertext ($|\boldsymbol{c}|$). The building blocks that do not require communication are indicated with a dash $-$.

In Table 4.2, forward and backward passes represent the per-layer complexity for FC layers, so they are an *overestimate* for CV layers. Note that the number of multiplications differs in a forward pass and a backward pass, depending on the packing scheme, e.g., if the current layer is row-packed, it requires 1 less $\mathsf{Mul}_{\mathsf{ct}}(\cdot)$ in the backward pass, and we have 1 less $\mathsf{Mul}_{\mathsf{pt}}(\cdot)$ in several layers, depending on the masking requirements. Furthermore, the last layer of forward pass and the first layer of backpropagation take 1 less $\mathsf{RR}(\cdot)$ operation that we gain from packing the labels in the offline phase, depending on the neural network structure (see Algorithm 4). Hence, we save $2\log_2(h_\ell)$ rotations per one LGD computation.

In the **Local Computation Phase**, we provide the complexity of the local computations per P_i, depending on the total number of layers ℓ. In the **Aggregate Phase**, each P_i performs an addition for the collective aggregation of the gradients in which the complexity is negligible. To update the weights, **Model-Update Phase** is done by one party (P_1) and divisions do not consume levels when performed with $\mathsf{SetScale}(\cdot)$. The complexity of an activation function ($\varphi(\cdot)$) depends on the approximation degree d_a. We note that the derivative of the activation function ($\varphi'(\cdot)$) has the same complexity as $\varphi(\cdot)$ with degree d_a-1.

For the cryptographic primitives represented in Table 4.2, we rely on the CKKS variant of the MHE cryptosystem in [209], and we report the dominating terms. The distributed bootstrapping takes 1 round of communication and the size of the communication scales with the number of parties (N) and the size of the ciphertext (see [209] for details).

4.3.6 Parameter Selection

We first discuss several details to optimize the number of $\mathsf{Res}(\cdot)$ operations and give a cost function which is computed by the complexities of each functionality presented in Table 4.2. Finally, relying on this cost function we formulate an optimization problem for choosing POSEIDON' parameters.

As discussed in Section 1.3, we assume that each multiplication is followed by a $\mathsf{Res}(\cdot)$ operation. The number of total rescaling operations, however, can be further reduced by checking the scale of the ciphertext. When the initial scale S is chosen such that $Q/S = r$ for a ciphertext modulus Q, the ciphertext is rescaled after r consecutive multiplications. This reduces the level consumption and is integrated into our cost function hereinafter.

Cryptographic Parameters Optimization. We define the overall complexity of an ℓ-layer MLP aiming to formulate a constrained optimization problem for choosing the cryptographic parameters. We first introduce the total number of bootstrapping operations ($\mathscr{B}$) required in one forward and backward pass. As $\mathscr{B}$ depends on the multiplicative depth, the factors that affect it are the number of multiplications and the initial level of the ciphertext. Thus we calculate $\mathscr{B}$ as

$$\mathscr{B} = \frac{\ell(5 + \lceil \log_2(d_a + 1) + \lceil \log_2(d_a) \rceil)}{(L - \tau)r},$$

where $r = Q/S$, for a ciphertext modulus Q and an initial scale S. The number of total bootstrapping operations is calculated by the total number of consumed levels (numerator), the level requiring a bootstrap $(L - \tau)$ and r which denotes how many consecutive multiplications are allowed before rescaling (denominator). Note that a forward and backward pass for each layer consumes 5 multiplications in total by default and the remaining multiplications depend on the degree of the activation function d_a. The initial level of a fresh ciphertext L has an effect on the design of the protocols, as the ciphertext should be bootstrapped before the level $L_{\boldsymbol{c}}$ reaches a number $(L - \tau)$ that is close to zero, where τ depends on the security parameters. For a cyclotomic ring size $\mathscr{N}$, the initial level of a ciphertext L, and for the fixed neural network parameters such as the number of layers ℓ, the number of neurons in each layer $h_1, h_2, ..., h_\ell$, and for the number of global iterations e, the overall complexity is defined as

$$C(\mathscr{N}, L) = m(\sum_{i=1}^{\ell} \{(2\log_2(h_{i-1}) + \log_2(h_{i+1})) \cdot \mathsf{KS} + 3\mathsf{Mul}_{\mathsf{ct}} + 2\mathsf{Mul}_{\mathsf{pt}} + \varphi + \varphi'\} - 2\log_2(h_\ell) + \mathscr{B} \cdot \mathsf{DB}). \tag{4.1}$$

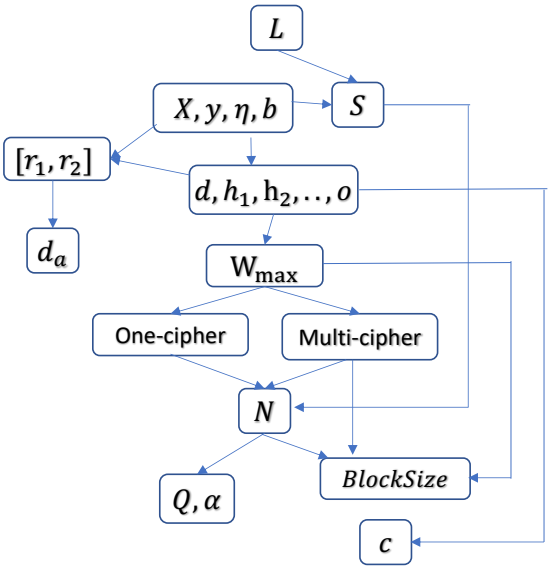


Figure 4.1: *The relationship between* POSEIDON*'s cryptographic and learning parameters.*

This calculation is thus takes the number of rotations that are needed for each vector-matrix multiplication $+\log_2(h)$, the number of multiplications ($\mathsf{Mul}_{\mathsf{pt}}$ or $\mathsf{Mul}_{\mathsf{ct}}$), the number of bootstraps ($\mathscr{B}$) and the activation functions (φ) into account. Note that the complexity of each $\mathsf{KS}(\cdot)$ operation depends on the level of the ciphertext that it is performed on (see Table 4.2), but we use the initial level L in the cost function for the sake of clarity. The complexity of $\mathsf{Mul}_{\mathsf{ct}}$,$\mathsf{Mul}_{\mathsf{pt}}$,DB, and KS is defined in Table 4.2. Then, the optimization problem for a fixed scale (precision) S and a security level λ, which defines the security parameters, can be formulated as

$$\min_{\mathscr{N},L} C(\mathscr{N},L) \tag{4.2}$$
$$\text{subject to } mc=\{q_1,...,q_L\};\ L=|mc|;\ Q=\prod_{i=1}^{L} q_i;\ Q=kS,\ k\in\mathbb{R}^+;$$
$$Q_{L-\tau}>2^{\lambda}|plaintext|N;\ \mathscr{N}\leftarrow \text{postQsec}(Q,\lambda),$$

where postQsec(Q,L,λ) gives the necessary cyclotomic ring size $\mathscr{N}$, depending on the ciphertext modulus (Q) and on the desired security level (λ), according to the homomorphic encryption standard whitepaper [27]. Eq. (4.2) gives the optimal $\mathscr{N}$ and L for a given neural network structure. We then pack each weight matrix into one ciphertext. It is worth mentioning that the solution might give an $\mathscr{N}$ that has fewer slots than the required number to pack the big weight matrices in the neural network. In this case, we use a multi-cipher approach where we pack the weight matrix using more than one ciphertext and do the operations in parallel.

One-cipher approach: We find the largest weight matrix to be packed and calculate the necessary number of slots to fit this matrix into one ciphertext, including the gaps introduced in between rows or columns. When optimization problem 4.2 finds optimal $\mathscr{N}$ such that the

number of slots fit into one ciphertext, we follow the one-cipher approach for the evaluation.
Multi-cipher Approach. In the case of a big weight matrix, we divide the flattened weight vector into multiple ciphertexts and carry out the neural network operations on several ciphertexts in parallel. E.g., for a weight matrix of size $1,024 \times 64$ and $\mathcal{N}/2 = 4,096$ slots, we divide the weight matrix into $1,024 \times 64/4,096 = 16$ ciphers. Last but not least, we show in Figure 4.1 the relationship between the aforementioned parameters.

4.4 Security Analysis

We demonstrate that POSEIDON achieves the Data and Model Confidentiality properties defined in Chapter 3, Section 3.2, under a passive-adversary model with up to $N-1$ colluding parties. We follow the real/ideal world simulation paradigm [180] for the confidentiality proofs.

The semantic security of the CKKS scheme is based on the hardness of the decisional RLWE problem [62, 193, 181]. The achieved practical bit-security against state-of-the-art attacks can be computed using Albrecht's LWE-Estimator [27, 28]. According to Lemma 1 in [62], a CKKS ciphertext generated with parameters $(\mathcal{N}, Q_L, S)$ that ensures a post-quantum security level of λ, is a valid encryption (i.e., indistinguishable from random data) due to the semantic security of RLWE-based encryptions. Moreover, following Lemmas 2, 3, and 4 in [62], any local homomorphic operation on a CKKS ciphertext, e.g., addition, multiplication, or rescaling, yields a valid encryption of the result of the operation. The security of the used distributed cryptographic protocols, i.e., $\mathsf{DKeyGen}(\cdot)$ and $\mathsf{DKeySwitch}(\cdot)$, relies on the proofs by Mouchet et al. [209]. They show that these protocols are secure in a passive-adversary model with up to $N-1$ colluding parties, under the assumption that the underlying RLWE problem is hard [209]. The security of $\mathsf{DBootstrap}(\cdot)$, and its variant $\mathsf{DBootstrapALT}(\cdot)$ is based on Lemma 1 which we state and prove in Appendix C.4.

Remark 1. Any encryption broadcast to the network in Algorithm 2 is re-randomized to avoid leakage about parties' confidential data by two consecutive broadcasts. We omit this operation in Algorithm 2 for clarity.

Proposition 1. *Assume that* POSEIDON*'s encryptions are generated using the CKKS cryptosystem with parameters $(\mathcal{N}, Q_L, S)$ ensuring a post-quantum security level of λ. Given a passive adversary corrupting at most $N-1$ parties,* POSEIDON *achieves* Data and Model Confidentiality *during training.*

Proof (Sketch). Let us assume a real-world simulator $\mathcal{S}_t$ that simulates the view of a computationally bounded adversary corrupting $N-1$ parties, as such having access to the inputs and outputs of $N-1$ parties. Without loss of generality, let us assume one training iteration in POSEIDON. As stated above, any encryption under CKKS with parameters that ensure a post-quantum security level of λ is semantically secure. During POSEIDON's training phase, the model parameters that are exchanged in between parties are encrypted, and all phases rely on the aforementioned CPA-secure-proven protocols. Moreover, as shown in Appendix C.4,

the DBootstrap(·) and DBootstrapALT(·) protocols are simulatable. Hence, $\mathscr{S}_t$ can simulate all of the values communicated during POSEIDON's training phase by using the parameters $(\mathcal{N}, Q_L, S)$ to generate random ciphertexts such that the real outputs cannot be distinguished from the ideal ones. The sequential composition of all cryptographic functions remains simulatable by $\mathscr{S}_t$ thanks to the use of different random values in each phase and due to Remark 1. As such, there is no dependency between the random values that an adversary can leverage on. Moreover, the adversary is not able to decrypt the communicated values of an honest party because decryption is only possible with the collaboration of *all* the parties. Following this, POSEIDON protects the data confidentiality of the honest party/ies.

Analogously, the same argument follows to prove that POSEIDON protects the confidentiality of the trained model, as it is a function of the parties' inputs, and its intermediate and final weights are always under encryption. Hence, POSEIDON eliminates federated learning attacks [128, 199, 212, 309], that aim at extracting private information about the parties from the intermediate parameters or the final model.

Proposition 2. *Assume that* POSEIDON*'s encryptions are generated using the CKKS cryptosystem with parameters $(\mathcal{N}, Q_L, S)$ ensuring a post-quantum security level of λ. Given a passive adversary corrupting at most $N-1$ parties,* POSEIDON *achieves* Data *and* Model Confidentiality *during prediction.*

Proof (Sketch). (a) Let us assume a real-world simulator $\mathscr{S}_p$ that simulates the view of a computationally-bounded adversary corrupting $N-1$ computing nodes (parties). The *Data Confidentiality* of the honest parties and *Model Confidentiality* is ensured following the arguments of Proposition 1, as the prediction protocol is equivalent to a forward-pass performed during a training iteration by a computing party. Following similar arguments to Proposition 1, the encryption of the querier's input data (with the parties common public key pk) can be simulated by $\mathscr{S}_p$. The only additional function used in the prediction step is DKeySwitch(·) that is proven to be simulatable by $\mathscr{S}_p$ [209]. Thus, POSEIDON ensures *Data Confidentiality* of the querier. (b) Let us assume a real-world simulator $\mathscr{S}'_p$ that simulates a computationally-bounded adversary corrupting $N-2$ parties and the querier. *Data Confidentiality* of the querier is trivial, as it is controlled by the adversary which is the querier that has its own public and secret key pairs. The simulator has access to the prediction result as the output of the process for P_q, so it can produce all the intermediate (indistinguishable) encryptions that the adversary sees (based on the simulatability of the key-switch/collective decrypt protocol [209]). Following this and the arguments of Proposition 1, *Data and Model Confidentiality* are ensured during prediction. We remind here that the membership inference [254] and model inversion [98] are out-of-the-scope attacks (see B.1 for complementary security mechanisms against these attacks).

4.5 Experimental Evaluation

In this section, we experimentally evaluate POSEIDON's performance and present our empirical results. We also compare POSEIDON to other state-of-the-art privacy-preserving solutions.

4.5.1 Implementation Details

We implement POSEIDON in Go [7], building on top of the Lattigo lattice-based library [200] for the multiparty cryptographic operations. We make use of Onet [5] and build a decentralized system where the parties communicate over TCP with secure channels (TLS).

4.5.2 Experimental Setup

We use Mininet [201] to evaluate POSEIDON in a virtual network with an average network delay of 0.17ms and 1Gbps bandwidth. All the 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. Unless otherwise stated, in our *default* experimental setting, we instantiate POSEIDON with $N = 10$ and $N = 50$ parties. As for the parameters of the cryptographic scheme, we use a precision of 32 bits, number of levels $L = 6$, and $\mathcal{N} = 2^{13}$ for the datasets with $d < 32$ or 32×32 images, and $\mathcal{N} = 2^{14}$ for those with $d > 32$, following the multi-cipher approach (see Section 4.3.6).

4.5.3 Datasets

For the evaluation of POSEIDON's performance, we use the following real-world and publicly available datasets: (a) the Breast Cancer Wisconsin dataset (BCW) [236] with $n = 699$, $d = 9$, $h_\ell = 2$, (b) the hand-written digits (MNIST) dataset [172] with $n = 70,000$, $d = 28 \times 28$, $h_\ell = 10$, (c) the Epileptic seizure recognition (ESR) dataset [91] with $n = 11,500$, $d = 179$, $h_\ell = 2$, (d) the default of credit card clients (CREDIT) dataset [291] with $n = 30,000$, $d = 23$, $h_\ell = 2$, (d) the street view house numbers (SVHN) dataset [215] with colored images (3 channels), $n = 600,000$, $d = 3 \times 32 \times 32$, $h_\ell = 10$, and (e) the CIFAR-10 and CIFAR-100 [168] datasets with colored images (3 channels), $n = 60,000$, $d = 3 \times 32 \times 32$, $h_\ell = 10$, and $h_\ell = 100$, respectively. Recall that h_ℓ represents the number of neurons in the last layer of a neural network (NN), i.e., the number of output labels. We convert SVHN to gray-scale to reduce the number of channels. Moreover, since we pad with zeros each dimension of a weight matrix to the nearest power-of-two (see Section 4.3.1), for the experiments using the CREDIT, ESR, and MNIST datasets, we actually perform the neural network training with $d = 32$, 256, and $1,024$ features, respectively. For SVHN, the number of features for a flattened gray-scale image is already a power-of-two ($32 \times 32 = 1,024$). To evaluate the scalability of our system, we generate synthetic datasets and vary the number of features or samples. Finally, for our experiments we evenly and randomly distribute all the above datasets among the participating parties. We note that the data and label distribution between the parties, and its effects on the model accuracy is

Dataset	Accuracy					Execution time (s)	
	C1	C2	L	D	POSEIDON	Training	Inference
BCW	97.8%	97.4%	93.9%	97.4%	96.9%	91.06	0.21
ESR	93.6%	91.2%	89.9%	91.1%	90.4%	851.84	0.30
CREDIT	81.4%	80.9%	79.6%	80.6%	80.2%	516.61	0.26
MNIST	92.1%	91.3%	87.8%	90.6%	89.9%	5,283.1	0.38

Table 4.3: POSEIDON*'s accuracy and execution times for* $N = 10$ *parties. The model accuracy is compared to several non-private approaches.*

orthogonal to this work (see Appendix B.2 for extensions related to this issue).

4.5.4 Neural Network Configuration

For the BCW, ESR, and CREDIT datasets, we deploy a 2-layer fully connected neural network with 64 neurons per layer, and we use the same neural network structure for the synthetic datasets used to test POSEIDON's scalability. For the MNIST and SVHN datasets, we train a 3-layer fully connected neural network with 64 neurons per-layer. For the CIFAR-10, we train two models: (i) a CNN with 2 CV and 2 average-pooling with kernel size of 2×2, and 2 FC layers with 128 neurons and 10 neurons, labeled as N1, and (ii) a CNN with 4 CV with kernel size of 3×4, 2 average-pooling with kernel size of 2×2 and 2 FC layers with 128 and 10 neurons labeled as N2. For CIFAR-100, we train a CNN with 6 CV with kernel size of 3×4, 2 average-pooling with kernel size of 2×2 and 2 FC layers with 128 neurons each. For all CV layers, we vary the number of filters between 3 to 16. We use the approximated sigmoid, SmoothReLU, or tanh activation functions (see Section 4.3.2), depending on the dataset. We train the above models for 100, 600, 500, 1,000, 18,000, 25,000, 16,800, and 54,000 global iterations for the BCW, ESR, CREDIT, MNIST, SVHN, CIFAR-10-N1, CIFAR-10-N2, and CIFAR100 datasets, respectively. For the SVHN and CIFAR datasets, we use momentum-based gradient descent [228] or Nesterov's accelerated gradient descent [214], which introduces an additional multiplication to the update rule (in the **Local Computation Phase**). Finally, we set the local batch size b to 10 and, as such, the global batch size is $B = 100$ in our default setting with 10 parties and $B = 500$ with 50 parties. For a fixed number of layers, we choose the learning parameters by grid search with 3-fold cross validation on clear data with the *approximated* activation functions. In a practical federated learning setting, however, the parties can collectively agree on these parameters by using secure statistics computations [104, 100].

4.5.5 Empirical Results

We experimentally evaluate POSEIDON in terms of accuracy of the trained model, execution time for both training and prediction phases, and communication overhead. We also evaluate POSEIDON's scalability with respect to the number of parties N, as well as the number of data samples n and features d in a dataset. We further provide microbenchmark timings

and communication overhead for the various functionalities and operations for FC, CV, and pooling layers in Appendix C.5.1 that can be used to extrapolate POSEIDON' execution time for different neural network structures. We further give per-global-iteration execution times of various neural network architectures in Appendix C.5.2 and various CNN architectures in Appendix C.5.3.

Model Accuracy. Tables 4.3 and 4.4 display POSEIDON's accuracy results on the used real-world datasets with 10 and 50 parties, respectively. The accuracy column shows four baselines with the following approaches: two approaches where the data is collected to a central party in its clear form: centralized with original activation functions (C1), and centralized with approximated activation functions (C2); one approach where each party trains the model only with its local data (L), and a decentralized approach with approximated activation functions (D), where the data is distributed among the parties, but the learning is performed on cleartext data, i.e., without any protection of the gradients communicated between the parties. For all baselines, we use the same neural network structure and learning parameters as POSEIDON, but adjust the learning rate (η) or use adaptive learning rate to ensure the range of the approximated activation functions is minimized, i.e., a smaller interval for an activation-function approximation requires smaller η to prevent divergence while bigger intervals make the choice of η more flexible. These baselines enable us to evaluate POSEIDON's accuracy loss due to the approximation of the activation functions, distribution, encryption and the impact of privacy-preserving federated learning. We exclude the (D) column from Table 4.4 for the sake of space; the pattern is similar to Table 4.3 and POSEIDON's accuracy loss is negligible. To obtain accuracy results for the CIFAR-10 and CIFAR-100 datasets, we simulate POSEIDON in Tensorflow [18] by using its approximated activation functions and a fixed-precision. We observe that the accuracy loss between C1, C2, D, and POSEIDON is $0.9-3\%$ when 32-bits precision is used. For instance, POSEIDON achieves 90.4% training accuracy on the ESR dataset, a performance that is equivalent to a decentralized (D) non-private approach and only slightly lower compared to centralized approaches. Note that the accuracy difference between non-secure solutions and POSEIDON can be further reduced by increasing the number of training iterations, however, we use the same number of iterations for the sake of comparison. Moreover, we remind that CIFAR-100 has 100 class labels (i.e., a random guess baseline of 1% accuracy) and is usually trained with special neural network structures (ResNet) or special layers (batch normalization) to achieve higher accuracy than the reported ones: we leave these neural network types as future work (see Appendix B.2).

We compare POSEIDON's accuracy with that achieved by one party using its local dataset (L), that is 1/10 (or 1/50) of the overall data, with *exact* activation functions. We compute the accuracy for the (L) setting by averaging the test accuracy of the 10 and 50 locally trained models (Tables 4.3 and 4.4, respectively). We observe that even with the accuracy loss due to approximation and encryption, POSEIDON still achieves $1-3\%$ increase in the model accuracy due to privacy-preserving collaboration (Table 4.3). This increase is more significant when the data is partitioned across 50 parties (Table 4.4) as the number of training samples *per-party* is further reduced and is not sufficient to learn an accurate model.

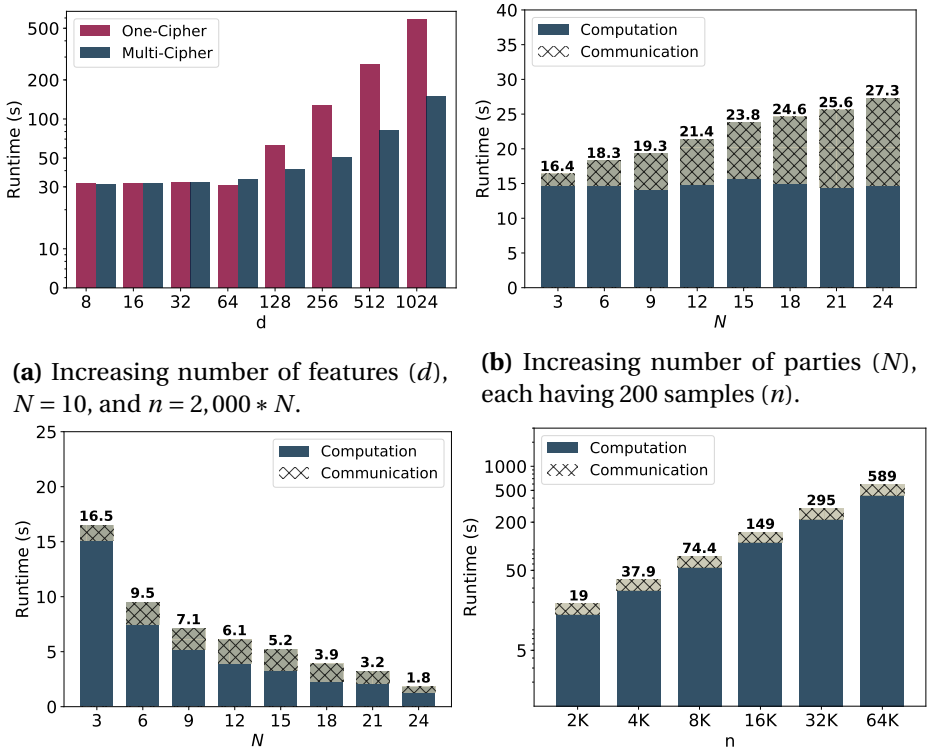
Dataset	Accuracy				Execution time (hrs)		
	C1	C2	L	POSEIDON	One-GI	Training	Inference
SVHN	68.4%	68.1%	35.1%	67.8%	0.0034	61.2	8.89×10^{-5}
CIFAR-10-N1	54.6%	52.1%	26.8%	51.8%	0.007	175	0.001
CIFAR-10-N2	63.6%	62.0%	28.0%	61.1%	0.011	184.8	0.004
CIFAR-100	43.6%	41.8%	8.2%	41.1%	0.026	1404	0.006

Table 4.4: POSEIDON*'s accuracy and execution times for* $N = 50$ *parties (extrapolated). One-GI indicates the execution time of one global iteration.*

Execution Time. As shown on the right-hand side of Table 4.3, POSEIDON trains the BCW, ESR, and CREDIT datasets in less than 15 minutes and the MNIST in 1.4 hours, when each dataset is evenly distributed among 10 parties. Note that POSEIDON's overall training time for MNIST is less than an hour when the dataset is split among 20 parties that use the same local batch size. We extrapolate the training times of POSEIDON on more complex datasets and architectures for one global iteration (one-GI) in Table 4.4; these can be used to estimate the training times of these structures with a larger number of global iterations. For instance, CIFAR-10 is trained in 175 hours with 2 CV, 2 pooling, 2 FC layers and with dropouts (adding one more multiplication in the dropout layer). Note that it is possible to increase the accuracy with higher run-time or fine-tuned architectures, but we aim at finding a trade-off between accuracy and run-time. For example, POSEIDON's accuracy on SVHN reaches 75% by doubling the training epochs and thus its execution time. The per-sample inference times presented in Tables 4.3 and 4.4 include the forward pass, the $\mathsf{DKeySwitch}(\cdot)$ operations that reencrypt the result with the querier's public key, and the communication among the parties. We note that as all the parties keep the model in encrypted form, any of them can process the prediction query. Hence, taking the advantage of parallel query executions and multi-threading, POSEIDON achieves a throughput of 864,000 predictions per hour on the MNIST dataset with the chosen neural network structure.

Scalability. Figure 4.2a shows the scaling of POSEIDON with the number of features (d) when the one-cipher and multi-cipher with parallelization approaches are used for a 2-layer neural network with 64 hidden neurons. The runtime refers to one epoch, i.e., a processing of all the data from $N = 10$ parties, each having 2,000 samples, and employing a batch size of $b = 10$. For small datasets with a number of features between 1 and 64, we observe no difference in execution time between the one-cipher and multi-cipher approaches. This is because the weight matrices between layers fit in one ciphertext with $\mathcal{N} = 2^{13}$. However, we observe a larger runtime of the one-cipher approach when the number of features increases further. This is because each power-of-two increase in the number of features requires an increase in the cryptographic parameters, thus introducing overhead in the arithmetic operations.

We further analyse POSEIDON's scalability with respect to the number of parties (N) and the number of total samples in the distributed dataset (n), for a fixed number of features. Figures 4.2b and 4.2c display POSEIDON's execution time, when the number of parties ranges



(a) Increasing number of features (d), $N = 10$, and $n = 2,000 * N$.

(b) Increasing number of parties (N), each having 200 samples (n).

(c) Increasing number of parties (N) when n (number of data samples) is fixed to 600.

(d) Increasing number of data samples (n) when N (number of parties) is fixed to 10.

Figure 4.2: POSEIDON*'s training execution time and communication overhead with increasing number of parties, features, and samples, for 1 training epoch.*

from 3 to 24, and one training epoch is performed, i.e., all the data of the parties is processed once. For Figure 4.2b, we fix the number of data samples per party to 200 to study the effect of an increasing number of members in the federation. We observe that POSEIDON's execution time is almost independent of N and is affected only by increasing communication between the parties. When we fix the global number of samples (n), increasing N results in a runtime decrease, as the samples are processed by the parties in parallel (see Figure 4.2c). Then, we evaluate POSEIDON's runtime with an increasing number of data samples and a fixed number of parties $N = 10$, in Figure 4.2d. We observe that POSEIDON scales linearly with the number of data samples. Finally, we remark that POSEIDON also scales proportionally with the number of layers in the neural network structure, if these are all of the same type, i.e, FC, CV, or pooling, and if the number of neurons per layer or the kernel size is fixed.

4.5.6 Comparison with Prior Work

A quantitative comparison of our work with the state-of-the-art solutions for privacy-preserving neural network executions is a non-trivial task. Indeed, the most recent cryptographic solutions for privacy-preserving machine learning in the N-party setting, i.e., Helen [307] and SPINDLE [102], support the functionalities of only regularized [307] and generalized [102]

linear models, respectively. We provide a detailed qualitative comparison with the state-of-the-art privacy-preserving deep learning frameworks in Table C.1 in Appendix and expand on it here.

POSEIDON operates in an federated learning setting where the parties maintain their data locally. This is a substantially different setting compared to that envisioned by MPC-based solutions [205, 204, 278, 279, 53, 57], for privacy-preserving neural network training. In these solutions, the parties' data has to be communicated (i.e., secret-shared) outside their premises, and the data and model confidentiality is preserved as long as there exists an honest majority among a limited number of computing servers (typically, 2 to 4, depending on the setting). Hence, a similar experimental setting is hard to achieve. Nonetheless, we compare POSEIDON to SecureML [205], SecureNN [278], and FALCON [279], when training a 3-layer neural network with 128 neurons per layer for 15 epochs, as described in [205], on the MNIST dataset. We set $N = 3$ to simulate a similar setting and use POSEIDON's approximated activation functions. POSEIDON trains MNIST in 73.1 hours whereas SecureML with 2-parties, SecureNN and FALCON with 3-parties, need 81.7, 1.03, and 0.56 hours, respectively. Depending on the activation functions, SecureML yields $93.1 - 93.4\%$ accuracy, SecureNN 93.4%, and FALCON 97.4%. POSEIDON achieves 92.5% accuracy with the approximated SmoothReLU and 96.2% with approximated tanh activation functions. We remind that POSEIDON operates under a different system (FL-based) and threat model, it supports more parties, and scales linearly with N whereas MPC solutions are based on outsourced learning with limited number of computing servers.

Federated learning approaches based on differential privacy (DP), e.g., [177, 253, 195], train a neural network while introducing some noise to the intermediate values to mitigate adversarial inferences. However, training an accurate neural network model with DP requires a high privacy budget [229], hence it remains unclear what privacy protection is obtained in practice [141]. We note that DP-based approaches introduce a different tradeoff than POSEIDON: they tradeoff privacy for accuracy, while POSEIDON decouples accuracy from privacy and tradeoffs accuracy for complexity (i.e., execution time and communication overhead). Nonetheless and as an example, we compare POSEIDON's accuracy results with those reported by Shokri and Shmatikov [253] on the MNIST dataset. We focus on their results with the distributed selective SGD configured such that participants download/upload *all* the parameters from/to the central server in each training iteration. We evaluate the same CNN structure used in [253], but with POSEIDON's approximated activation functions and average-pooling instead of max-pooling. We compare the accuracy results presented in [253, Figure 13] with $N = 30, N = 90$, and $N = 150$ participants. In all settings, POSEIDON yields $> 94\%$ accuracy whereas [253] achieves similar accuracy only when the privacy budget per parameter is ≥ 10. For more private solutions, where the privacy budget is 0.001, 0.01 or 0.1, [253] achieves $\leq 90\%$ accuracy; smaller ϵ yields better privacy but degrades utility.

Finally, existing HE-based solutions [126, 211, 276], focus on a centralized setting where the neural network learning task is outsourced to a central server. These solutions, however,

employ non-realistic cryptographic parameters [276, 211], and their performance is not practical [126] due to their costly homomorphic computations. Our system, focused on a federated learning-based setting and a multiparty homomorphic encryption scheme, improves the response time 3 to 4 orders of magnitude. The execution times produced by Nandakumar et al. [211] for processing one batch of 60 samples in a single thread and 30 threads for a neural network structure with $d = 64$, $h_1 = 32$, $h_2 = 16$, $h_3 = 2$, are respectively 33,840s and 2,400s. When we evaluate the same setting, but with $N = 10$ parties, we observe that POSEIDON processes the same batch in 6.3s and 1s, respectively. We also achieve stronger security guarantees (128 bits) than [211] (80 bits). Finally, for a neural network structure with 2-hidden layers of 128 neurons each, and the MNIST dataset, CryptoDL [126] processes a batch with $B = 192$ in 10,476.3s, whereas our system in the distributed setting processes the same batch in 34.7s.

Therefore, POSEIDON is the only solution that performs both training and inference of MLP and CNNs in an N-party setting, yet protects data and model confidentiality withstanding collusions up to $N-1$ parties.

4.6 Conclusion

In this chapter, we presented POSEIDON, a novel system for zero-leakage privacy-preserving federated MLP and CNN learning among N parties. Based on lattice-based multiparty homomorphic encryption, our system protects the confidentiality of the training data, of the model, and of the evaluation data, under a passive adversary model with collusions of up to $N-1$ parties. By leveraging on packing strategies and an extended distributed bootstrapping functionality, POSEIDON is the first system demonstrating that secure federated learning on neural networks is practical under multiparty homomorphic encryption. Our experimental evaluation shows that POSEIDON significantly improves on the accuracy of individual local training, bringing it on par with centralized and decentralized non-private approaches. Its computation and communication overhead scales linearly with the number of parties that participate in the training, and is between 3 to 4 orders of magnitude faster than equivalent centralized outsourced approaches based on traditional homomorphic encryption. This work opens up the door of practical and secure federated training in passive-adversarial settings.

5 Federated Recurrent Neural Network Learning

5.1 Introduction

In this chapter, we address the privacy-preserving training and evaluation on RNNs in an N-party setting by relying on MHE to perform all the computations under encryption. Collecting and mining sequential data, e.g., time-series, has become the base for numerous real-life 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 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) [239, 90] 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 [246].

Training effective RNNs for time-series tasks requires large amounts of data that are usually generated by multiple sources and scattered across several parties. Sharing or centralizing this data is difficult because it is privacy sensitive [298]. For example, smart-meter data might leak householders' identities [51] and their activities [206], location data reveal information about peoples' lifestyles and beliefs [256, 255, 217, 296, 17, 73], and healthcare time-series are by nature very private information [36, 186]. Additionally, privacy regulations enforce strict rules for sharing this type of data [3, 8].

Federated learning has been applied in many time-series applications by using RNNs to forecast the energy load [93], to detect out-of-vocabulary words [59], to predict the next word typed on mobile phones [121], and to classify cancer in healthcare systems [107]. We remind here that federated learning raises privacy issues through the shared model updates [128, 283, 309, 199, 212, 304, 110, 145, 280, 85, 87]. To overcome these shortcomings, several works propose integrating protection mechanisms such as differential privacy (DP), secure multiparty computation (MPC), or homomorphic encryption (HE) in the federated learning process (see Chapter 2 for a detailed discussion on the related work). Our contribu-

tions in Chapter 4 focus solely on the training of 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 (or vanishing) gradients [221, 37]. For instance, the packing scheme and the matrix/vector operations employed in [244] are tailored for the processing of a single sample (by relying on vector-matrix multiplications and thus eliminating explicit transpose and matrix multiplications; see Section 5.5.6 for microbenchmarks demonstrating the unsuitability of this approach for training RNNs). Whereas in this chapter, we propose a novel packing scheme suitable for mini-batch training of RNNs and enable matrix transpose and matrix multiplications. Moreover, we propose several polynomial approximations for enabling critical operations for RNNs, such as gradient clipping, under MHE.

To accelerate RNN training, we enable mini-batch training by relying on a novel packing scheme that reduces the processing time of a batch. To address the problem of exploding/vanishing gradients, our novel system, RHODE, uses efficient polynomial approximations of the gradient clipping function. We implement RHODE and our experimental evaluation shows that its accuracy is on par with non-secure 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 parties.

In summary, we make the following contributions in this chapter:

- We present RHODE, a novel system for the training of RNNs in a cross-silo federated learning setting by relying on MHE. By using MHE, RHODE conceals all intermediate values and the model that are communicated in the federated learning process. After training, RHODE enables oblivious predictions to a querier that provides its encrypted inputs for PaaS.
- We design a novel multi-dimensional packing scheme that enables efficient encrypted matrix operations suitable for mini-batch training. We show that performing matrix multiplication with our scheme improves the state of the art [143, 244] in terms of throughput proportionally to the dimension of the matrices. Moreover, our packing scheme preserves the number and size of the cryptographic keys regardless of the matrix size.
- We propose and evaluate various polynomial approximations for performing gradient clipping under encryption to alleviate the problem of vanishing/exploding gradient 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 parties depending on the number of samples in the dataset.

– 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 distribution among the parties.

To the best of our knowledge, RHODE is the first system that enables the training of and prediction on sequential data with RNNs in a cross-silo federated learning setting under encryption. Our system mitigates passive federated learning inference attacks during training and preserves the privacy of the parties' data, the intermediate model/gradients communicated through network, the querier's evaluation data, and the final model.

Acknowledgements. This work is a result of the collaboration with Abdulrahman Diaa, Dr. Apostolos Pyrgelis, Jean-Philippe Bossuat, and Prof. Jean-Pierre Hubaux.

5.2 RHODE Design

The system and threat model of RHODE is described in Chapter 3, Section 3.1. We present the high-level federated learning algorithm in Chapter 1, Algorithm 1 for the collective training and our encrypted version of it in Chapter 3, Section 3.3 in Algorithm 2. As introduced in Chapter 3, RHODE's training pipeline is composed of four phases: **Setup Phase**, **Local Computation Phase**, **Aggregation Phase**, and **Model-Update Phase**; and PaaS is enabled for the predictions to the querier. RHODE enables the training and prediction execution of RNNs under encryption by keeping the model in encrypted form throughout the execution pipeline of this algorithm. Hence, the **local gradient computation phase** requires several solutions to the challenges introduced by RNNs. Below, we define the system-specific **Local Gradient Descent Computation** that refers to Line 9 of the Algorithm 2 and also present the challenges associated with RNN training.

Local Gradient Descent Computation: The execution of the forward and backward pass on the parties' local data are executed under encryption and we introduce the algorithm for a simple many-to-many RNN with $\kappa = T$ outputs in Algorithm 6 for a party P_i with its own input data and the labels X_i and y_i. We note here that configuring RHODE to support different RNN variants in terms of architecture, such as Gated Recurrent Unit (GRU) or Long Short-Term Memory (LSTM), requires modifications to the hidden units, e.g., to account for memory units in LSTMs. Throughout this chapter, we consider the traditional RNN architecture that is called simple, vanilla, or Elman RNN [90]. A detailed discussion about how to extend RHODE to more coplex RNN architectures and an example of the **Local Computation Phase** for a Gated Recurrent Unit (GRU) architecture are given in Appendix B.

We remind that the encryption of any value, vector, or matrix are denoted with bold-face, e.g., $\mathbf{W}$, and for clarity, we omit the indices of the global iteration and the party in Algorithm 1. $\bar{x_t}$ is the input d-dimensional feature-vector for a specific timestep t (or a matrix of size

Algorithm 6 Local Computation Phase (RNN)

Input: $\mathbf{U}, \mathbf{W}, \mathbf{V}, \mathbf{h}_{prev}, X_i = (\bar{x_1}, \bar{x_2}, \cdots, \bar{x_t}), Y_i = (\bar{y_1}, \bar{y_2}, \cdots, y_t)$
Output:$\nabla\mathbf{U}, \nabla\mathbf{W}, \nabla\mathbf{V}$

$\mathbf{h}_0 \leftarrow \mathbf{h}_{prev}$
for $t \leftarrow 1 : T$ **do** ▷ Forward Pass
  $\mathbf{z}_t \leftarrow \mathbf{h}_{t-1} \times \mathbf{W} + \bar{x}_t \times \mathbf{U}$
  $\mathbf{h}_t \leftarrow \varphi(\mathbf{z}_t)$
  $\mathbf{p}_t \leftarrow \mathbf{h}_t \times \mathbf{V}$
end for
$\mathbf{h}_{prev} \leftarrow \mathbf{h}_t$
$\mathbf{dh}_{next} = \mathbf{0}$
$\nabla\mathbf{U}, \nabla\mathbf{W}, \nabla\mathbf{V} = \mathbf{0}$
for $t \leftarrow T : 1$ **do** ▷ Backward Pass
  $\mathbf{dy} \leftarrow \mathbf{p}_t - \bar{y}_t$
  $\mathbf{dh} \leftarrow \mathbf{dy} \times \mathbf{V}^T + \mathbf{dh}_{next}$
  $\mathbf{dz} \leftarrow \mathbf{dh} \odot \varphi'(\mathbf{z}_t)$
  $\mathbf{dh}_{next} \leftarrow \mathbf{dz} \times \mathbf{W}^T$
  $\nabla\mathbf{V} \leftarrow \nabla\mathbf{V} + \mathbf{h}_t \otimes \mathbf{dy}$
  $\nabla\mathbf{U} \leftarrow \nabla\mathbf{U} + \bar{x}_t \otimes \mathbf{dz}$
  $\nabla\mathbf{W} \leftarrow \nabla\mathbf{W} + \mathbf{h}_{t-1} \otimes \mathbf{dz}$
end for

$b \times d$ 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). Recall that T denotes the number of timesteps and $\mathbf{U}, \mathbf{W}, \mathbf{V}$ denotes the encrypted input-to-hidden, hidden-to-hidden, and hidden-to-output weight matrices, respectively.

φ (φ') indicates any type of activation function (and its derivative), e.g., Tanh or Sigmoid.

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 and choosing appropriate solutions is crucial. We discuss these challenges in Section 5.2.1 and propose several techniques and optimizations to alleviate them in Section 5.3.

5.2.1 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 homomorphic encryption. Section 5.3 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 or, in other words, 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; during the execution of the back-propagation through time (BBPT) algorithm these are computed based on numerous consecutive matrix multiplications according to the chain rule [221, 37].

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 [158, 252, 135]. 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 [71]) 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 5.3.1, we propose a novel multi-dimensional packing scheme that reduces the processing time of a large batch.

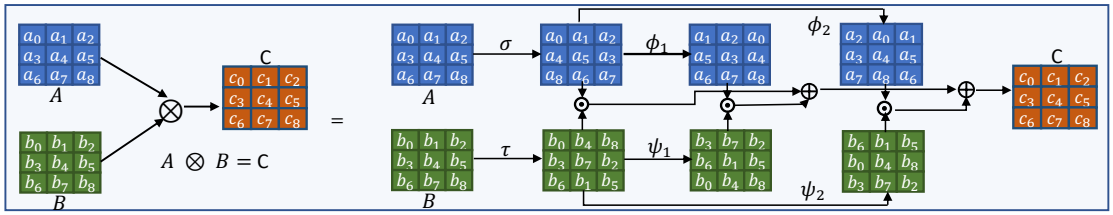
The second challenge of exploding or vanishing gradients is addressed in the literature by several techniques such as the use of gating mechanisms, e.g., LSTMs [129], along with a possible modification of the gradient propagation [152], encoder-decoder approaches [68], gradient clipping [221], non-saturating functions [55], and various recurrent-weight initialization techniques via identity or orthogonal matrix initialization [171, 125]. Extending RNNs to 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 describe a similar extension in Appendix B. Although various weight-initialization techniques are straightforward to adopt, the most widely applied solution, i.e., gradient clipping, is not easy to compute under encryption as it requires a comparison function that is not homomorphically enabled. Therefore, in Section 5.3.2, we present various approximations for the efficient computation of an element-wise clipping function, through different polynomials.

5.3 Cryptographic Building Blocks

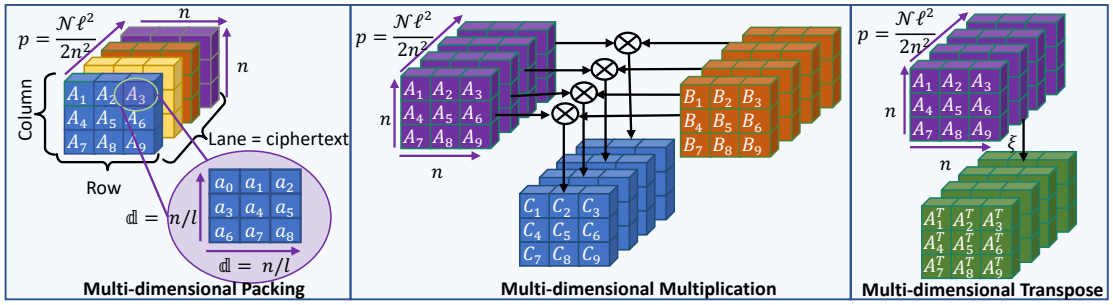
We detail the main cryptographic building blocks that enable the training of and prediction on RNNs in RHODE. We describe the packing scheme and the optimized matrix operations including the matrix-multiplication and matrix-transpose in Section 5.3.1 and the approximated neural network building blocks in Section 5.3.2. Then, we analyze RHODE's security in Section 5.4.1.

5.3.1 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 fully homomorphic encryption, 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



(a) Matrix Multiplication Method of Jiang et al. [143] for $n = 3$.



(b) An overview of our Multi-dimensional Packing scheme, its Multiplication and Transpose operations.

Figure 5.1: *Schematic Overview of the (a) Matrix Multiplication Protocol by Jiang et al. [143] and (b) our Multi-dimensional Packing Scheme.*

the most heavy and frequent operations in RNNs (see Lines 2-5 and 12-17 in Algorithm 6). Thus, 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 approach where matrices are decomposed row-wise and packed in one plaintext and we use Jiang et al.'s [143] 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 fully homomorphic encryption [120, 203] 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, or one of the matrices is encoded in diagonal form. 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 [143]. 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. However, it enables the packing of multiple matrices in a single ciphertext (assuming the number of entries of the matrix 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 favor-

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

able 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 5.1a. 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 $\mathcal{O}(n)$, $\mathcal{O}(n)$, $\mathcal{O}(1)$, $\mathcal{O}(1)$ homomorphic rotations, respectively. Hence, the matrix multiplication method of Jiang et al. has a complexity of $\mathcal{O}(n)$ rotations for two $n \times n$ matrices. This is smaller than previous approaches, such as the naive approach with $\mathcal{O}(n^3)$ or the one proposed by Halevi and Shoup [120] with $\mathcal{O}(n^2)$.

Matrix Transpose. Another operation required for RNN training is matrix transpose, as described in Algorithm 6 (Lines 12 and 14). The transpose of a row-packed matrix is straightforward and can be realized with a single linear transform of $2n$ non-zero diagonals [143] (ξ), hence, it can be evaluated with $2n$ rotations. This cost can be further reduced to $3\sqrt{n}$ by using the baby-step giant-step (BSGS) algorithm proposed by Halevi and Shoup [120] and later improved by Bossuat et al. [46]. Thus, the cost of a matrix transpose operation is $\mathcal{O}(\sqrt{n})$ for a $n \times n$ matrix.

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 5.1b. Given an $n \times n$ matrix and ℓ a divisor of n, we split this matrix into ℓ^2 smaller matrices of size $\mathrm{d} \times \mathrm{d}$, with $\mathrm{d} = \frac{n}{\ell}$. Each of these $\mathrm{d} \times \mathrm{d}$-size 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 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 $\frac{\mathcal{N}}{2} \cdot \frac{\ell^2}{n^2}$ matrices (instead of $\frac{\mathcal{N}}{2n^2}$) in parallel with $\mathcal{O}(n\ell)$ complexity for the multiplication, but amortizes to $\mathcal{O}(n/\ell)$ per matrix (instead of $\mathcal{O}(n)$), (iii) it reduces the complexity of the transpose operation from $\mathcal{O}(\sqrt{n})$ to $\mathcal{O}(\sqrt{n/\ell})$ (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 [46] to evaluate linear transforms and delay ciphertext relinearization, further reducing the complexity. We describe the protocol for multi-dimensional matrix multiplication in Algorithm 7. We also present guidelines for configuring d and other cryptographic parameters in Section 5.4.

RHODE employs the multi-dimensional packing for efficient mini-batch RNN training under encryption. In particular, we distribute the RNN input batch along the “third" dimension (see Figure 5.1b), 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 they are aligned with their parallel slices from the input batch. The computation of the forward and backward passes for all the timesteps per iteration 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 [143] approach. However, for bigger matrix dimensions, where the matrix is split over more than one ciphertext, multi-dimensional packing yields better amortized matrix multiplication cost (see Section 5.5.6).

5.3.2 Approximated Neural Network 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 [126].

Activation Functions. A neural network is a pipeline of layers composed of neurons on which linear and non-linear transformations (activations) are applied to activate them. Typical activation functions include 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 and exponentiation). To overcome this issue, previous works either change the activation functions to other linear or square functions [35, 112], or rely on various approximation techniques employing the least-squares method [102, 244], polynomial splines [184], Legendre [126] or Chebyshev polynomials [200, 126]. 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., $\varphi(x) = ln(1+e^x)$ [244]). Thus, similar to prior work, we rely on polynomial approximations employing the least-squares or Minimax methods to enable the execution of these activation functions as we empirically observed that changing them to linear functions

decreases the accuracy of RNNs.

Gradient Clipping. As described in Section 5.2.1, when the weights have a large norm, the gradients accumulated over multiple steps of the BBPT algorithm can grow exponentially; gradient clipping is a well-established technique to mitigate this issue. Thus, in RHODE, the *aggregation server* applies gradient clipping during the Model Update Phase (see Section 5.2). Pascanu et al. [221] 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 $|m|$. To avoid the calculation of the norm under encryption, we apply the following function element-wise to the accumulated gradient vector x:

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

$\mathsf{Clip}(x,m)$ 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 gradients. The first one is based on Tanh, which is very similar to Clip except for the clipping interval limits $-m$ and m:

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

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

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

Overall, RHODE enables the approximated execution of both TanhClip and SoftClip with these polynomial approximations. We evaluate the performance of both clipping approximations and provide the plots of the approximated curves (by using Minimax) and their errors for polynomial degrees of $\mathfrak{p} = 5$ and $\mathfrak{p} = 15$ in Appendix D.3. We observe that both functions achieve a similar average error over the approximation interval, yet SoftClip is slightly better than TanhClip for higher degrees (Figures D.1c and D.1d) in terms of error and behavior around the limits of the approximation interval.

Algorithm 7 Multi-dimensional Matrix Multiplication

Inputs: $\mathbf{A}$, $\mathbf{B}$ 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 (see Section 5.3.1 and Figure 5.1a).
Outputs: $\mathbf{C} \leftarrow \mathbf{A} \otimes \mathbf{B}$

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

5.4 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 how to select the cryptographic parameters and the sub-matrix dimension $\mathbb{d} = n/\ell$ (see Section 5.3.1) that is crucial for the efficiency of the multi-dimensional matrix multiplication (Algorithm 7). 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 levels $\mathscr{L}$ and Q, for reducing the number of $\mathsf{DBootstrap}(\cdot)$ 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 $\mathscr{N}$ (given the ciphertext modulus Q) [26]. 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 their choice, e.g., the number of layers, the polynomial approximation degree, the number of data holders, etc. We refer the reader to [244] for additional details on how to configure the cryptographic parameters based on other neural network learning parameters and to [209] for an overview of the cryptographic parameters.

Choosing $\mathbb{d}$. Recall that the complexity of the matrix multiplication described in Algorithm 7 amortizes to $\mathscr{O}(\mathbb{d})$. In the extreme case where $\mathbb{d} = 1$, there are no costly ciphertext operations,

except for arithmetic multiplications and additions (no linear transformations are needed). Thus, choosing $\mathbb{d} = 1$ 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 $\mathbb{d} = 1$. 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 $\mathbb{d}$ that provides an acceptable trade-off between computational cost and memory complexity. To maximize the packing density, the batch size should be $b = \frac{\mathcal{N}}{2*\mathbb{d}}$, thus we can derive that $\mathbb{d} = \frac{\mathcal{N}}{2*b}$. 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 that:

$$\frac{\mathcal{N} * \alpha}{2 * \mathbb{d}} \leq b \leq maxB$$

which translates to:

$$\mathbb{d} \geq \frac{\mathcal{N} * \alpha}{2 * maxB}.$$

For example, in our experiments (see Section 5.5), we choose $\mathcal{N} = 2^{14}$. Thus, for a full ciphertext utilization ($\alpha = 1$) at $b = 256$, we have that:

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

For a ciphertext utilization of $\alpha = 0.5$, the batch size can be as small as $b = 128$ at the same $\mathbb{d} = 32$, following a similar calculation.

5.4.1 Security Analysis

We demonstrate that RHODE fulfills the data and model confidentiality objectives (Chapter 3, Section 3.2) by arguing that a computationally-bounded adversary that controls up to $N-1$ parties (i.e., yielding a collusion among $N-1$ parties) cannot infer any information about the honest party'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 [62, 193, 181]. We rely on the proofs by Mouchet et al. [209], which show that the MHE cryptographic protocols, i.e., $\mathsf{DKeyGen}(\cdot)$ and $\mathsf{DKeySwitch}(\cdot)$, 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 and hard problem. The security of the collective

bootstrapping operation, i.e., DBootstrap(·), is proven analogously in Chapter 4.

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 [180] and consider a real-world simulator $\mathscr{S}$ that simulates a computationally-bounded adversary corrupting $N-1$ parties. During RHODE 's training protocol, the model parameters that are exchanged among parties 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 [62], is a valid encryption under CKKS, thus, achieves a security level of λ. Furthermore, the decryption of any ciphertext requires collaboration among *all* the parties that participated in DKeyGen(·). To avoid leakage due to two consecutive broadcasts, we rely on an existing countermeasure [209] 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 $\mathscr{S}$. 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 DKeySwitch(·) functionality. Hence, $\mathscr{S}$ 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 party's data and the model (following analogous arguments).

5.5 System Evaluation

We first summarize the theoretical complexity of RHODE in Section 5.5.1 before empirically evaluating it. We present our experimental setup in Section 5.5.2 and evaluate RHODE's model performance, scalability with different RNN and dataset parameters, runtime performance, and provide various microbenchmarks in Sections 5.5.3, 5.5.4, 5.5.5, and 5.5.6, respectively.

5.5.1 Complexity Analysis

We theoretically analyze 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 6). We first derive the total number of ciphertexts required which allows us to estimate the memory usage per data holder. Then, we use the number of ciphertexts to analyze the communication and computation cost of the training process by accounting 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 [209] for details). For example, the

generation of the rotation keys take 75% of the Setup Phase and the number of rotation keys depends on the application (the parameters of the RNN).

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 $\mathbb{d}$, 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. 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 $\mathscr{L} = 10$. Then, assume that $d = 16$, $b = 256$, $h = 32$, $\mathbb{d} = 32$, $\kappa = 1$ (many-to-one RNN), and $o = 1$ (i.e., a regression 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 6), 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 on row-based packing, the number of matrix rows per ciphertext is $\lfloor \frac{\mathcal{N}}{2*\mathbb{d}} \rfloor$ and the number of columns per ciphertext is $\mathbb{d}$, 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 $\lceil \frac{b}{(\mathcal{N}/2)/\mathbb{d}} \rceil = \lceil \frac{2*\mathbb{d}*b}{\mathcal{N}} \rceil$ and the number of sub-matrix columns is $\lceil \frac{c}{\mathbb{d}} \rceil$, 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:

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

$$|\mathbf{z}_t| = |\mathbf{h}_t| = \lceil \frac{2 * \mathbb{d} * b}{\mathcal{N}} \rceil * \lceil \frac{h}{\mathbb{d}} \rceil \quad \text{(hidden ciphertexts)}$$

$$|\mathbf{p}_t| = \lceil \frac{2 * \mathbb{d} * b}{\mathcal{N}} \rceil * \lceil \frac{o}{\mathbb{d}} \rceil \quad \text{(output ciphertexts)}$$

As we replicate the weight matrices for the sub-matrix operations (see Section 5.3.1), for an $n \times m$ weight matrix, the number of sub-matrix rows and columns per weight ciphertext is $\lceil \frac{n}{\mathbb{d}} \rceil$ and $\lceil \frac{m}{\mathbb{d}} \rceil$, respectively. Therefore, the total number of ciphertexts for the **weight** matrices are:

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

$$|\mathbf{W}| = \lceil \frac{h}{\mathbb{d}} \rceil * \lceil \frac{h}{\mathbb{d}} \rceil \quad \text{(hidden-weight ciphertexts)}$$

$$|\mathbf{V}| = \lceil \frac{h}{\mathbb{d}} \rceil * \lceil \frac{o}{\mathbb{d}} \rceil \quad \text{(output-weight ciphertexts)}$$

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 14, Algorithm 6) is equivalent to $\mathbf{dh} \odot (1 - \mathbf{h}_t^2)$ as the derivative of $\mathsf{Tanh}(x)$ is $1 - \mathsf{Tanh}^2(x)$. 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. The number of ciphertexts storing the weight matrices has a direct impact on RHODE's communication cost. During RNN training, communication among the data holders is triggered by: (a) the collective bootstrapping ($\mathsf{CBootstrap}(\cdot)$) operation to refresh the ciphertexts, and (b) the federated learning 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 [244] 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 5) and $\mathbf{dz}$ (Line 14) in the forward pass of each local timestep iteration (Algorithm 6). After the execution of all timesteps, the intermediate term $\mathbf{I} = \mathbf{h}_t \otimes \mathbf{dy}$ (Line 16) 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 sent/received 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.

Federated Learning (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 estimate RHODE's computational cost per data holder and per local iteration taking into consideration the *dominating terms*, i.e., the number of CBootstrap$(\cdot)$ operations (which are ~ 2 orders of magnitude slower than a homomorphic addition/multiplication), the degree $\mathfrak{p}$, and the cost of the linear transformations (i.e., σ, τ, ψ_i and ϕ_i) required for the matrix multiplication. Note that bootstrapping a ciphertext at a level $\mathscr{L}_{\boldsymbol{c}}$ (with an initial level $\mathscr{L}$) has a complexity of $C(BS) = \mathscr{N}\log_2(\mathscr{N})(\mathscr{L}+1) + \mathscr{N}\log_2(\mathscr{N})(\mathscr{L}_{\boldsymbol{c}}+1)$, homomorphic rotations have a complexity of $C(R) = \mathscr{O}(\mathscr{N}\log_2(\mathscr{N})\mathscr{L}_{\boldsymbol{c}}^2)$, and ciphertext-ciphertext multiplications have a complexity of $C(M) = 4N(\mathscr{L}_{\boldsymbol{c}}+1) + C(R)$ [244].

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 5.3.1, the multi-dimensional matrix multiplication (Algorithm 7) consumes 3 levels and yields an amortized cost of $\mathscr{O}(\mathfrak{d})$ rotations per matrix (depending on $\mathfrak{d}$, see Section 5.4). 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 $\mathscr{O}(6T\mathfrak{d})$ rotations for a local iteration of T timesteps and per ciphertext. Note that the matrix multiplication complexity predominantly comes from the linear transformations; the rest of the algorithm comprises only multiplications and additions, which are negligible in comparison. 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 of each timestep, we can pre-compute their linear transformations at the beginning of each iteration and re-use them for all timesteps. For the Example Configuration, this reduces the cost to $\mathscr{O}(6\mathfrak{d})$ per local iteration, at a memory cost of $\times$ per matrix.

5.5.2 Experimental Setup

We now present our implementation details and the datasets used for RHODE's model performance evaluation. We describe the data distribution, the RNN configuration, and the security parameters.

Implementation Details. We implement RHODE in Go [7] and employ the Lattigo [200] lattice-based HE library for the multiparty homomorphic operations. We rely on Onet [5] to build a decentralized system where the parties communicate over TCP with secure channels and Mininet [201] 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.

Datasets and Tasks. We employ the Hourly Energy Consumption (HEC) [132], Stock Prices (Stock) [261], and Inflation datasets [79], for time-series forecasting tasks. HEC contains the hourly energy consumption of several electricity distribution companies during a specific time period [132]. 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 and varying dates [261]. 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 different countries and the task is to predict the inflation rate of the next quarter given the data of previous T quarters. We also employ the Breast Cancer Wisconsin dataset (BCW) [236] that contains benign and malignant breast cancer samples for classification. More details about these datasets can be found in Appendix D.1. 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 5.5.3), we split the data among $N = 10$ parties and evaluate two different settings: (i) *Even (E)* distribution, where we uniformly distribute the dataset samples to the N parties, and (ii) *Imbalanced (I)* distribution where we simulate a heterogeneous setting, with each party having one dataset file (see Appendix D.1), e.g., for the Stock dataset, a party might be Apple Inc. contributing the stock prices of only Apple. We annotate the results per setting as "DatasetName-*Type*T" where *Type* denotes the type of the data distribution, and T the sequence length T, e.g., HEC-I10, denotes imbalanced data distribution and a sequence length of $T = 10$ 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 [312, 207, 311, 122, 137, 88] 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 [306, 72, 263, 210] 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: $\mathsf{SoftClip}(x, m)$ with a clipping threshold of $|m|$=5, approximated with a $\mathfrak{p}$=7 polynomial over the interval $[-60, 60]$ as a result of a preliminary evaluations on the datasets. To expedite the model performance experiments (Section 5.5.3), 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

Dataset	Model Performance (MAE | R^2 or Acc.)			
	L	**C**	**FL**	**RHODE**
Stock-*E*5	0.023 | 0.966	0.008 | 0.996	0.020 | 0.982	0.017 | 0.987
Stock-*I*5	0.024 | 0.976	0.008 | 0.996	0.012 | 0.992	0.012 | 0.992
HEC-*E*10	0.066 | 0.749	0.014 | 0.987	0.024 | 0.965	0.026 | 0.957
HEC-*I*10	0.081 | 0.625	0.014 | 0.987	0.025 | 0.962	0.027 | 0.955
HEC-*E*20	0.066 | 0.749	0.014 | 0.986	0.024 | 0.964	0.026 | 0.957
HEC-*I*20	0.081 | 0.626	0.014 | 0.986	0.025 | 0.962	0.027 | 0.954
Inflation-*E*8	0.067 | 0.778	0.067 | 0.775	0.067 | 0.774	0.067 | 0.771
Inflation-*I*8	0.103 | 0.579	0.066 | 0.777	0.066 | 0.777	0.067 | 0.770
BCW-*E*9	0.888	0.936	0.936	0.929
BCW-*I*9	0.880	0.936	0.900	0.914

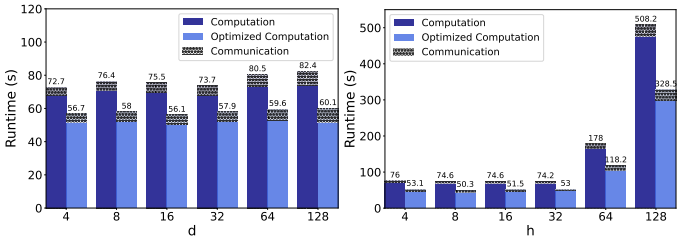
Table 5.1: RHODE*'s model performance in various settings where the Stock and HEC datasets are split among* $N = 10$ *parties. The performance is compared to three baselines: local training without collaboration (**L**), centralized (**C**), and federated learning (**FL**).*

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 [27]. We set the plaintext scale to 2^{31}, and the initial ciphertext level is $\mathcal{L} = 9$.

5.5.3 Model Performance

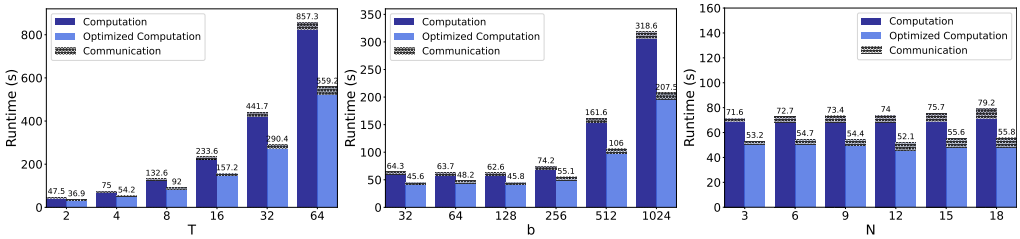
Table 5.1 shows RHODE's model performance results on the HEC and Stock datasets in various settings. For each setting, we report the mean absolute error (MAE) and R^2-scores for forecasting tasks and accuracy (Acc) for the classification one. For comparison, the table displays the performance of three baselines: (a) **L** stands for local training, where each party trains its own local model with the original network (original activation functions and clipping), i.e., without taking advantage of the other parties' inputs, (b) **C** stands for centralized training, where all the data is collected to a central server and trained with the original network (original activation functions and clipping), and (c) **FL** stands for a cross-silo federated learning approach, where the data is distributed among 10 parties and the learning is performed without any privacy mechanism. The last column shows the performance of RHODE for privacy-preserving federated learning training. The baseline column **L** shows the performance gain of collaborative training (note that we report the MAE and R^2 averaged across the 10 parties). 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, e.g., at most 0.03 difference in R^2-score compared to a centralized solution, and at most 0.01 from a federated learning (FL) approach and at most 1% accuracy difference for the classification task. We also observe that there is always a performance gain, compared



(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 parties (N).

Figure 5.2: RHODE*'s training execution times for computation and communication for one global iteration with increasing* ***(a)*** *number of features,* ***(b)*** *hidden dimension,* ***(c)*** *timesteps,* ***(d)*** *batch size, and* ***(e)*** *number of parties. The default network parameters are* $h = d = T = 4, b = 256, \mathbb{d} = 32$ *(we vary one of them in the corresponding experiment while keeping the others fixed), and the number of parties is* $N = 10$.

to **L** (except for the even distribution of the Inflation dataset where the local data already captures the global insights). For instance, 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-I10 or HEC-I20, 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 over local training without collaboration. Similarly, RHODE yields a ~3% accuracy gain on BCW compared to **L**.

5.5.4 Scalability Analysis

We evaluate RHODE's scalability by employing synthetic datasets on which we vary the number of features (d), the hidden dimension (h), the timesteps (T), and the batch size (b), and by experimenting with the number of parties (N). Our default network parameters are $h = d = T = 4$ and $b = 256$, $\mathbb{d} = 32$, the number of data providers is $N = 10$, and we vary the parameter under analysis. The calculated runtime includes average per-party Local Computation, as well as the Aggregate, and Model-Update Phases. Figure 5.2 shows the time spent by RHODE (and its optimized version with pre-computed linear transforms, described in Section 5.5.1) on computation and on communication during a global iteration. The communication indicates the total time spent for the collective bootstrapping operations (DBootstrap(·)) and the collective aggregation. Overall, we observe that the optimized computation is always more

Structure (b,d,h,T)	Runtime (s)			
	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 5.2: RHODE*'s training and prediction runtime (including communication) for several many-to-one RNN architectures with* $N = 10$ *parties. Training runtimes are for one global iteration and Prediction runtimes are for oblivious predictions (both the data and the model are encrypted) on* 256 *samples. Opt stands for optimized execution.*

efficient in terms of runtime at a memory cost of 32× more ciphertexts per weight matrix (see Section 5.5.1). Moreover, Figures 5.2a, 5.2b and 5.2d demonstrate that RHODE's computation time remains constant, with increasing the respective parameters $\{d, h, b\}$ when $d, h, b < \mathbb{d}$. This is due to the SIMD operation support by the HE scheme; as for these settings the number of matrix entries fit into one ciphertext, RHODE enables processing larger batches (more samples) or larger networks at zero additional cost.

In more detail, Figure 5.2a shows that RHODE scales sub-linearly with d and that the effect of increasing d is almost negligible in runtime. This is because each party's data remains in clear form through the training and the plaintext operations are negligible, compared to ciphertext ones. Increasing d has a direct effect on the matrix multiplication ($x_t \times \mathbf{U}$) 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, i.e., $\mathbf{h_{t-1}} \times \mathbf{W}$, dominate the runtime. The communication overhead also scales sub-linearly; increasing d increases the number of ciphertexts to be bootstrapped (see Section 5.5.1). Figure 5.2b shows that RHODE scales sub-quadratically, with increasing h when $\mathbb{d} = 32$; 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 5.5.1). This trend holds also for the communication due to an increased number of ciphertexts that are bootstrapped by the aggregation server for $\mathbf{W}$. Figures 5.2c and 5.2d demonstrate that RHODE's runtime scales linearly with increasing T or b; this is due to the sequential operations over the T timesteps or the linear increase in the number of ciphertexts processed when increasing b. Finally, Figure 5.2e shows that RHODE's local computation time remains constant with increasing number of parties, as the local computations per party are performed in parallel. The communication overhead increases linearly with increasing N, as more parties are involved in the interactive protocols, i.e., the DBootstrap(·) and the Aggregate Phase.

Dimension	Jiang et al. [143]			POSEIDON [244]			Multi-dimensional ($\mathbb{d}$ = 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 5.3: *Comparison between* RHODE*'s multi-dimensional packing to Jiang et al.'s [143] and* POSEIDON *[244] (Chapter 4) 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.*

5.5.5 Runtime Performance

Table 5.2 displays RHODE's training and prediction runtimes (for the original and optimized implementations) for various many-to-one RNN architectures with $N = 10$ parties. The Training runtime is for one global iteration (including the communication for $\mathsf{DBootstrap}(\cdot)$ 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 $\mathsf{DKeySwitch}(\cdot)$ 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 of Section 5.5.4 to estimate RHODE's total training runtime for various RNN structures. For instance, RHODE can process 256K samples split among 10 parties 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, respectively) implementation. Moreover, as the Prediction runtime is calculated for a batch of 256 samples, RHODE yields a prediction throughput of ~ 14.43 (~ 9.31) samples per second with its optimized (non-optimized, respectively) implementation, for an RNN with the same structure.

5.5.6 Microbenchmarks

We compare RHODE with prior work based on matrix multiplication microbenchmarks as they are the dominant and expensive operations in RHODE. Then, we present how different RNN structures affect the runtime of forward and backward passes.

Comparison with Prior Work. We compare RHODE's multi-dimensional packing scheme with Jiang et al.'s [143] and POSEIDON [244] packing approaches in Tables 5.3 (timings) and 5.4 (memory). We report the total and amortized time (or throughput) by executing $\#M$ multiplications of $n \times n$-size matrices in parallel.

We observe that our multi-dimensional packing approach and Jiang et al.'s one perform identically when $n = \mathbb{d}$; this is expected since Jiang et al.'s approach is a special case of our multi-dimensional packing with only one ciphertext. For $n > \mathbb{d}$ the multi-dimensional packing requires more time to complete the multiplications but achieves a better amortized time than Jiang et al's approach. Due to memory limitations, we were not able to benchmark Jiang et al.'s approach for $n = 256$ and $n = 512$. This is a consequence of each method's

memory requirements (see Table 5.4). The memory overhead arises particularly from the ring dimension ($\mathcal{N}$), the number of ciphertexts, and the size of the evaluation keys. The first row of Table 5.4 shows that, with Jiang et al.'s method, $\mathcal{N}$ scales quadratically with n compared to ours that scales similarly with $\mathbb{d}$ (or with the closest power of two above n^2 or $\mathbb{d}^2$, respectively). In other words, 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 $\mathbb{d}$ (second row of Table 5.3). However, their approach also significantly impacts the size of the evaluation keys. Indeed, as in their approach $\mathcal{N}$ scales with $\mathcal{O}(n^2)$, increasing n has the side effect of increasing the evaluation keys' size (third row of Table 5.3). Recall that Jiang et al.'s method also requires $\mathcal{O}(n)$ evaluation keys to perform the multiplication, thus resulting in an overall $\mathcal{O}(n^3)$ memory complexity. In contrast, our approach provides a better memory complexity, with only $\mathcal{O}(\mathbb{d}^3)$ that is $\mathcal{O}(1)$ for a fixed $\mathbb{d}$. In conclusion, our multi-dimensional packing is at least equivalent to Jiang et al.'s [143] and can be configured to provide a better throughput for settings with large batch sizes or hidden dimensions such that the inputs/weights do not fit into one ciphertext. It also uses a constant memory for the evaluation keys, regardless of the matrix dimension n; enabling the multiplication between large matrices, without the need for larger cryptographic parameters or the re-generation of the evaluation keys.

	[143]	POSEIDON [244]	Multi-dimensional
$\mathcal{N}$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\mathbb{d}^2)$
#Ciphertexts	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2/\mathbb{d}^2)$
Evaluation Keys (Size)	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(\mathbb{d}^3)$

Table 5.4: *Comparison of the memory complexity between* RHODE*'s multi-dimensional packing to Jiang et al.'s [143] and* POSEIDON *[244] approaches, with respect to the matrix and sub-matrix dimensions n and* $\mathbb{d}$.

We remind that we propose a system, POSEIDON [244] in Chapter 4, for training feed-forward neural networks under multiparty homomorphic encryption using a packing scheme that optimizes the learning over one sample (instead of a mini-batch) with efficient 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 the same total and amortized time, see Table 5.4). Following their approach, a vector-matrix multiplication requires $\mathcal{O}(\log n)$ inner sum rotations, and repeating this n times to achieve a matrix multiplication yields a complexity of $\mathcal{O}(n \log n)$ for a fixed $\mathcal{N}$ (increasing $\mathcal{N}$ for bigger matrices adds a factor of $\mathcal{O}(\mathcal{N})$ to this complexity). For example, for the multiplication of two 32×32 matrices, our multi-dimensional packing is $\sim 14\times$ and $\sim 122\times$ faster than POSEIDON's packing approach for total and amortized times, respectively. Similar to Jiang et al., the memory complexity of their approach is bound to n for configuring $\mathcal{N}$ and hence the size of the evaluation keys (see Table 5.3), whereas our multi-dimensional packing decouples the memory complexity from n. As a result, employing the one-sample processing approach by

POSEIDON to train RNNs would significantly increase computational complexity due to the large number of homomorphic rotations and vector-matrix multiplications that are required for the forward and backward pass of the various timesteps and for the multiple samples of an input batch. We estimate that if POSEIDON's framework was extended to RNNs, training an RNN with $d = 32$, $h = 32$, $b = 256$, and $T = 4$ would be $\sim 25 - 32\times$ slower compared to RHODE and its optimized version.

Finally, we discuss another packing scheme similar to our multi-dimensional packing. Aharoni et al., improving on earlier work [23], propose a packing scheme that leverages on the CKKS complex domain to enable dense packing [24]. Their approach is based on data structures that pack tensors, e.g., matrices or hypercubes, in fixed sized chunks (tiles), thus enabling a natural use of the SIMD capabilities. Yet, 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, which implies that the result will be sparsely packed compared to our technique that fully-utilizes the ciphertext slots. Furthermore, their approach is not format-preserving, i.e., it requires pre- or post-processing for each multiplication, whereas our approach supports chaining operations without additional processing steps. Thus, even though their approach consumes two levels less than Jiang et al. [143] (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 $\mathbb{d}$, 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 nor matrix multiplication microbenchmarks, an experimental comparison with our multi-dimensional packing is not possible.

RNN Structure. We evaluate RHODE's performance for different RNN structures including many-to-one and many-to-many. Table 5.5 depicts the results of microbenchmarks for the forward and backward passes *per timestep* without bootstrapping and *without optimization* for different RNN structures. FP, BP, FP-t_o, and BP-t_o, stands for the forward pass, the backward pass, 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 $\kappa = 1$. 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 $\kappa = 2$, 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 $T = \kappa = 6$. Overall, Table 5.5 shows that the forward and backward pass calculations of the output-stage timestep (FP-t_o, BP-t_o) are more costly than the usual FP and BP timesteps ($1.7\times$). This is due to the extra multiplication required in the FP for calculating the prediction (Line 5, Algorithm 6), and the extra subtraction and multiplication during the BP (Lines 11-12, Algorithm 6) for the output error calculation. Recall that Algorithm 6 is presented for a many-to-many RNN structure with $\kappa = T$ 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 increase in the runtime depends on κ.

We here note that a Jordan network requires the calculation of the output y_t at every timestep (see Section 1). 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 5.5, 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(·) operation during the backward pass, resulting in slightly higher communication than Elman networks.

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 5.5: *Microbenchmarks for the forward pass (FP) and backward pass (BP) per timestep for various RNN structures without optimization. t_o represents the timestep at the output stage.*

5.6 Conclusion

In this chapter, we presented RHODE, a novel system that enables privacy-preserving training of and predictions on Recurrent Neural Networks (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 novel multi-dimensional packing scheme and polynomial approximations for clipping and activation functions, 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 parties, 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 and its variants under encryption. As future work, we plan to evaluate RHODE on sequential data types other than time-series and on more complex recurrent neural network architectures.

6 Federated Neural Network Learning for Disease-Associated Cell Classification

6.1 Introduction

In this chapter, we address the privacy-preserving training and evaluation of a convolutional neural network (CNN) that is tailored for single-cell analysis to show-case the maturity of our contributions in this thesis. We enable the under-encryption training of a published state-of-the-art CNN in an N-party setting by relying on MHE.

Single-cell analysis has been a trending topic over the last decade. This technique involves collecting cells from different patients in clinical applications to identify cell types, which is of significant value for biomedicine [170, 265]. Single-cell analysis is a powerful tool for studying the molecular and functional characteristics of *individual* cells, and it enables investigations of genomics, transcriptomics, proteomics, metabolomics, and cell–cell interactions at the single-cell level. Before the advent of single-cell analysis, bulk populations of cells collected from a patient's tissue were studied to discover mechanisms. However, with the discovery of cellular heterogeneity, analyzing individual cells has become a powerful approach to uncovering mechanisms that may be masked in bulk cell studies.

Machine learning models, in particular neural networks, extract valuable insights from data and have achieved unprecedented predictive performance in the healthcare domain, e.g., in single-cell analysis [282], aiding medical diagnosis and treatment [164, 275], or in personalized medicine [273]. Training accurate and unbiased models without overfitting requires access to a large amount of diverse data that is usually isolated and scattered across different healthcare institutions [150]. As mentioned earlier, sharing or transferring personal healthcare data is, however, often unfeasible or limited due to privacy regulations such as GDPR [8] or HIPAA [3]. Consequently, privacy-preserving collaborative learning solutions play a particularly important role for studies that involve novel informative, yet not universally established, data modalities such as high dimensional single-cell measurements, where the number of examples is typically low at individual study centers and only amounts to critical mass for the successful training of machine learning models across multiple study centers [235]. The ability to satisfy privacy regulations in an efficient and effective manner constitutes a pivotal

requirement to carry out translational multi-center studies.

Federated learning (FL) has enabled collaborative learning for several medical applications, and it has been shown that federated learning performs comparably to centralized training on medical datasets [240, 250, 109]. Recently, the concept of swarm learning has been proposed; it enables decentralized machine learning for precision medicine. The seminal work of swarm learning [284] is based on edge computing and permissioned blockchains and removes the need for a central server in the federated learning approach. Despite the advantages of federated learning and swarm learning for keeping the sensitive data local and for reducing the amount of data transferred/outsourced, the model and the intermediate values exchanged between the parties remain prone to several privacy attacks executed by the other parties or the aggregator (in federated learning), such as membership inference attacks [199, 212] or reconstructing the parties' inputs [128, 283, 309]. In this chapter, we provide a solution that further conceals the global machine learning model from the participants, as in previous chapters, by relying on mathematically secure cryptographic techniques to mitigate these inference attacks in the *healthcare framework*.

In order to mitigate or prevent the leakage in the federated learning setting, several privacy-preserving mechanisms have been proposed that are presented in Chapter 2. These mechanisms are classified under three main categories, depending on the strategy they are based on: differential privacy (DP), secure multiparty computation (MPC), and homomorphic encryption (HE).

We remind that differential privacy (DP)-based solutions aim to perturb the parties' input data or the intermediate model values exchanged throughout the learning. Several studies in the medical domain keep the data on the local premises and use federated learning with a differential privacy-mechanism on the exchanged model parameters [70, 161, 177]. Despite being a pioneering mitigation against privacy attacks, DP-based solutions perturb the model parameters, thus decreasing the utility and making the deployment harder for medical applications, where the accuracy is already constrained by limited data. Quantification of the privacy achieved via DP-based approaches is also very difficult [141] and the implementation of DP, especially in medical imaging applications, is not a trivial task [150].

MPC techniques are also applied to ensure privacy and to enable collaborative training of machine learning models [138, 67, 74, 151, 127]. MPC techniques rely on secret-sharing the data of the parties and on performing the training on the secret-shared data among multiple computing nodes (usually 2,3, or 4 nodes). Nevertheless, it is usually hard to deploy these solutions, as they often rely on a trusted third party for the sake of efficiency. Moreover, their scalability with the number of parties is poor due to the large communication overhead.

Finally, several works employ homomorphic encryption (HE) to enable secure aggregation or to secure outsourcing of the training in the medical appliacation to a cloud server [162, 45]. These solutions, however, cannot solve the distributed scenario where parties keep their local data in their premises.

The adoption of each of the aforementioned solutions introduces several privacy, utility, and performance trade-offs that need to be carefully balanced for healthcare applications. To balance these trade-offs, several works employ multiparty homomorphic encryption (MHE) [103, 230]. Although the underlying model in these solutions enables privacy-preserving distributed computations and maintains the local data of the parties on their local premises, the functionality of these works is limited to the execution of simple operations, i.e., basic statistics, counting, or linear regression and the underlying protocols do not support an efficient execution of neural networks in the federated learning setting. We proposed in Chapter 4 a more versatile solution, POSEIDON, for enabling privacy-preserving federated learning for neural networks by relying on MHE [244] to mitigate federated learning inference attacks by keeping the model and intermediate values encrypted. Our solution, however, does not address the efficient implementation and execution of convolutional neural networks that are tailored to analyze complex data types such as single-cell data.

In this chapter, we propose PRICELL, a novel solution based on MHE to enable the training of a federated convolutional neural network in a privacy-preserving manner, thus preserving the utility of the data for single-cell analysis. To the best of our knowledge, PRICELL is the first of its kind in the regime of privacy-preserving multi-center single-cell studies under encryption. By bringing privacy-by-design and by preventing the transfer of patients' data to other institutions, our work contributes to single-cell studies and streamlines the slow and demanding process of the reviewing of independent ethics committees for consent forms and study protocols. To mitigate federated learning attacks, we keep the model and any value that is exchanged between the parties in an encrypted form, and we rely on the threat model and setting proposed in Section 3.1.

By designing new packing strategies and homomorphic matrix operations, we improve the performance of the protocols for encrypted convolutional neural networks (CNNs) that are predominantly used in the healthcare domain [234]. To evaluate our system within the framework of single-cell analysis, we train a convolutional neural network (CellCnn), designed by Arvaniti and Claassen [32], within our privacy-preserving system for the disease classification task. We also show the feasibility of our solution with several single-cell datasets utilized for cytomegalovirus infection (CMV) [131] and acute myeloid leukaemia (AML) [175] classification, and one dataset for non-inflammatory neurological disease (NIND) and relapsing–remitting multiple sclerosis (RRMS) [106] classification. We compare our classification accuracy in a privacy-preserving federated learning setting with the centralized and non-encrypted baseline. Our solution converges comparably to the training with centralized data, and we improve on our contributions with POSEIDON, in Chapter 4, in terms of training time. For example, in a setting with 10 parties, we improve POSEIDON's execution time by at least one order of magnitude.

Acknowledgements. This work is made possible by the collaboration between Jean-Philippe Bossuat, Dr. Juan R. Troncoso-Pastoriza, Prof. Manfred Claassen, and Prof. Jean-Pierre Hubaux. We would like to also thank Apostolos Pyrgelis, David Froelicher, and Sylvain Chatel who gave

valuable feedback on the manuscript. We also thank Shufan Wang and Joao Sa Sousa for their contribution on the experiments and benchmarking.

6.2 PRICELL Design

PRICELL's system and threat model is based on the model described in Section 3.1. In PRICELL's scenario, there are N healthcare institutions (parties), each holding its own patient dataset and collectively training a CNN model, without sharing/transferring their local data. Our aim is to preserve the confidentiality of the local data, the intermediate model updates in the federated learning setting, the querier's evaluation data, and optionally the final model. We remind that we rely on a synchronous learning protocol, assuming that all parties are available throughout the training and evaluation executions. We note here that this assumption can be relaxed by using different HE schemes, such as threshold or multi-key HE [191, 249], but with a relaxed security assumption for the former and an increased computation cost for the latter.

We summarized PRICELL's system and its workflow for collaborative training and query evaluation (prediction), in Figure 3.2. As introduced earlier, the training pipeline is composed of four phases: **Setup Phase**, **Local Computation Phase**, **Aggregation Phase**, and **Model-Update Phase**; and PaaS is enabled for the predictions to the querier. Below, we define the system-specific **Local Gradient Descent Computation** that refers to Line 9 of Algorithm 2. Assuming there are four healthcare institutions with each holding its respective secret key, the workflow starts with the generation of a collective public key and a set of evaluation keys that are necessary for the encrypted operations, using each participant's secret key (**Setup Phase**). Then, the participants agree on the initial random global model weights (W_g) and encrypt them with the collective public key. We denote the encryption of any value with boldface letters, i.e., $\boldsymbol{W_g}$. After encrypting the initial global weights, the local computation begins. To find the model gradients ($\nabla \boldsymbol{W_k}$), each party performs several encrypted training iterations on their local data (**Local Computation Phase**). The local-model gradients are then sent and aggregated at one of the parties that will perform the global model update (**Model-Update Phase**). The updated model is then broadcast back. After a fixed number of training iterations, the participants can choose to keep the model confidential (option 5.1 in Figure 3.2) or to decrypt it for further analysis (option 5.2 in Figure 3.2).

If prediction-as-a-service is offered to a querier (a researcher) and the model is kept encrypted, the querier must encrypt the evaluation data ($\boldsymbol{X_q}$) with the collective public key of the parties. Once the prediction is done, the result ($\hat{\boldsymbol{y}}$) is collectively switched to the public key of the querier by using the underlying cryptoscheme's collective key switching functionality. If the model is instead decrypted, the querier encrypts the data with their own key hence no key switch is needed after the prediction. As a result, regardless of the model being confidential or not, the evaluation data of the querier and the prediction result always remain protected, as only the querier can decrypt the end result.

We describe the CNN model that is used in **Local Computation Phase** in Section 6.2.1. We

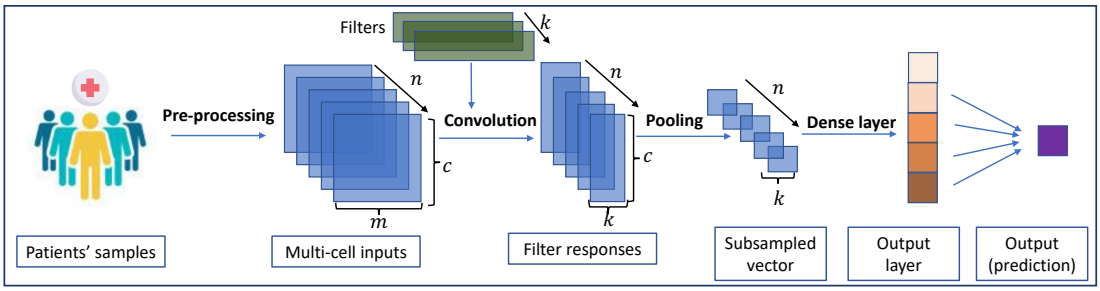


Figure 6.1: *CellCnn [32] Neural Network Architecture that is used in* ***Local Computation Phase****. The network takes multi-cell samples as an input and applies a 1D convolution with h filters followed by a pooling layer. A dense (fully-connected) layer then outputs the phenotype prediction.*

listed the frequently used symbols and notations in Section 1.4.

6.2.1 CellCnn Model Overview

CellCnn is a convolutional neural network that enables multi-instance learning and associates a set of observations on cellular population, namely multi-cell inputs, with a phenotype [32]. This architecture is designed for detecting rare cell subsets associated with a disease, by using multi-cell inputs generated from high-dimensional single-cell marker measurements. By their nature, these inputs can be used to predict the phenotype of a donor or the associated phenotype for a given subset of cells. In this scenario, we enable privacy-preserving and distributed multi-instance learning, and we compare our classification performance with the baseline (CellCnn [32] trained on centralized data with no privacy protection). We note here that replicating the full-pipeline of CellCnn [32] for downstream analysis requires either heavy approximations under encryption or the decryption of the trained model. Our solution enables the collective and privacy-preserving training for the classification task, whereas subsequent analyses that require access to the model are out-of-the-scope of this work. Yet, we show the negligible effect that our encryption would practically have on these analyses in Appendix E.3.

We show the architecture of CellCnn [32] in Figure 6.1. The network comprises a 1D convolutional layer followed by a pooling layer and a dense (fully-connected) layer. Each multi-cell input sample in Figure 6.1 is generated using c cells per phenotype with m features (markers), and these samples are batched to construct multi-cell inputs. The training set is then generated by choosing z multi-cell inputs per output label or per patient.

We refer the reader to the work of Arvaniti and Claassen [32] for the details of the neural network architecture. We detail the changes we introduce to this architecture to enable operations under homomorphic encryption in Section 6.2.2.

6.2.2 Local Neural Network Operations

In this section, we give a high-level description of the neural network circuit that is evaluated in the encrypted domain (a detailed and step-by-step description can be found in Section 6.3). We first present the changes introduced to the original CellCnn circuit to enable an efficient evaluation under encryption: (i) we approximate the non-polynomial activation functions by polynomials by using least-squares approximation, (ii) we replace the max pooling with an average pooling, (iii) we replace the ADAM optimizer with the stochastic gradient descent (SGD) optimizer with mean-squared error and momentum acceleration. Finally, we introduce the packing strategy used in PRICELL and give a high-level circuit overview. We give more details on these steps and optimizations, in Section 6.3, and we empirically evaluate the effect of these optimizations on the model accuracy in a distributed setting in Section 6.4.

Polynomial Approximations. We remind that with additions and multiplications, the CKKS scheme can efficiently evaluate polynomials in the encrypted domain. However, these two basic operations are not sufficient for easily evaluating non-linear functions, such as sign or sigmoid. A common strategy that we rely on in this thesis to circumvent this problem is to find a polynomial approximation of the desired function. We rely on polynomial least-squares approximations for the non-polynomial activation functions, such as sigmoid, and we use identity function after convolution (instead of ReLU). We show in Section 6.4 that these changes only have a negligible effect on the model accuracy.

Pooling. The original CellCnn circuit makes use of both max pooling and average pooling. Max pooling requires the computation of the non-linear sign function which cannot be efficiently done under encryption. We replace the max pooling with the average pooling, which is a linear transformation and brings the following advantages: (i) it is efficient for computing under encryption with only additions and constant multiplication, (ii) it simplifies the backward pass under encryption, (iii) it commutes with other linear transformations or functions, such as the convolution and the identity activation, which allows for an efficient preprocessing of the data and reduces the online execution cost. Indeed, we are able to pre-compute the average pooling on the data, which reduces the input size of a batch of samples from $n \times c \times m$ to $n \times m$, i.e., we remove the dependency on c.

Optimizer. The original CellCnn training relies on the ADAM optimizer, which requires the computation of square roots and inverses. Although approximating these operations is possible, a high-precision approximation requires an excessive use of ciphertext levels and significantly reduces the efficiency of the training. To avoid these costly operations, we rely instead on the SGD optimizer with momentum acceleration that, for an equivalent amount of epochs, shows a comparable rate of convergence to the ADAM optimizer.

Packing Strategy. The CKKS scheme provides complex arithmetic on $\mathbb{C}^{\mathcal{N}/2}$ in a SIMD fashion. The native operations are addition, multiplication by a constant, multiplication by a plaintext, multiplication by a ciphertext, slots rotation (shifting the values in the vector), and complex conjugation. As the rotations are expensive, when considering encrypted matrix operations, one of the main challenges is to minimize the number of rotations, which can be done by adopting efficient packing strategies and algorithms. We give more details about the packing

and algorithms, in Section 6.3.

With the aforementioned pre-computed pooling, only a $1 \times m$ vector is needed to represent a sample, instead of a $c \times m$ matrix. Hence, we pack an entire batch of n samples in a single ciphertext and compute the forward and backward pass on the whole batch in parallel, which reduces the complexity of the training proportionally to the size of the batch.
Encrypted Circuit Overview. Given a batch size of n samples, each sample being a matrix $L_{c\times m}$, for c the number of cells per sample and m number of features (markers) per cell, we first evaluate the mean pooling across the cells in plaintext. The result is a set of n vectors of size $1 \times m$, which is packed in an $L_{n\times m}$ matrix. The 1D convolution is evaluated with a $L_{n\times m} \times C_{m\times k}$ matrix multiplication. We feed the result to the dense layer $W_{k\times o}$ where o is the number of output labels. Lastly, we perform an approximated activation function to the output of the dense layer. Our encrypted circuit with reduced complexity is

$$Y_{n\times o} = \mathsf{Poly}_{\mathsf{act}}\Big(\big(\mathsf{MeanPool}(L_{n\times c\times m}) \times C_{m\times k}\big) \times W_{k\times o}\Big).$$

In the next section (Section 6.3), we detail each of the aforementioned building block (pooling, convolution, packing) of the encrypted circuit and present several optimizations to improve the efficiency under encryption.

6.3 Cryptographic Operations and Optimizations

Notations. We denote a batch of samples, the convolution layer and the dense layer matrices by $L_{n\times c\times m}$, $\boldsymbol{C}_{m\times h}$ and $\boldsymbol{W}_{h\times o}$, respectively, with n the number of samples, c the number of cells per sample per batch, m the number of features (markers), h the number of filters and o the number of output classes (labels). When there is no ambiguity, we eliminate the sub-index of the matrices, e.g., $\boldsymbol{C}_{m\times h}$ is often referred to as $\boldsymbol{C}$. We recall that encrypted matrices are denoted in boldface. We denote the plaintext of binary values as *mask*. When multiplied with a ciphertext, *mask* selects specific slots of the ciphertext by setting the other slots to zero. The terms *row*, *column* and *diagonal* encoding of a matrix denote the mapping of a 2D matrix on a 1D vector by concatenating each row, each column or each diagonal of the matrix respectively.
Convolution With Pre-Pooling. Given a hyper-cube batch of samples $L_{n\times c\times m}$, we first preprocess L by applying the average pooling across the cells. As the convolution, the average pooling, and the activation of this step are all linear transformations, their order is interchangeable. This preprocessing reduces the size of the hyper-cube from $n \times c \times m$ to only $n \times m$, thus removing its dependency on c. The convolution is computed with a single matrix multiplication $\boldsymbol{P}_{n\times h} = L_{n\times m} \times \boldsymbol{C}_{m\times h}$, with a row-encoded $\boldsymbol{P}_{n\times h}$ matrix where each row stores the result of the convolution layer for one sample. In the rest of this section, we describe how we pack $L_{n\times m}$ and $\boldsymbol{C}_{m\times h}$ in order to enable an efficient convolution through SIMD operations.

We evaluate the convolution with a diagonally-encoded plaintext and row-encoded ciphertext matrix multiplication. As we operate with non-square matrices, we pad the matrix $\boldsymbol{C}$ with

the copies of itself until its number of rows reaches n. As such, the result will yield n rows, each of h values. With this approach, the convolution can be evaluated with only m plaintext-ciphertext multiplications and additions, and $m-1$ rotations. If $m \times n$ is not a power of two, cyclic rotations of the ciphertext slots will not result in a cyclic rotation of the flattened matrix. Instead, it requires using the *masking* and rotations, which consumes a level. To overcome this, we pad the flattened matrix with additional copies of itself until it reaches a total of $n+(m-1)$ rows (hence the final size of the flattened matrix is $h(n+m-1)$). This enables us, at the expense of more slots used, to simulate a cyclic rotation. Note that those extra rows are removed by the plaintext multiplication by L that also acts as a masking.

We further reduce the number of operations by making use of complex arithmetic, which is natively provided by the CKKS scheme. Using the following, we compute the dot product of $\langle (a_0, a_1), (b_0, b_1) \rangle$ in a single multiplication:

$$(a_0 - i\,a_1) \cdot (b_0 + i\,b_1) = (a_0 b_0 + a_1 b_1) + i(a_0 b_1 - a_1 b_0).$$

Hence, the convolution of two half-sized complex matrices $L'_{n,m/2} \times B'_{m/2,h}$ (the matrix is padded in case of odd number of rows/columns) is sufficient to compute the convolution of the real matrices $L_{n\times m} \times \boldsymbol{C}_{m\times h}$:

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,m} \\ \vdots & \ddots & \\ a_{n,1} & \dots & a_{n,m} \end{pmatrix}_{(n\times m)} \times \begin{pmatrix} b_{1,1} & \dots & b_{1,h} \\ \vdots & \ddots & \vdots \\ b_{m,1} & \dots & b_{m,h} \end{pmatrix}_{(m\times h)} \rightarrow \begin{pmatrix} a_{1,1} - i\,a_{1,2} & \dots & a_{1,m-1} - i\,a_{1,m} \\ \vdots & \ddots & \vdots \\ a_{n,1} - i\,a_{n,2} & \dots & a_{n,m-1} - i\,a_{n,m} \end{pmatrix}_{(n\times m/2)} \times \begin{pmatrix} b_{1,1} + i\,b_{2,1} & \dots & b_{1,m} + i\,b_{2,m} \\ \vdots & \ddots & \vdots \\ b_{n-1,1} + i\,b_{n,1} & \dots & b_{n-1,m-1} + i\,b_{n,m} \end{pmatrix}_{(m/2\times h)}$$

Below we present a toy-example with 2×4 and 4×3 matrices:

$$\begin{pmatrix} 1 & 2 & 4 & 1 \\ 3 & 5 & 6 & 2 \end{pmatrix}_{(2\times 4)} \times \begin{pmatrix} 4 & 2 & 1 \\ 9 & 5 & 3 \\ 2 & 2 & 3 \\ 3 & 5 & 6 \end{pmatrix}_{(4\times 3)} \rightarrow \begin{pmatrix} 33 & 25 & 25 \\ 75 & 53 & 48 \end{pmatrix}_{(2\times 3)}$$

becomes

$$\begin{pmatrix} 1-2i & 4-i \\ 3-5i & 6-2i \end{pmatrix}_{(2\times 2)} \times \begin{pmatrix} 4+9i & 2+5i & 1+3i \\ 2+3i & 2+5i & 3+6i \end{pmatrix}_{(2\times 3)} \rightarrow \begin{pmatrix} (1-2i)\times(4+9i) + (4-i)\times(2+3i) & \cdots \\ (3-5i)\times(4+9i) + (6-2i)\times(2+3i) & \cdots \end{pmatrix}_{(2\times 3)}$$

$$\rightarrow \begin{pmatrix} (4+18-8i+9i) + (8+3-2i+12i) & \cdots \\ (12+45-20i+27i) + (12+6-4i+18i) & \cdots \end{pmatrix}_{(2\times 3)}$$

$$\rightarrow \begin{pmatrix} 33+11i & \cdots \\ 75+21i & \cdots \end{pmatrix}_{(2\times 3)}$$

The extraction of the real part can be done with complex conjugation and addition. The mapping from $\boldsymbol{C}_{m,h}$ to $\boldsymbol{C}'_{m/2,h}$ is straightforward and can be homomorphically computed with

$\boldsymbol{C}' = \boldsymbol{C} + \mathsf{RotL}_h(\mathsf{MultImag}(\boldsymbol{C}))$. Note that it requires $\boldsymbol{C}$ to be padded with an additional row. The encoding of the plaintext matrix $L_{n,m}$ is done by encoding each diagonal of $L'_{n,m/2}$ in a separate plaintext.

The matrix multiplication $L' \times \boldsymbol{C}'$ is then done with

$$\boldsymbol{P}'_{n\times h} = \sum_{i=0}^{m/2-1} L'_{n\times m/2} \odot \mathsf{RotL}_{2hi}(\boldsymbol{C}'_{m/2\times h}).$$

The result is a row-encoded $n \times h$ complex matrix. We remove its imaginary part $\boldsymbol{P} = \frac{1}{2}(\boldsymbol{P'} + \mathsf{Conjugate}(\boldsymbol{P'}))$ with the $\frac{1}{2}$ factor being pre-applied to L'. Because the number of rotations is reduced by a factor of two, the number of rows for the padding must also be readjusted:

$$\underbrace{n}_{\text{result}} + \underbrace{(\lceil m/2 \rceil - 1)\cdot 2}_{\text{rotations}} + \underbrace{1}_{i \text{ repacking}},$$

and the total number of slots used to encode $\boldsymbol{C}_{m\times h}$ is $nh + (\lceil m/2 \rceil - 1)2h + h$. We give an overview of how $\boldsymbol{C}_{n\times h}$ and $\boldsymbol{P}_{n\times h}$ are each encoded on a vector:

$$\boldsymbol{C}_{m\times h} = (\underbrace{\boldsymbol{C}_{(1,1)}, \ldots, \boldsymbol{C}_{(1,h)}, \boldsymbol{C}_{(2,1)}, \ldots, \boldsymbol{C}_{(2,h)}, \ldots, \boldsymbol{C}_{m,1}, \ldots, \boldsymbol{C}_{(m,h)}, \boldsymbol{C}_{(1,1)}, \ldots}_{nh+(m/2-1)2h+h}, 0, \ldots, 0),$$

$$\boldsymbol{P}_{n\times h} = \big(\boldsymbol{P}_{(1,1)}, \ldots, \boldsymbol{P}_{(1,h)}, \ldots, \boldsymbol{P}_{(n,1)}, \ldots, \boldsymbol{P}_{(n,h)}, 0, \ldots, 0\big).$$

Dense Layer. The input to the dense layer is a row-encoded $\boldsymbol{P}_{n\times h}$ matrix that is multiplied with the $\boldsymbol{W}_{h\times o}$ matrix. As the matrix $\boldsymbol{P}_{n\times h}$ is row-encoded and requires a homomorphic extraction of its diagonals, the technique used in the convolution step becomes costly for the dense layer. Instead, we use the multiply-then-inner-sum approach, as in POSEIDON [244]. The values of $\boldsymbol{W}_{h\times o}$ are grouped by samples. We first preprocess $\boldsymbol{P}$ by duplicating it o times for each label. This duplication is done with $\log_2(o) + \mathsf{hw}(o) - 1$ rotations. The matrix $\boldsymbol{W}_{h\times o}$ is column-encoded (row-encoding of its transpose), with each of its columns replicated n times (for each sample):

$$\boldsymbol{W}_{h\times o} = \big(\underbrace{(\boldsymbol{W}_{(1,1)}, \ldots, \boldsymbol{W}_{(h,1)}), \ldots, (\boldsymbol{W}_{(1,1)}, \ldots, \boldsymbol{W}_{(h,1)})}_{n\times h}, \ldots, (\boldsymbol{W}_{(1,o)}, \ldots, \boldsymbol{W}_{(h,o)}), \ldots, (\boldsymbol{W}_{(1,o)}, \ldots, \boldsymbol{W}_{(h,o)}), 0, \ldots, 0\big).$$

The multiplication $\boldsymbol{U}_{n\times o} = \boldsymbol{P}_{n\times h} \times \boldsymbol{W}_{h\times o}$ is carried on with a single ciphertext-ciphertext multiplication, followed by an inner-sum of batch n and h ($\log_2(h) + \mathrm{hw}(h) - 1$ rotations). The resulting vector has a size of onh:

$$\boldsymbol{U}_{n\times o} = \big(\underbrace{(\boldsymbol{U}_{(1,1)}, \times, \ldots, \times), \ldots, (\boldsymbol{U}_{(n,1)}, \times, \ldots, \times)}_{n\times h}, \ldots, (\boldsymbol{U}_{(1,o)}, \times, \ldots, \times), \ldots, (\boldsymbol{U}_{(n,o)}, \times, \ldots, \times), 0, \ldots, 0, \underbrace{(\times, \ldots, \times)}_{h-1}\big),$$

with $\times$ denoting unusable by-product values in the ciphertext slots.
Repacking for Bootstrapping. We repack the following elements in a single ciphertext for the optimized bootstrapping:

- $\boldsymbol{U}_{n\times o}$: the result of the dense layer, which uses $onh + h - 1$ slots.
- $\boldsymbol{P}_{n\times h}$: the result of the convolution layer, which uses nh slots.
- $\boldsymbol{W}_{h\times o}$: the dense layer matrix, which uses onh slots.
- $\nabla \boldsymbol{W}^{\mathsf{prev}}_{h\times o}$: the updated dense layer weights of the previous backward pass, which uses onh slots.
- $\nabla \boldsymbol{C}^{\mathsf{prev}}_{m\times h}$: the updated convolution layer weights of the previous backward pass, which uses size $nh + (\lceil m/2 \rceil - 1) \cdot 2h + h$ slots.

The repacking is done solely with additions and rotations, concatenating the empty slots of $\boldsymbol{U}_{n\times o}$:

$$\boldsymbol{D}_{\mathsf{repack}} = \boldsymbol{U}_{n\times o} + \mathsf{RotL}_{-onh}(\boldsymbol{P}_{n\times h}) + \mathsf{RotL}_{-2onh}(\boldsymbol{W}_{h\times o}) + \mathsf{RotL}_{-3onh}(\nabla \boldsymbol{W}^{\mathsf{prev}}_{h\times o}) + \mathsf{RotL}_{-4onh}(\nabla \boldsymbol{C}^{\mathsf{prev}}_{m\times h}).$$

Bootstrapping and Repacking for Backward Pass. The goal of this step is to refresh the ciphertext $\boldsymbol{D}_{\mathsf{repack}}$ to a higher level, to enable more computation and to re-arrange its slots optimally for the backward pass.

- $\boldsymbol{U}_{n\times o}$: We re-order the slots to arrange them first by samples then by classes, and we duplicate each value h times (replacing the non-zero by-product slots):

  $$\boldsymbol{U}_{\mathsf{backW}} = \big(\underbrace{(\boldsymbol{U}_{(0,0)}, \ldots, \boldsymbol{U}_{(0,0)}, \boldsymbol{U}_{(0,1)}, \ldots, \boldsymbol{U}_{(0,1)})}_{2h}, \ldots, (\boldsymbol{U}_{(n-1,0)}, \ldots, \boldsymbol{U}_{(n-1,0)}, \boldsymbol{U}_{(n-1,1)}, \ldots, \boldsymbol{U}_{(n-1,1)})\big).$$

  We note that the size of this vector remains onh. $\boldsymbol{U}_{\mathsf{backW}}$ will be used to compute the dense layer error for the updated dense layer weights. We pack an additional copy of $\boldsymbol{U}$, $\boldsymbol{U}_{\mathsf{backC}}$, which is pre-formatted for the convolution layer error and clustered by

sample. By computing twice the same values in parallel, but packed in two different ways (one for the dense layer, one for the convolution layer), we avoid expensive and level-consuming repacking procedures, at the cost of more slot usage. Hence for each label, we pad the nh values with $(m/2-1)2h+h$ additional copies of the relevant rows. The used size is therefore $(nh+(m/2-1)2h+h)o$.

$$\boldsymbol{U}_{\mathsf{backC}} = \Big(\underbrace{(\underbrace{\boldsymbol{U}_{(1,1)},\ldots,\boldsymbol{U}_{(1,1)}}_{h},\ldots,\boldsymbol{U}_{(n,1)},\ldots,\boldsymbol{U}_{(n-1,0)},\boldsymbol{U}_{(1,1),\ldots})}_{nh+(m/2-1)2h+h},\ldots,(\boldsymbol{U}_{(1,o)},\ldots,\boldsymbol{U}_{(1,o)},\ldots,(\boldsymbol{U}_{(n,o)},\ldots,\boldsymbol{U}_{(n,o)},\boldsymbol{U}_{(1,o)},\ldots)\Big).$$

- $\boldsymbol{P}_{n\times h}$: The result of the convolution layer, which is an $n\times h$ row-encoded matrix, is re-arranged by duplicating each of its rows for each class of the dense layer (2 in this example) and multiplied by the learning rate (η).

$$\boldsymbol{P}_{\mathsf{back}} = \eta\Big((\underbrace{\boldsymbol{P}_{(1,1)},\ldots,\boldsymbol{P}_{(1,h)},\boldsymbol{P}_{(1,1)},\ldots,\boldsymbol{P}_{(1,h)}}_{2h}),\ldots,(\boldsymbol{P}_{(n,1)},\ldots,\boldsymbol{P}_{(n,h)},\boldsymbol{P}_{(n,1)},\ldots,\boldsymbol{P}_{(n,h)})\Big).$$

- $\boldsymbol{W}_{h\times o}$: The dense layer matrix, which is an $h\times o$ column-encoded matrix, is re-arranged by padding each column with itself such that each column has a size of $nh+(m/2-1)2h+h$, for a total size of $o(n\times h+(m/2-1)2h+h)$ and is multiplied by the learning rate (η).

$$\boldsymbol{W}_{\mathsf{back}} = \eta\Big((\underbrace{\boldsymbol{W}_{(1,1)},\ldots,\boldsymbol{W}_{(h,1)},\boldsymbol{W}_{(1,1)},\ldots}_{nh+(m/2-1)2h+h}),\ldots,(\boldsymbol{W}_{(1,o)},\ldots,\boldsymbol{W}_{(h,o)},\boldsymbol{W}_{(1,o)},\ldots)\Big).$$

- $\nabla\boldsymbol{W}^{\mathsf{prev}}_{h\times o}$: The previous dense layer updated weights, of size nho. The format is preserved (column encoded matrix of size ho with each column replicated n times), but the values are multiplied by the momentum (μ).

$$\nabla\boldsymbol{W}_{\mathsf{back}} = \mu\Big(\underbrace{(\nabla\boldsymbol{W}_{(1,1)},\ldots,\nabla\boldsymbol{W}_{(h,1)}),\ldots,(\nabla\boldsymbol{W}_{(1,1)},\ldots,\nabla\boldsymbol{W}_{(h,1)})}_{n\times h},\ldots,(\nabla\boldsymbol{W}_{(1,o)},\ldots,\nabla\boldsymbol{W}_{(h,o)}),\ldots,(\nabla\boldsymbol{W}_{(1,o)},\ldots,\nabla\boldsymbol{W}_{(h,o)})\Big).$$

- $\nabla\boldsymbol{C}^{\mathsf{prev}}_{m\times h}$: The previous convolution layer updated weights, of size $n\times h+(m/2-1)2h+h$. The format is preserved (row encoded matrix, padded), but the values are multiplied by the momentum (μ).

$$\nabla\boldsymbol{C}_{\mathsf{back}} = \mu(\underbrace{\nabla\boldsymbol{C}_{(1,1)},\nabla\boldsymbol{C}_{(1,2)},\ldots,\nabla\boldsymbol{C}_{(1,h)},\nabla\boldsymbol{C}_{(2,1)},\ldots,\nabla\boldsymbol{C}_{(2,h)},\ldots,\nabla\boldsymbol{C}_{(m,h)},\nabla\boldsymbol{C}_{(1,1)},\ldots}_{nh+(m/2-1)2h+h}).$$

In summary, the bootstrapped ciphertext contains the following elements:

$$\boldsymbol{D}_{\text{boot}} = \underbrace{U_{\text{backW}}}_{onh} \;||\; \underbrace{U_{\text{backC}}}_{o(nh+(m/2-1)2h+h)} \;||\; \underbrace{P_{\text{back}}}_{onh} \;||\; \underbrace{W_{\text{back}}}_{o(nh+(m/2-1)2h+h)} \;||\; \underbrace{\nabla W_{\text{back}}}_{onh} \;||\; \underbrace{\nabla C_{\text{back}}}_{nh+(m/2-1)2h+h},$$

and the total number of slots used in the ciphertext must respect

$$3onh + (2o+1)(nh + (m/2-1)2h + h) \leq \mathcal{N}/2$$

for a ring degree $\mathcal{N}$. Therefore, a bootstrapped ciphertext can hold up to $n = \lfloor (N/(2h) - (2o+1)(m-1))/(5o+1) \rfloor$ samples. For example, given $N = 2^{15}$, $m = 38$, $h = 8$ and $o = 2$, the ciphertext holds 169 samples. This number is smaller than the number of samples that can be repacked in a single ciphertext before the bootstrapping, hence it sets an upper bound for the number of samples that can be trained in a single batch.

Backward Pass. The backward pass is computed using the ciphertext $\boldsymbol{D}_{\text{boot}}$. The different values contained in $\boldsymbol{D}_{\text{boot}}$ are accessed via rotations and ciphertext duplication. Masking is used only at the very end to minimize the use of levels. We start by computing the error of the dense layer formatted for the dense layer update ($\boldsymbol{E}_1$) and formatted for the convolution layer ($\boldsymbol{E}'_1$) at the same time:

$$(\boldsymbol{E}_1 || \boldsymbol{E}'_1) = \varphi'(\boldsymbol{U}_{\text{backW}} || \boldsymbol{U}_{\text{backC}}) \odot \big(\varphi(\boldsymbol{U}_{\text{backW}} || \boldsymbol{U}_{\text{backC}}) - (Y_{\text{backW}} || Y_{\text{backC}})\big),$$

with $Y_{\text{backW}} || Y_{\text{backC}}$, the plaintext labels, accordingly encoded and formatted. We then compute in parallel the updated weights of each sample of the dense layer and the partial error of the convolution layer by multiplying $\boldsymbol{E}_1 || \boldsymbol{E}'_1$ with $\boldsymbol{P}_{\text{back}} || \boldsymbol{W}_{\text{back}}$. Note that $\boldsymbol{P}_{\text{back}} || \boldsymbol{W}_{\text{back}}$ can be accessed and aligned with a rotation on $\boldsymbol{D}_{\text{boot}}$.

$$\nabla \boldsymbol{W} || \boldsymbol{E}_0 = (\boldsymbol{E}_1 || \boldsymbol{E}'_1) \odot (\boldsymbol{P}_{\text{back}} || \boldsymbol{W}_{\text{back}}).$$

$\nabla \boldsymbol{W}$ is clustered by samples, hence we add a summation across the n samples to obtain the updated dense layer weights of the batch. The output contains only a single copy, column-encoded, of $\nabla \boldsymbol{W}$, and of size oh. An additional step first adds, then masks and extracts, each column of the result and replicates them n times to expand its size back to onh and to match the original encoding format of W (this masking also removes all the unwanted by-product values). $\nabla \boldsymbol{W}_{\text{back}}$ is added to the result to get the final updated weights of the dense layer.

We finalize the computation of E_0 by a summation across the labels, reducing its size to $nh + (m/2-1)2h + h$). E_0 is already formatted to be multiplied with the plaintext transposed sample matrix $\eta \cdot L^T$ (pre-pooled and multiplied by η). This step is the same as the convolution layer matrix multiplication:

$$\nabla \boldsymbol{C} = \eta \cdot L^T \times \boldsymbol{E}_0.$$

The result is of size nh, with no by-product garbage slots due to the plaintext multiplication, but it needs to be extended to a size of $nh + (m/2-1)2h + h$ to comply with the formatting of $\boldsymbol{C}$. This is done by replicating the nh slots until it reaches at least this amount of slots and by masking the overflow of slots. Similarly, $\nabla \boldsymbol{C}_{\mathsf{back}}$ is added to $\nabla \boldsymbol{C}$, and the result is stored as the newly updated weights for the next batch of samples.

We summarize the given steps in Algorithm 8. We initialize the weight matrices with the levels that are compatible with the algorithm. That is, the final level of the ciphertexts after weight updates are used for initialization. Note that the algorithm describes the local computations. Then, the parties collectively aggregate and update the global model, which includes the additional step of taking the mean of $\nabla \boldsymbol{W}$ and $\nabla \boldsymbol{C}$ across all the parties.

Algorithm 8 The *local computation* algorithm for PRICELL. The superscript of encrypted values (e.g., y for $\boldsymbol{C}^y$) denotes the current ciphertext level. Encryption, encoding, and detailed steps of the repacking during the bootstrapping are omitted for clarity. The value csize represents $nh + (\lceil m/2 \rceil - 1)2h + h$. We give the function definitions in Section 1.4.

Inputs: X and Y set of samples and labels, learning rate η, momentum μ, batch size n, number of iterations d, number of features m, number of filters h, number of labels o, $maskW$ a vector containing ones in the first onh slots, $maskW_i$ a set of o vectors containing ones in the slots inh to $(i+1)nh$ slots for $0 < i < o$, and $maskC$ a vector containing ones in the first csize slots.

Outputs: The encrypted weights $\boldsymbol{C}$ and $\boldsymbol{W}$.

$\boldsymbol{C}^4 \leftarrow \mathsf{Init}(m,h), \boldsymbol{W}^3 \leftarrow \mathsf{Init}(h,o)$ ▷ Initialize convolution and dense weights

$\boldsymbol{\nabla W}^5_{\mathsf{prev}}, \boldsymbol{\nabla C}^4_{\mathsf{prev}} \leftarrow 0$ ▷ Initialize previous updated weights

for $i = 0; i < d; i = i + 1$ **do**

Batch Selection

$X_{\mathsf{batch}} \leftarrow \mathsf{Select}_n(X)$ ▷ Select a batch of random samples

$Y_{\mathsf{batch}} \leftarrow \mathsf{Select}_n(Y)$ ▷ Select the corresponding labels

$L_{\mathsf{pool}} \leftarrow \mathsf{Pre\text{-}pooling}(X_{\mathsf{batch}})$ ▷ Apply the pre-pooling to the batch

Forward Pass

$\boldsymbol{C}^4_{\mathsf{tmp}} \leftarrow \boldsymbol{C}^4 + \mathsf{RotL}_h(\mathsf{MultImag}(\boldsymbol{C}^4))$ ▷ Preprocessing for complex matrix multiplication

$\boldsymbol{P}^3 \leftarrow \sum_{i=0}^{\lceil m/2 \rceil - 1} L^{\mathsf{diag}[i]}_{\mathsf{pool}} \odot \mathsf{RotL}_{2hi}(\boldsymbol{C}^4_{\mathsf{tmp}})$ ▷ Convolution

$\boldsymbol{P}^3 \leftarrow \mathsf{Replicate}_{nh,o}(\boldsymbol{P}^3)$ ▷ Replicate the result for each label

$\boldsymbol{U}^2 \leftarrow \mathsf{InnerSum}_{1,h}(\boldsymbol{P}^3 \odot \boldsymbol{W}^3)$ ▷ Dense layer

Bootstrapping

$\boldsymbol{D}^2_{\mathsf{repack}} = \boldsymbol{U}^2 + \mathsf{RotL}_{-nho}(\boldsymbol{P}^3) + \mathsf{RotL}_{-2nho}(\boldsymbol{W}^3) + \mathsf{RotL}_{-3nho}(\boldsymbol{\nabla W}^5_{\mathsf{prev}}) + \mathsf{RotL}_{-4nho}(\boldsymbol{\nabla C}^4_{\mathsf{prev}})$ ▷ Pack all necessary values in a single ciphertext

$\boldsymbol{D}^9_{\mathsf{boot}} \leftarrow \mathsf{DBootstrap}_{\eta,\mu}(\boldsymbol{D}^2_{\mathsf{repack}})$ ▷ Refresh the ciphertext and formatting for the backward pass

Backward Pass

$\boldsymbol{U1}^7 \leftarrow \varphi(\boldsymbol{D}^9_{\mathsf{boot}})$ ▷ Activation

$\boldsymbol{U2}^7 \leftarrow \varphi'(\boldsymbol{D}^9_{\mathsf{boot}})$ ▷ Activation derivative

$\boldsymbol{E1}^6 \leftarrow \boldsymbol{U2}^7 \odot (\boldsymbol{U1}^7 - Y_{\mathsf{batch}})$ ▷ Dense layer error

$\boldsymbol{P}^9 \leftarrow \mathsf{RotL}_{nho+o\cdot\mathsf{csize}}(\boldsymbol{D}^9_{\mathsf{boot}})$ ▷ Access pooling result and dense layer weights

$\boldsymbol{\nabla W}^5 \leftarrow \boldsymbol{P}^9 \odot \boldsymbol{E1}^6$ ▷ Dense layer updated weights and convolution layer error

$\boldsymbol{E0}^5 \leftarrow \mathsf{RotL}_{nho}(\boldsymbol{\nabla W}^5)$ ▷ Access convolution layer error

$\boldsymbol{\nabla W}^5 \leftarrow \mathsf{InnerSum}_{oh,n}(\boldsymbol{\nabla W}^5)$ ▷ Finish updated weights with summation across the samples

$\boldsymbol{E0}^5 \leftarrow \mathsf{InnerSum}_{\mathsf{csize},o}(\boldsymbol{E0}^5)$ ▷ Finish E1 with summation across the labels

$\boldsymbol{\nabla C}^4 \leftarrow \sum_{i=0}^{\lceil n/2 \rceil - 1} (0.5 \cdot L^{T,\mathsf{diag}[i]}_{\mathsf{pool}}) \odot \mathsf{RotL}_{2mi}(\boldsymbol{E0}^5)$ ▷ Multiply with the transposed samples

$\boldsymbol{\nabla C}^4 \leftarrow \boldsymbol{\nabla C}^4 + \mathsf{Conjugate}(\boldsymbol{\nabla C}^4)$ ▷ Clean imaginary part

$\boldsymbol{\nabla C}^4 \leftarrow \mathsf{Replicate}_{mh,\lceil \mathsf{csize}/mh \rceil}(\boldsymbol{\nabla C}^4)$ ▷ Format updated weights for convolution layer

$\boldsymbol{\nabla W}^5 \leftarrow \mathsf{Replicate}_{h,n}(\sum_{i=0}^{o-1} \mathsf{RotL}_{-inh}(maskW_i \odot \boldsymbol{\nabla W}^5))$ ▷ Format weights for dense layer

$\boldsymbol{\nabla W}^8_{\mathsf{prev}} \leftarrow maskW \odot \mathsf{RotL}_{2nho+2o\cdot\mathsf{csize}}(\boldsymbol{D}^9_{\mathsf{boot}})$ ▷ Access and extract the previous weights

$\boldsymbol{\nabla C}^8_{\mathsf{prev}} \leftarrow maskC \odot \mathsf{RotL}_{3nho+2o\cdot\mathsf{csize}}(\boldsymbol{D}^9_{\mathsf{boot}})$ ▷ Access and extract the previous weights

Weights Update

$\boldsymbol{\nabla W}^5 \leftarrow \boldsymbol{\nabla W}^5 + \boldsymbol{\nabla W}^8_{\mathsf{prev}}$ ▷ Add previous updated weights with momentum

$\boldsymbol{\nabla C}^4 \leftarrow \boldsymbol{\nabla C}^4 + \boldsymbol{\nabla C}^8_{\mathsf{prev}}$ ▷ Add previous updated weights with momentum

$\boldsymbol{C}^4 \leftarrow \boldsymbol{C}^4 - \boldsymbol{\nabla C}^4$ ▷ Update the weights

$\boldsymbol{W}^3 \leftarrow \boldsymbol{W}^3 - \boldsymbol{\nabla W}^5$ ▷ Update the weights

$\boldsymbol{\nabla C}^4_{\mathsf{prev}} \leftarrow \boldsymbol{\nabla C}^4$ ▷ Store the new updated weights

$\boldsymbol{\nabla W}^5_{\mathsf{prev}} \leftarrow \boldsymbol{\nabla W}^5$ ▷ Store the new updated weights

end for

6.4 Experimental Evaluation

In this section, we describe the datasets that are used in the evaluation. Then, we present the experimental setting and lay out our experimental findings.

6.4.1 Datasets

We detail the features of the three used datasets:
Non-inflammatory neurological disease (NIND), relapsing–remitting multiple sclerosis (RRMS). We rely on a large cohort of peripheral blood mononuclear cells (PBMCs), including 29 healthy donors (HD), 31 NIND, and 31 RRMS donors [106]. The dataset comprises samples with a varying number of cells for each donor and 35 markers for each cell. We use this dataset for two classification tasks: (i) HD vs. NIND, and (ii) HD vs. RRMS, as shown in Figure 6.3 and 6.4. For both NIND and RRMS experiments and in all experimental settings, we use 48 donors (24 HD, 24 NIND / RRMS) for training and 12 donors (5 HD, 7 NIND / RRMS) for testing.
Cytomegalovirus Infection (CMV) classification. We use a mass cytometry dataset [131] for the classification of cytomegalovirus infection (CMV). This dataset comprises samples from 20 donors with a varying number of cells for each donor and mass cytometry measurements of 37 markers for each cell and has 11 CMV- and 9 CMV+ labels. We use 14 donors for training and 6 donors as a test set in all experimental settings.
Acute Myeloid Leukaemia (AML). We rely on the mass cytometry dataset from Levine et al. [175] for the 3-class classification problem for healthy, cytogenetically normal (CN), and core-binding factor translocation (CBF). For each cell, the dataset includes mass cytometry measurements of 16 markers. As in the original work [32], we use the AML samples with at least 10% CD34+ blast cells with the availability of additional cytogenetic information. The final training dataset comprises 3 healthy bone marrows (BM1, BM2, BM3), 2 CN samples (SJ10,SJ12), and 2 CBF samples (SJ1, SJ2). The test set in all experimental settings comprises 2 healthy bone marrows (BM4, BM5), 1 CN (SJ13), and 3 CBF (SJ3, SJ4, SJ5) samples.

The individual donors in all aforementioned training sets are then evenly distributed among N parties for PRICELL collective training. To construct our baselines and to make a fair comparison with the baseline, we use the same data preprocessing for all experiments per setting (centralized CellCnn, Local, or PRICELL). We give the details of the data preprocessing and parameter selection, in Appendix E.1.

6.4.2 Experimental Settings

We implemented our solution in Go [7] by using the open-source lattice-based cryptography Lattigo [200]. We use the implementation of CellCnn [32] to preprocess the data and to construct baselines. We use Onet [5] to build a decentralized system and Mininet [201] to

evaluate our system in a virtual network with an average network delay of 0.17 ms and 1 Gbps bandwidth on 10 Linux servers with Intel Xeon E5-2680 v3 CPUs running at 2.5 GHz with 24 threads on 12 cores and 256 GB RAM. The parties communicate over TCP with secure channels (TLS). We choose our security parameters that achieve at least 128-bits security [27].

6.4.3 Empirical Results

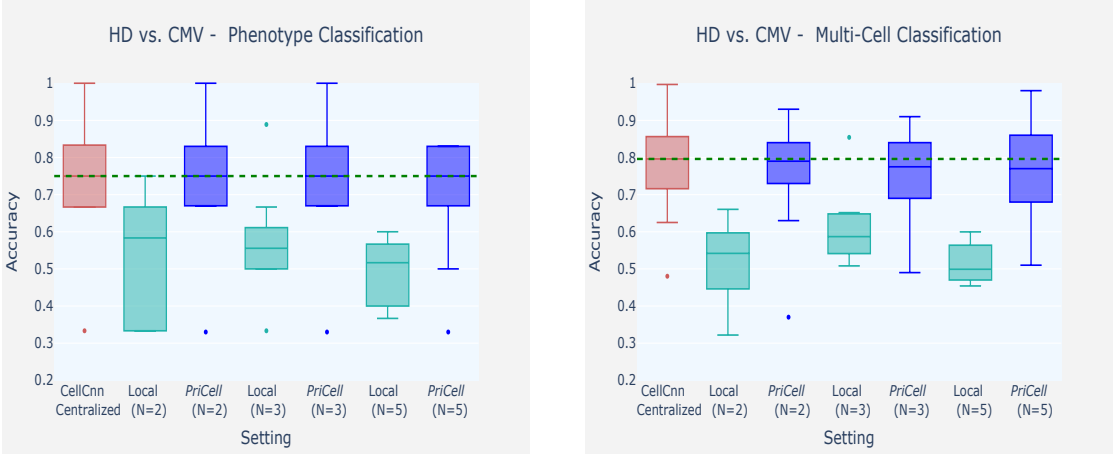
We evaluate our proposed solution in terms of model accuracy, runtime performance, scalability with the number of parties, number of data samples, number of features, and communication overhead. And we provide a comparison with POSEIDON in this section. We give details on the machine learning hyperparameters and security parameters used for our evaluation in the Appendix E.1.

Model Accuracy. To assess our solution in terms of accuracy, we use the same datasets used in two peer-reviewed biomedical studies [32, 106]. We rely on three aforementioned datasets to perform non-inflammatory neurological disease (NIND), relapsing–remitting multiple sclerosis (RRMS), Cytomegalovirus Infection (CMV), and Acute Myeloid Leukaemia (AML) classification. **Our aim is to show that PRICELL achieves a classification performance on par with the centralized non-private baseline.**

As the original studies rely on centralized datasets, we evenly distribute the individual donors in the respective dataset over N parties. We give the classification performance on these datasets in Figure 6.2, 6.3, and 6.4. The x-axis shows different training approaches: (i) the data is centralized and the original CellCnn approach [32] is used for the training and classification to construct a baseline, (ii) each party trains a model only with its local data (Local), without collaborating with other parties, and (iii) our solution for privacy-preserving collaboration between parties is used (PRICELL). For the Local training (ii), we average the test accuracy achieved by individual parties.

In our experiments, random multi-cell inputs that are used for training are drawn with replacement from the original training samples. Drawing multi-cell inputs can be done in two ways: using the bag of all cells per class or individually drawing them from each patient. We report the classification performance by using two test datasets: One set is generated by using multi-cell inputs with $c = 100 - 200$ cells drawn from all patients in the test set to increase the size of the test set for multi-cell classification; and the second set is generated by drawing $1000 - 10000$ cells from each *donor* separately for phenotype prediction. We give more details about the setting and hyperparameters for each experiment in the Appendix E.1.

For CMV classification, we generate the training data by drawing random cell subsets from the cell bags *per phenotype*. For NIND and RRMS classification, we observe that drawing multi-cells per phenotype varies the accuracy between runs and that the median accuracy over 10 runs increases when distributing the initial dataset among $N = 6$ parties (see Figure 6.3a, 6.3b and 6.4a, 6.4b). This suggests that separately drawing multi-cell inputs from each individual performs better for this task, as corroborated by the results obtained with drawing 2000



(a) Phenotype Classification

(b) Multi-Cell Classification

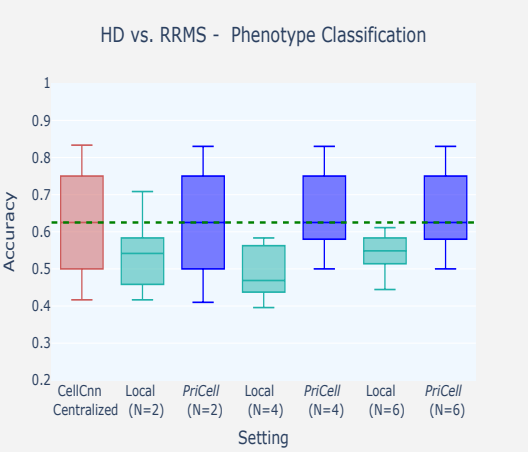
Figure 6.2: *Accuracy Boxplots when classifying healthy donor (HD) vs. cytomegalovirus infection (CMV) for training multi-cells drawn from the bag of all cells per class. Experiments are repeated 10 times with different train and test set splits, the vertical dashed line illustrates the median for the baseline (CellCnn) and the dots represent the outliers. Classification accuracy is reported for two datasets:* ***(a)*** *phenotype classification of 6 patients and* ***(b)*** *multi-cell input classification on 4000 samples.*

cells from *each patient* with replacement (see Figure 6.3c, 6.3d and 6.4c, 6.4d). Finally, in Appendix E.2, Table E.1, we report the median accuracy, precision, recall, and F-score of 10 runs (with different train and test set splits) on patient-based sub-sampling for NIND and RRMS, and phenotype-based sub-sampling for CMV.

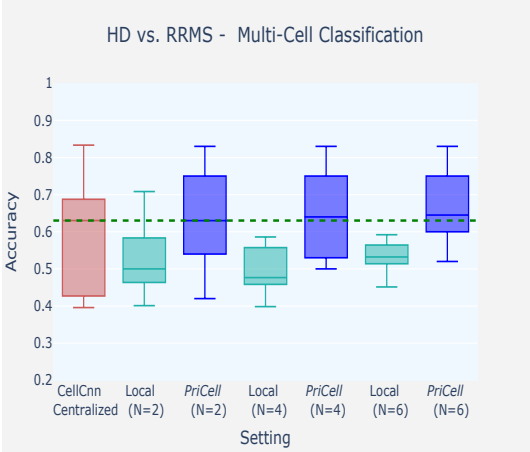
To construct a realistic overall distribution, we limit the number of parties to be lower than the number of donors in the dataset. We observe that, given a sufficient number of samples per party, our distributed secure-solution achieves classification performance comparable to the original work, where the data is centralized and the training is done without privacy-protection. In the experiments on CMV, for example, the median accuracy achieved by PRICELL is exactly the same as the centralized baseline for phenotype classification and very close (at most 2% gap) for multi-cell classification. Analogous results are obtained for the other experiments: Our privacy-preserving distributed solution achieves almost the same median accuracy with the baseline in RRMS and NIND with patient-based sub-sampling, where the datasets are sufficiently large to be distributed among up to 6 parties.

Lastly, we provide the classification performance on AML in Appendix E.2, Table E.1. As the dataset is relatively small, emulating a distributed setting with more than two parties was not feasible for this task and, as the accuracy does not vary in between different train-test splits, we do not provide the boxplots on the accuracy. However, we observe that with two parties in PRICELL training, the accuracy remains exactly same as the centralized baseline for AML classification.

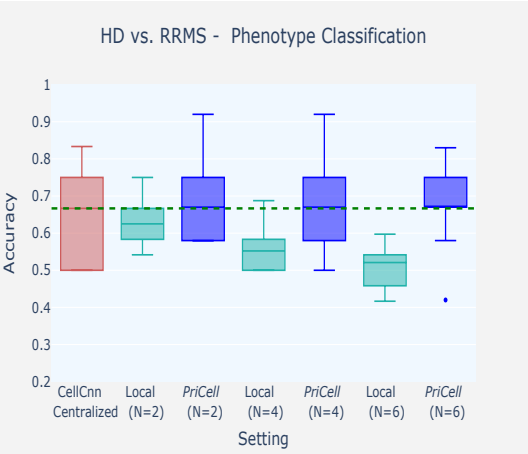
Most importantly, our evaluation shows that there is always a significant gain in classification performance when switching from local training to privacy-preserving collaboration. The number of donors that each institution has is insufficient for individually training a ro-



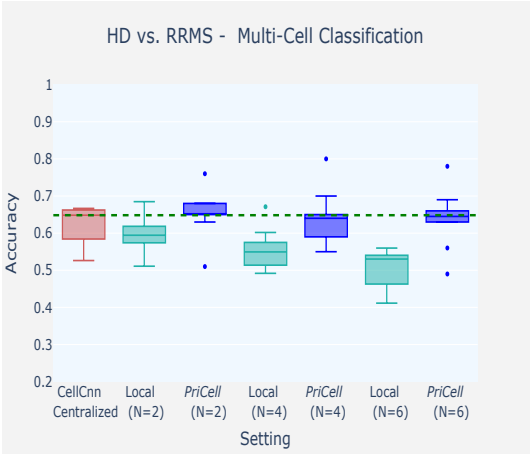
(a) Phenotype Classification with bag of all cells



(b) Multi-Cell Classification with bag of all cells

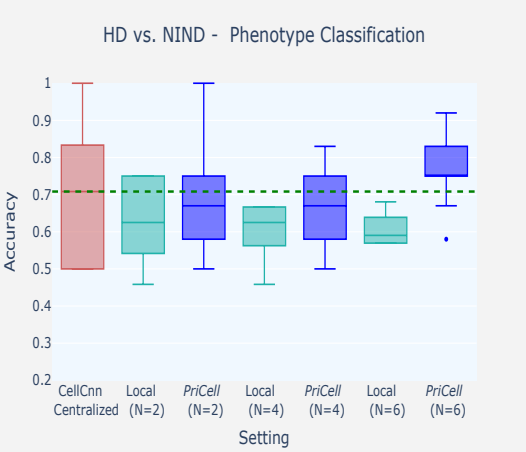


(c) Phenotype Classification with cells drawn from each patient separately

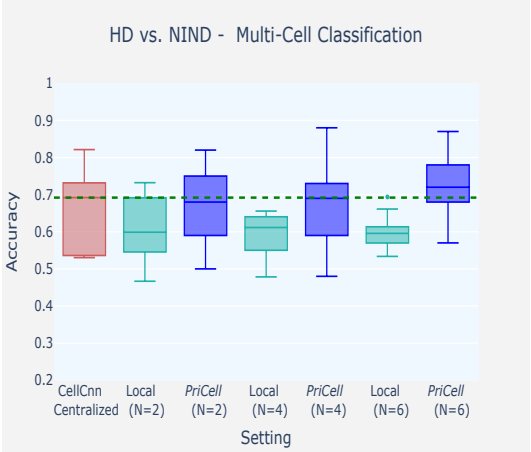


(d) Multi-Cell Classification with cells drawn from each patient separately

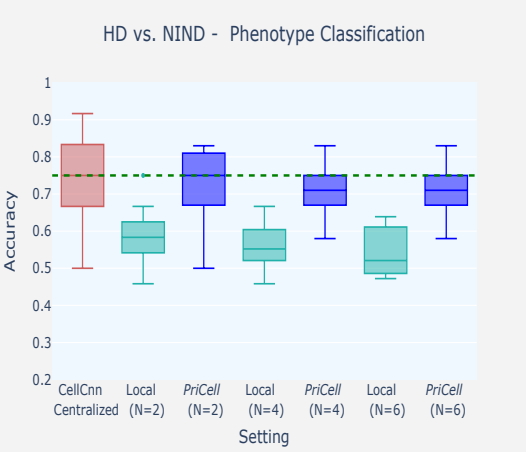
Figure 6.3: *Accuracy boxplots when classifying healthy donor (HD) vs. relapsing–remitting multiple sclerosis (RRMS), for training multi-cells drawn from the bag of all cells per class* ***(a-b)*** *and drawn from each patient separately* ***(c-d)****. Experiments are repeated 10 times with different train and test set splits, the vertical dashed line illustrates the median for the baseline (CellCnn) and the dots represent the outliers. Classification accuracy is reported for two datasets: multi-cell input classification on 96 samples, and phenotype classification of 12 patients.*



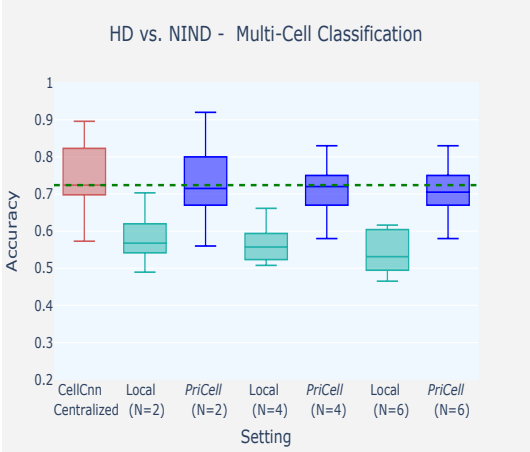
(a) Phenotype Classification with bag of all cells



(b) Multi-Cell Classification with bag of all cells



(c) Phenotype Classification with cells drawn from each patient separately



(d) Multi-Cell Classification with cells drawn from each patient separately

Figure 6.4: *Accuracy Boxplots when classifying healthy donor (HD) vs. non-inflammatory neurological disease (NIND), for training multi-cells drawn from the bag of all cells per class* ***(a-b)*** *and drawn from each patient separately* ***(c-d)****. Experiments are repeated 10 times with different train and test set splits, the vertical dashed line illustrates the median for the baseline (CellCnn) and the dots represent the outliers. Classification accuracy is reported for two datasets: multi-cell input classification on 96 samples and phenotype classification of 12 patients.*

bust model. In all experimental settings, for a fixed number of N, PRICELL achieves better performance than the local training while ensuring the confidentiality of the local data.

Runtime. We report in Table 6.1, the execution times for the training and prediction with $N = 10$ parties and a ring degree $\mathcal{N} = 2^{15}$. To be able to compare the runtimes at a larger scale, we use synthetically generated data for this set of experiments and vary the number of features (m). We generate a data set of 1000 samples per party with $c = 200$ cells per sample. We use $h = 8$ filters, a local batch size of $n = 100$, and 20 global epochs for training. We report the execution time of the setup phase, of the local computations, and of its communication. We include the execution time of distributed bootstrapping (see Section 6.3 for details) as part of the communication time, which takes 1.2s per iteration and 122s over 20 epochs. Hence, the communication column for training comprises the time to perform all communication between parties throughout the training, distributed bootstrapping, and the model update.

We observe that PRICELL trains, in less than 20 minutes, a CellCnn model on a training set of 200 cells per sample, 1000 samples per party, and 32 features across 10 parties, including the setup phase and communication. The training time, when the number of features varies, remains 20-25 minutes, which is the result of our efficient use of the SIMD operations provided by the cryptosystem; this is further discussed in the scalability analysis.

We also report in Table 6.1, the execution times of an oblivious prediction when both the model and the data are encrypted (Phase 5.1 of Figure 3.2). We recall that the collective key-switching operation enables us to change the encryption key of a ciphertext from the parties' collective key to the querier's key. The maximum number n of samples that can be batched together for a given ring degree $\mathcal{N}$, number of labels o, and number of features m, is $(\mathcal{N}/2)/(m \cdot o)$ (we also need $m/2$ ciphertexts to batch those samples, see Appendix 6.3 for more details). Hence, in our case the maximum prediction batch size for $\mathcal{N} = 2^{15}$ and $o = 2$ is $n = 2^{13}/m$.

We observe that the local computation for the prediction increases linearly with m, and is linked to the cost of the dominant operation, the convolution, which is, unlike training, carried out between two encrypted matrices (see Appendix 6.3). The communication required for prediction includes $m/2$ ciphertexts sent by the querier and one ciphertext (prediction result) sent back by the server. Hence, the communication time also increases linearly with m. Lastly, the time for the collective key-switch remains constant, as it is performed once at the end of the prediction protocol on only one ciphertext.

Scalability Analysis. Figure 6.5 shows the scalability of PRICELL with the number of parties, the global number of rows (samples), number of features (markers), and the number of filters for one global training epoch that is to process once all the data of all parties. Unless otherwise stated, we use $c = 200$ cells per sample, a local batch size of $n = 100$, $m = 38$ features, and $h = 8$ filters, for all settings. We first report the runtime with an increasing number of parties (N) in Figures 6.5a and 6.5b when the global number of data samples is fixed to $s = 18000$ and when the number of samples per party is fixed to 500, respectively. As the parties

	Training Execution Time [sec]			Prediction Execution Time [sec]		
m	Setup	Local Computation	Communication	Local computation Querier + Server	Communication	Collective Key-Switch
8	17.8	753.4	370.1	0.2 + 0.1	0.3	0.3
16	18.1	778.7	387.0	0.3 + 0.2	0.6	0.3
32	19.3	836.1	393.7	0.3 + 0.4	1.0	0.3
64	21.9	951.1	373.1	0.6 + 0.6	2.2	0.3
128	24.5	1135.9	374.8	1.6 + 1.5	4.2	0.3

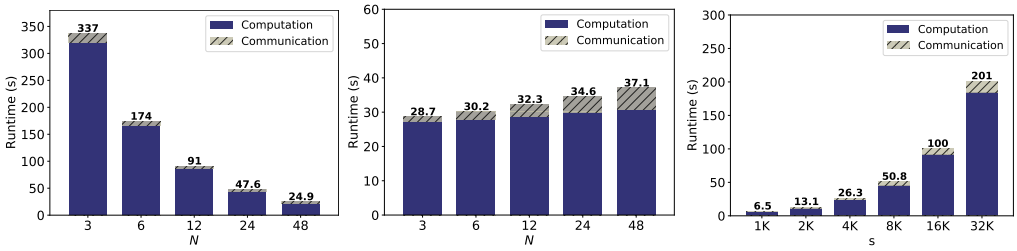
Table 6.1: PRICELL*'s execution times for training and prediction with a varying number of features (m), 10 parties, and ring degree* $\mathcal{N} = 2^{15}$ *(*2^{14} *ciphertext slots). The computation is single-threaded in a virtual network with an average network delay of 0.17 ms and 1 Gbps bandwidth on 10 Linux servers with Intel Xeon E5-2680 v3 CPUs running at 2.5 GHz with 24 threads and 12 cores and 256 GB RAM.*

perform local computations in parallel, PRICELL's runtime decreases with increasing N when s is fixed (Figure 6.5a). When the number of data samples is constant per party, PRICELL's computation time remains almost constant and only the communication overhead increases when increasing N (Figure 6.5b).

We further analyze PRICELL's scalability for $N = 10$ when varying the number of global samples (s), the number of features (m), and the number of filters (h). In Figure 6.5c, we show that PRICELL scales linearly when increasing the number of global samples with $N = 10$. Increasing the number of features and filters has almost no effect on PRICELL's runtime, due to our efficient packing strategy that enables SIMD operations through features and filters. However, we note that the increase in $h = 64$ in Figure 6.5e is due to increasing the cryptosystem parameter $\mathcal{N}$ to have a sufficient number of slots to still rely on our one-cipher packing strategy. The increase in runtime is still linear with respect to $\mathcal{N}$ and, as expected, the use of larger ciphertexts also produces a slight increase in the communication.

Comparison with POSEIDON

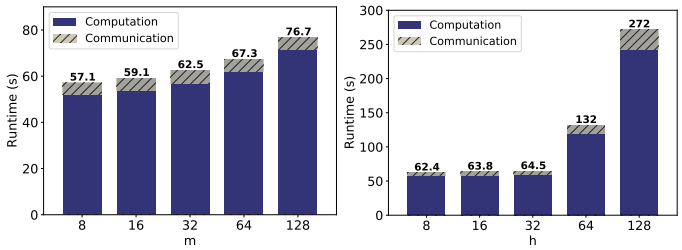
HE-based solutions for privacy-preserving analytics in distributed medical settings [103, 230] allow for functionalities (e.g., basic statistics, counting, or linear regression) different than those PRICELL enables. Due to the fact that the underlying system, the threat model, and the enabled functionalities of all these solutions are different from PRICELL, a quantitative comparison with these works is a challenging task. We build on the system and threat model proposed in Chapter 3.1 and thus make a quantitative comparison with POSEIDON (see Chapter 4). PRICELL improves upon the state-of-the-art solution, POSEIDON, by at least one order of magnitude when training CellCnn with the same number of epochs and filters. This is due to PRICELL's design for optimizing the use of SIMD operations, with a packing strategy that enables encrypting all samples of a batch in a single ciphertext whereas, POSEIDON packs the samples within a batch in different ciphertexts. For a local batch size of 1, 8 filters, 38 features, and 200 cells per sample, PRICELL's local computation time is 1.7s; whereas, POSEIDON's is 15.4s. Increasing the batch size to 100 results in a 100x slower local execution for POSEIDON, whereas it remains constant for PRICELL, as all samples are packed in one ciphertext. In



(a) Increasing number of parties (N) when the number of global data samples (s) is fixed to 18000.

(b) Increasing number of parties (N), each having 500 samples.

(c) Increasing number of data samples (s) when $N = 10$.



(d) Increasing number of features (m) when $N = 10$.

(e) Increasing number of filters (h) when $N = 10$.

Figure 6.5: PRICELL*'s training execution time and communication overhead for one training epoch with increasing number of parties, data samples, features, and filters. The computation is single-threaded in a virtual network with an average network delay of 0.17 ms and 1 Gbps bandwidth on 10 Linux servers with Intel Xeon E5-2680 v3 CPUs running at 2.5 GHz with 24 threads on 12 cores and 256 GB RAM.*

summary, increasing the batch size or the number of filters yields a linear increase in the advantage of our solution, in terms of local computation time.

Downstream Analysis. The training in the original CellCnn study aims at detecting rare disease-associated cell subsets via further analysis [32]. Assuming the end model is decrypted upon pre-agreement to conduct these analyses, we further investigate how the changes that we introduce in the CellCnn architecture (see Local Neural Network Operations, Section 6.2.2) affect the detection capability. To be able to make a comparison with the original study in terms of detection capability, we introduce these changes in the original implementation of CellCnn, simulate our encryption, and evaluate the impact in the subsequent analyses. We report our results and about how our changes to the circuit and training affect the detection capability on rare CMV infection, in the Appendix E.3.

6.5 Conclusion

In this chapter, we present PRICELL, a system that enables privacy-preserving federated neural network learning for healthcare institutions, in the framework of an increasingly relevant single-cell analysis, by relying on multiparty homomorphic encryption (MHE). To the best

of our knowledge, PRICELL is the first solution to enable the training of convolutional neural networks with N parties under encryption on single-cell data.

In this chapter, we demonstrate the flexibility of PRICELL with the different learning parameters (e.g., batch size, number of features, number of filters), different real-world datasets, and a varying number of parties. Our empirical evaluation shows that PRICELL is able to efficiently train a collective neural network with a large number of parties while protecting the model and the data through homomorphic encryption. We also show that PRICELL's computation and communication overhead remains either constant or scales linearly with the number of parties and with the model parameters.

Furthermore, we show that PRICELL achieves classification accuracy comparable to the centralized and non-encrypted training. Our evaluation demonstrates a substantial accuracy gain by collaboration between the parties when compared to locally training with their data only.

As data sharing in the healthcare domain is usually prevented due to the sensitive nature of data, and due to privacy regulations such as HIPAA [3] or GDPR [8], PRICELL brings unprecedented value for the healthcare domain, exemplified in this chapter for single-cell analysis, where the data is scarce and sparse. These benefits are extensible to federated healthcare scenarios that rely on machine learning, and constitutes an important landmark for real-world applications of collaborative training between healthcare institutions while preserving privacy.

7 Conclusion

In this thesis, we have proposed novel systems for addressing the privacy-preserving training of neural networks in cross-silo federated learning settings. Our systems also enable the prediction-as-a-service for a querier in a privacy-preserving way. The proposed systems enable the training pipeline to executed under encryption by relying on multiparty homomorphic encryption (MHE). As such, they provide security against inference attacks to federated learning [199, 212, 128, 283, 309]. Furthermore, our solutions are quantum-resistant and ensure at the same time the confidentiality of the model, the training data, and the evaluation data under a passive adversary threat model.

We have shown that our high-level solution that enables encrypted training scale linearly with the number of parties in the federated learning system and do not degrade the utility of the data. We have further investigated the model performance and shown that all proposed systems achieve a performance that is on par with centralized or decentralized non-private approaches.

In Chapter 4, we have proposed POSEIDON: a system that enables the training and evaluation of multilayer perceptrons and convolutional neural networks in a cross-silo federated learning setting with N data providers. By building a modular system, we have shown that several blocks can be changed or added to our system to better tune the trade-off between the model performance and the runtime performance or to improve the system to enable the training of different network structures (e.g., MLP or CNNs). We have efficiently performed the training with the benchmark datasets and showcased the applicability of our solutions. We believe that our system will be a key enabler for the privacy-preserving predictive tasks on the cross-silo settings by enabling the data providers to have control over their data.

In Chapter 5, by building on the solution proposed in Chapter 4, we have proposed RHODE: a system that tackles the training and evaluation of recurrent neural networks in the same federated learning setting. For this, we have addressed the challenges of training recurrent neural networks under encryption by our technical contributions. RHODE is the first system that enables predictive time-series analysis under encryption in the federated learning setting.

By its modular design and building blocks, RHODE serves as a guideline for building more complex recurrent neural network models.

In Chapter 6, we have demonstrated a real-life medical application of a privacy-preserving federated learning by leveraging the systems that we built. We have introduced further optimizations to the system to enable the training with single-cell data. We have demonstrated the flexibility of our solution in terms of several network parameters and different datasets. Our solution brings a remarkable value for the medical applications where the data is scarce and sparse.

We believe that our contributions in this thesis will promote the capabilities of homomorphic encryption within the context of collaborative learning with multiple data holders in a privacy-preserving way. Data sharing between multiple institutions is inherently difficult and incorporates legislation and ethics committees. According to European General Data Protection Regulation [8], the legal status of homomorphically encrypted data remains unclear and might depend on the argument. Indeed, several principles have to be examined before homomorphically encrypted data can be qualified as anonymous, thus this qualification is heavily dependent on the setting and the adversarial model. Within the context of the 'relative approach' (which states that the legal status of the data must be examined through the eyes of each processor), a large part of the legal doctrine argues that encrypted data can be qualified as anonymous to a processor who cannot have access to the decryption key [117, 96]. Thus, one can argue that MHE provides full anonymity as the key is split among all the participating entities and no single entity is able to decrypt the data by itself. Regardless of the legal status of the data, this thesis shows that homomorphic encryption is mature enough to be practical in a wide range of applications where multiple parties must collaborate. This alone can be used as a tool to provide technical measures and ensure data protection by design and by default (art. 25 GDPR [2]), enabling collaboration that would have been difficult, if not impossible, previously. One example is computing a function on data that is split between multiple jurisdictions and that would have previously required a cross-border transfer to achieve the same result. Consequently, we believe that our solutions that rely on MHE will be the key facilitator for more secure and easier collaborations for machine learning models.

Limitations and Future Work.

In terms of limitations, our systems protect the confidentiality through encryption and, by design, do not decrypt any value throughout the learning for end-to-end protection. Hence, monitoring the learning process, e.g., the training/validation errors, is not possible as these values will be under encryption. Yet, by allowing a small amount of leakage through the validation losses, we can ensure a collective decryption of the loss that is calculated in several pre-defined iterations by all the parties or part of the parties, depending on a pre-agreement. Quantifying the leakage through validation-set losses is an interesting future direction.

The systems that we have proposed in this thesis preserve the confidentiality under a passive adversary threat model. Future work involves extensions to other scenarios with active adver-

saries and, to enable the practical execution with active attacks, further optimizations to the learning process. We discuss an extension to active adversaries in Appendix B. Note that this extension is not trivial and comes at the cost of a substantial increase in the computational complexity.

In this work, we assume that all data holders are available (online) during the training and prediction workflows. As we rely on a cross-silo federated learning setting, this assumption is realistic. However, it is also prohibitive in terms of failures of any party as the collective cryptographic operations, e.g., $\mathsf{DBootstrap}(\cdot)$ or $\mathsf{DKeySwitch}(\cdot)$, require collaboration among all the data holders. Therefore, to support asynchronous federated learning (driven by a time-threshold or by relying on a subset of online data holders), our systems can be deployed with a threshold multi-party homomorphic encryption scheme that enables a subset of t-out-of-N data holders to perform the collective operations [208]. However, note that this cryptographic scheme would introduce a relaxation in the threat model (allowing collusions of up to $t-1$ data holders instead of $N-1$).

Lastly, we have shown that our systems provide a balanced trade-off between the computation and communication through our scalability analysis. For more complex neural network architectures, we note that homomorphic encryption requires further optimizations, e.g., hardware accelerations or GPU-compatible cryptographic functions, to enable practical training of these complex structures.

Appendix Part I

A Cryptography Glossary

Here, we provide a summary of the cryptography terms that are frequently used throughout this thesis in an alphabetical order.

Bootstrapping: the act of homomorphically refreshing a ciphertext to allow for further computations.

Ciphertext Slots: available space in a ciphertext to encrypt multiple values. In CKKS, the maximum number of *slots* that a ciphertext can have is half of the dimension of the ring degree, i.e. $\mathcal{N}/2$.

Collective Public Key: a public key generated with the interaction of a set of parties and that can be used by any party to encrypt. The decryption of a ciphertext that is encrypted with the collective key requires all parties to participate in the decryption protocol.

Collective Key Switching: an interactive re-encryption of a ciphertext to a different secret key.

Distributed Bootstrapping: bootstrapping that requires interaction between the parties but that is less computationally expensive than its non-interactive variant.

Evaluation Keys: special public keys used during the homomorphic evaluation of a circuit (e.g., homomorphic slot rotations).

Multiparty Homomorphic Encryption: a set of protocols that enable a group of parties to securely compute joint functions over their private inputs by using homomorphic encryption. Compared to LSSS-based (linear secret-sharing scheme) approaches, these protocols scale linearly with the number of parties and do not require private channels.

Packing: the act of encrypting multiple scalar values in a single ciphertext by using ciphertext slots.

Ring Degree ($\mathcal{N}$): the degree of the RLWE cyclotomic polynomial $X^{\mathcal{N}} + 1$.

Ring Learning With Errors (RLWE): a computational problem based on the difficulty of

solving linear equations that are perturbed by an error. The security of the cryptographic schemes used in this work is based on this problem.

Secret Key: a secret value used to decrypt a ciphertext and to generate the encryption key and evaluation keys.

Single Instruction, Multiple Data (SIMD) Operations: the ability to carry out operations in parallel on a batch of data that is encrypted in one ciphertext by using ciphertext slots, in the context of this work.

Slots Rotation: cyclic shift of the values encrypted in a ciphertext.

B Extensions

We introduce here several security, learning, and optimization extensions that can be integrated to POSEIDON, RHODE, or PRICELL.

B.1 Security Extensions

We provide several security extensions that can be integrated to POSEIDON as a future work.

Active Adversaries: Our systems preserve the privacy of the parties under a passive-adversary model with up to $N-1$ colluding parties, motivated by the cooperative federated learning scenario presented throughout this thesis. If applied to other different scenarios, our work could be extended to an active-adversarial setting by using standard verifiable computation techniques, e.g., resorting to zero-knowledge proofs where the active adversary is the aggregation server. This would, though, come at the cost of an increase in the computational complexity, that will be analyzed as future work.

Extending our solutions 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 [260] or zero-knowledge proofs [307], 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 [61] or proofs of authenticity on encrypted data [56], as well as their consistency, e.g., using cryptographic commitments [307]. While such techniques can reduce the risk of poisoning attacks in federated learning [268], their elimination remains an open research problem. Currently proposed defenses rely on gradient inspection [176, 303, 269, 290, 30] or operations that are not HE-friendly (min/max comparison, division under encryption) [154], thus, raising new challenges for encrypted FL pipelines. Time-coupled attacks, on the other hand, can be adressed via integration of the clipping (which slightly changes the packing and should be approximated) and client-side momentum proposed in [153].

Out-of-the-Scope Attacks: We briefly discuss here out-of-the-scope attacks and countermeasures that are not addressed in this thesis. By maintaining the intermediate values of the learning process and the final model weights under encryption, during the training process, we protect data and model confidentiality. As such, our systems protect against federated learning attacks [212, 199, 128, 309, 283]. Nonetheless, there exist inference attacks that target the outputs of the model's predictions, e.g., membership inference [254], model inversion [98], or model stealing [271]. Such attacks can be mitigated via complementary countermeasures that can be easily integrated to our systems: (i) limiting the number of prediction queries for the queriers, and (ii) adding noise to the prediction's output to achieve differential privacy guarantees. The choice of the differential privacy parameters in this setting remains an interesting open problem.

RHODE can further 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).

B.2 Learning Extensions

Early Stop. There are several techniques proposed for the early stopping of the training of a neural network. They also prevent over-fitting as described and evaluated by Prechelt [226]. These approaches are: (i) GL_α: stop when the generalization loss exceeds a certain threshold α, (ii) PQ_α: stop when the quotient of generalization loss and progress exceeds a certain threshold α, and (iii) UP_s: stop when the generalization error increased in s successive strips. The generalization error is estimated by the error on a validation set. We note that these methods can be seamlessly integrated into our systems by dividing each party's data into training and validation sets. Depending on the threshold and the method, the privacy-preserving implementation would require the homomorphic aggregation of the generalization error evaluated on each P_i's validation set and a collective decryption of the error, after a number of global iterations t. As the error is the averaged scalar value. The leakage from the loss remains negligible when there are sufficient validation samples.

Availability, Data Distribution, and Asynchronous Distributed Neural Networks. In this thesis, we rely on a multiparty cryptographic scheme that assumes that the parties are always available. We here note that our systems can support asynchronous distributed neural network training [83] without waiting for all parties to send the local gradients. As such, a time threshold could be used for updating the global model. However, we note that the collective cryptographic protocols (e.g., $\mathsf{DBootstrap}(\cdot)$ and $\mathsf{DBootstrapALT}(\cdot)$) require that all the parties be available. Changing POSEIDON's distributed bootstrapping with a centralized one that achieves a practical security level would require increasing the size of the ciphertexts and result in higher computation and communication overhead.

For the evaluation of POSEIDON, we evenly distribute the dataset across the parties; we con-

sider the effects of uneven distributions or the asynchronous gradient descent to the model accuracy — which are studied in the literature [179, 83, 281] — orthogonal to this work. However, a preliminary analysis with the MNIST dataset and the neural network structure defined in our evaluation (see Section 4.5) shows that asynchronous learning decreases the model accuracy between 1 and 4% when we assume that a server is down with a failure probability between 0.4 and 0.8, i.e., when there is between 40 and 80% chance of not receiving the local gradients from a server in a global iteration. Finally, we find that the uneven distribution of the MNIST dataset for $N = 10$ parties with one party holding 90% of the data results to a 6% decrease in the model accuracy.

Lastly, we note that the non-iid or heterogeneous distribution of the data in federated learning settings causes weight/parameter divergence [178, 305] or slow down the convergence due to client drift [155] that is introduced by the updates of each party (client) in the non-iid settings. Some mitigation techniques do not change the working principle of POSEIDON and can be integrated to the pipeline, e.g., by adjusting hyperparameters [178], gradient alignment algorithm [78], or creating and globally sharing a set of data with uniform distribution among the participants [305]. Other mitigation techniques such as the algorithm SCAFFOLD [155] is harder to integrate into our solutions as they require operations that are not HE-friendly such as comparison for checking the actual value of the gradient/local updates.

Other Neural Networks. In this work, we focus on the training of MLPs and CNNs with POSEIDON and present our packing scheme and cryptographic operations for these neural networks. We note that POSEIDON's packing protocols are tailored to MLPs and CNNs and might require adaptation for other neural network structures. We introduce the new packing protocols for RNNs with RHODE in Chapter 5 where we focus on simple RNN architectures with an input layer, hidden layers with RNN units, and an output layer of various structures, e.g., many-to-one, many-to-many, etc. We note that other RNN architectures, e.g., GRUs or LSTMs, comprise more complex pipelines that are harder to implement under (M)HE. Yet, 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 RNN architectures. These RNN architectures, e.g., GRUs or LSTMs, require changes only on the Local Computation Phase of RHODE.

As an example, we summarize the protocol for the forward and backward pass of a GRU architecture in Algorithm 9 for a party P_i holding its own input data X_i. Such an RNN has the same workflow as a conventional 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 timesteps, a Reset Gate (Z-Gate) that decides how much information to forget, and a Candidate activation (N-Gate) which is similar to the hidden state of an RNN unit. Other GRU variants modify these gates by computing only the bias or by excluding them [124, 86]. Both Reset and Update gates include an activation function that is typically a Sigmoid, $\varrho = \frac{1}{1+e^{-x}}$, and the hidden state includes an activation function that is typically Tanh, $\varphi = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$.

Given the above gate rules, the pipeline for the forward pass of a GRU is similar to a conventional RNN and the backward pass follows the BBPT algorithm with the chain rule for the calculation of the derivatives and the loss (see Algorithm 9). In more detail, the forward pass (Lines 2-10) comprises matrix multiplications, additions, and activation functions, thus, the packing scheme of RHODE, the matrix multiplication and the approximated building blocks can directly be applied. Similarly, the backward pass comprises the same operations as the forward pass but additionally requires multiple executions of the transpose operation, which is also supported by RHODE (Section 5.3.1). Thus, different RNN architectures can be enabled using the main building blocks of RHODE. Yet, it is worth mentioning that the implementation of GRUs under MHE will increase RHODE's computation and communication overhead and its evaluation on more complex RNN architectures is a direction for future work.

B.3 Optimization Extensions

Optimizations for Convolutional Neural Networks. We present a scheme for applying the convolutions on the slots, similar to FC layers, by representing them with a matrix multiplication. Convolution on a matrix, however, can be performed with a simple polynomial multiplication by using the coefficients of the polynomial. This operation requires a Fast-Fourier Transform (FFT) from slots (Number Theoretic Transform (NTT)) to coefficients domain, and vice versa (inverseFFT) for switching between CV to pooling or FC layers. Although it achieves better performance for CV layers, domain-switching is expensive. In the case of multiple CV layers before an FC layer, this operation could be embedded into the distributed bootstrapping ($\mathsf{DBootstrapALT}(\cdot)$) for efficiency. The evaluation of the trade-off between the two solutions for larger matrix dimensions is an interesting direction for future work.

Graphics Processing Units (GPUs). In this work, we evaluate our system on CPUs. Using GPUs to improve POSEIDON's performance requires GPU-compatible cryptographic functions, i.e., extending the underlying cryptographic library Lattigo [200]. In a recent work, Badawi et al. [34] proposed the first GPU implementation of the full RNS-variant of the CKKS scheme, for which they report speedups of one to two orders of magnitude over a CPU implementation. Hence, GPU-accelerated FHE is an option that could greatly improve the practicality of POSEIDON.

Algorithm 9 Local Computation Phase (GRU)

Input: $\mathbf{U}_z, \mathbf{U}_r, \mathbf{U}_n, \mathbf{W}_z, \mathbf{W}_r, \mathbf{W}_n, \mathbf{V}, \mathbf{b}_y, \mathbf{b}_z, \mathbf{b}_r, \mathbf{b}_n, \mathbf{h}_{prev}, X_i = (\bar{x_1}, \bar{x_2}, \cdots, x_t), Y_i = (\bar{y_1}, \bar{y_2}, \cdots, \bar{y_t})$

Output: $\nabla\mathbf{U}_z, \nabla\mathbf{U}_r, \nabla\mathbf{U}_n, \nabla\mathbf{W}_z, \nabla\mathbf{W}_r, \nabla\mathbf{W}_n, \nabla\mathbf{V}, \nabla\mathbf{b}_y, \nabla\mathbf{b}_z, \nabla\mathbf{b}_r, \nabla\mathbf{b}_n$

$\mathbf{h}_0 \leftarrow \mathbf{h}_{prev}$
for $t \leftarrow 1 : T$ **do** ▷ Forward Pass
  $\mathbf{z}_{raw} \leftarrow \mathbf{h}_{t-1} \times \mathbf{W}_z + \bar{x_t} \times \mathbf{U}_z + \mathbf{b}_z$ ▷ Z-Gate
  $\mathbf{z}_t \leftarrow \varrho(\mathbf{z}_{raw})$
  $\mathbf{r}_{raw} \leftarrow \mathbf{h}_{t-1} \times \mathbf{W}_r + \bar{x_t} \times \mathbf{U}_r + \mathbf{b}_r$ ▷ R-Gate
  $\mathbf{r}_t \leftarrow \varrho(\mathbf{r}_{raw})$
  $\mathbf{n}_{raw} \leftarrow (\mathbf{r}_t \odot \mathbf{h}_{t-1}) \times \mathbf{W}_n + \bar{x_t} \times \mathbf{U}_n + \mathbf{b}_n$ ▷ N-Gate
  $\mathbf{n}_t \leftarrow \varphi(\mathbf{n}_{raw})$
  $\mathbf{h}_t \leftarrow \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \mathbf{n}_t$ ▷ Hidden State
  $\mathbf{p}_t \leftarrow \mathbf{h}_t \times \mathbf{V} + \mathbf{b}_y$ ▷ Output Gate
end for
$\mathbf{dh}_{nxt} = \mathbf{0}$ ▷ $\frac{\partial loss[T+1:\infty]}{\partial \mathbf{h}_t}$
$\nabla\mathbf{U}_z, \nabla\mathbf{U}_r, \nabla\mathbf{U}_n, \nabla\mathbf{W}_z, \nabla\mathbf{W}_r, \nabla\mathbf{W}_n, \nabla\mathbf{V}, \nabla\mathbf{b}_y, \nabla\mathbf{b}_z, \nabla\mathbf{b}_r, \nabla\mathbf{b}_n = \mathbf{0}$
for $t \leftarrow T : 1$ **do** ▷ Backward Pass
  $\mathbf{dy} \leftarrow \mathbf{p}_t - \bar{y_t}$
  $\mathbf{dh} \leftarrow \mathbf{dy} \times \mathbf{V}^T + \mathbf{dh}_{nxt}$
  $\mathbf{dn} \leftarrow (1 - \mathbf{z}_t) \odot \mathbf{dh}$
  $\mathbf{dn}_{raw} \leftarrow \varphi'(\mathbf{n}_t) \odot \mathbf{dn} = (1 - n_t^2) \odot \mathbf{dn}$
  $\mathbf{dr} \leftarrow (\mathbf{dn}_{raw} \times \mathbf{W}_n^T) \odot \mathbf{h}_{t-1}$
  $\mathbf{dr}_{raw} \leftarrow \varrho'(\mathbf{r}_t) \odot \mathbf{dr} = \mathbf{r}_t \odot (1 - \mathbf{r}_t) \odot \mathbf{dr}$
  $\mathbf{dz} \leftarrow (\mathbf{h}_{t-1} - \mathbf{n}_t) \odot \mathbf{dh}$
  $\mathbf{dz}_{raw} \leftarrow \varrho'(\mathbf{z}_t) \odot \mathbf{dz} = \mathbf{z}_t \odot (1 - \mathbf{z}_t) \odot \mathbf{dz}$
  $\mathbf{dh}_h \leftarrow \mathbf{dh} \otimes \mathbf{z}_t$
  $\mathbf{dh}_z \leftarrow \mathbf{dz}_{raw} \times \mathbf{W}_z^T$
  $\mathbf{dh}_r \leftarrow \mathbf{dr}_{raw} \times \mathbf{W}_r^T$
  $\mathbf{dh}_n \leftarrow \mathbf{r}_t \otimes (\mathbf{dn}_{raw} \times \mathbf{W}_n^T)$
  $\mathbf{dh}_{nxt} \leftarrow \mathbf{dh}_z + \mathbf{dh}_h + \mathbf{dh}_n + \mathbf{dh}_r$
  $\nabla\mathbf{V} += \mathbf{h}_t \otimes \mathbf{dy}$
  $\nabla\mathbf{U}_z += \bar{x_t} \otimes \mathbf{dz}_{raw}$
  $\nabla\mathbf{U}_r += \bar{x_t} \otimes \mathbf{dr}_{raw}$
  $\nabla\mathbf{U}_n += \bar{x_t} \otimes \mathbf{dn}_{raw}$
  $\nabla\mathbf{W}_z += \mathbf{h}_{t-1} \otimes \mathbf{dz}_{raw}$
  $\nabla\mathbf{W}_r += \mathbf{h}_{t-1} \otimes \mathbf{dr}_{raw}$
  $\nabla\mathbf{W}_n += \mathbf{r}_t \odot (\mathbf{h}_{t-1} \otimes \mathbf{dn}_{raw})$
  $\nabla\mathbf{b}_y += \mathbf{dy}$
  $\nabla\mathbf{b}_z += \mathbf{dz}_{raw}$
  $\nabla\mathbf{b}_r += \mathbf{dr}_{raw}$
  $\nabla\mathbf{b}_n += \mathbf{dn}_{raw}$
end for

C Federated Multilayer Perceptron and Convolutional Neural Network Learning.

C.1 Comparison to Other State-of-the-Art Solutions

Table C.1 displays a qualitative comparison of POSEIDON with the state-of-the-art privacy-preserving neural network training and/or inference solutions. The MPC-setup row of the table denotes the number of parties responsible for the execution of the neural network operations. The adversarial model for data confidentiality indicates the capabilities of the parties (active (A) or passive (P)), and collusion shows the maximum number of possible colluding parties.

We note that several works allow as admissible adversary (collusions between one server and an arbitrary number of clients/data owners) [205]. For a fair comparison, we consider only the collusions permitted between the parties (servers) that are responsible for the training. To the best of our knowledge, POSEIDON is the only solution that performs both training and inference of neural networks, in an N-party setting, yet protects data and model confidentiality and withstands collusions up to $N-1$ parties. Therefore, our work differentiates itself from cloud outsourcing models and enables a privacy-preserving federated learning approach.

		XONN [237]	**Gazelle** [148]	**Blaze** [223]	**MiniONN** [184]	**ABY3** [204]	**SecureML** [205]	**SecureNN** [278]	**FALCON** [279]	**FLASH** [53]	**TRIDENT** [57]	**CryptoNets** [112]	**CryptoDL** [126]	[211]	**POSEIDON**
MPC Setup		2PC	2PC	3PC	2PC	3PC	2PC	3PC	3PC	4PC	4PC	1PC	1PC	1PC	**N-Party**
Private Infer.		✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Private Train.		✗	✗	✗	✗	✔	✔	✔	✔	✔	✔	✗	✔	✔	✔
Data Conf.															
Adversarial Model*		1 P	1 P	1 A	1 P	1 A/P	1 P	1 A/P	1 A/P	1 A	1 A/P	1 P'	1 P'	1 P'	$N-1$ P
Collusion*		No	No	No	No	No	No	No	No	No	No	NA	NA	NA	$N-1$
Techniques		GC,SS	HE,GC,SS	GC,SS	HE,GC,SS	GC,SS	HE,GC,SS	SS	SS	SS	GC,SS	HE	HE	HE	HE
Supported Layers	Linear	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
	Conv.	✔	✔	✗	✔	✔	✔	✔	✔	✔	✔	✔	✗	✗	✔
	Pooling	✔	✔	✗	✔	✔	✔	✔	✔	✔	✔	✔	✗	✗	✔

Table C.1: *Qualitative comparison of private deep learning frameworks. Conf. stands for confidentiality. A and P stand for active and passive adversarial capabilities, respectively. GC, SS, HE denote garbled-circuits, secret sharing, and homomorphic encryption. Adversarial model* and collusion* take into account the servers responsible for the training/inference.* 1 *P' denotes our interpretation as [112], [211], and [126] do not present an adversarial model. NA stands for not applicable.*

C.2 Approximated Activation Function Alternatives

For the piece-wise function ReLU, we propose two alternatives: (i) approximation of square-root for the evaluation of $\varphi(x) = 0.5(b+\sqrt{b^2})$ that is equivalent to ReLU, and (ii) approximating the *smooth* approximation of ReLU (SmoothReLU), or softplus, $\varphi(x) = \ln(1+e^x)$, both with least-squares. Our analysis shows that the latter achieves a better approximation for a degree $d_a = 3$, whereas the former approximates better the exact ReLU if one increases the multiplicative depth by 1 and uses $d_a = 7$. In our evaluations, we use SmoothReLU for efficiency.

We note that the derivative of softplus is a sigmoid function, and we evaluate the approximated sigmoid as the derivative, as this achieves better accuracy. Finally, the Lattigo cryptographic library [200] comes with a native way of approximating functions using Chebyshev interpolants and an efficient algorithm to evaluate polynomials in standard or Chebyshev basis. The least-squares is the optimal solution for minimizing the squared error over an interval, whereas Chebyshev asymptotically minimizes the maximum error. Hence, Chebyshev is more appropriate for keeping the error bounded throughout the whole interval, but requires a larger degree for a high accuracy approximation. Thus, when a high accuracy is needed for the activation function, we suggest using a large degree Chevshev interpolant. The advantage of the Chebyshev approximation, along with its asymptotic optimality and simple computation, is that it ensures that the polynomial interpolant has small coefficients and is numerically stable regardless of its degree, which is well suited for homomorphic evaluation.

C.3 Approximation of the Max/Min Pooling and Its Derivative

For the sake of clarity, we describe the max-pooling operation. Given a vector $x = (x[0], \ldots, x[n-1])$ the challenge is to compute y with $y[0 \leq i < n] = \max(x)$. To approximate the index of $\max(x)$, which can then be used to extract the max value of x, we follow an algorithm similar to that presented in [63], described below.

Given two real values a, b, with $0 \leq a, b \leq 1$, we observe the following: If $a > b$, then $a - b < a^d - b^d$ for $d > 1$, i.e., with increasing d, smaller values converge to zero faster and the ratio between the maximum value and other values increases. The process can be repeated to increase the ratio between a and b but, unless $a = 1$, both values will eventually converge to zero. To avoid this, we add a second step that consists in renormalizing a and b by computing $a = a/(a+b)$ and $b = b/(a+b)$. Thus, we ensure that after each iteration, $a + b = 1$ and since b will eventually converge to zero, a will tend towards 1. If $a = b$, both values will converge to 0.5. This algorithm can be easily generalized to vectors: Given a vector $x = (x[0], \ldots, x[n-1])$, at each iteration it computes $x[i] = x[i]^d / \sum_{j=0}^{n-1} x[j]^d$, and multiplies the result with the original vector to extract the maximum value.

This max-pooling algorithm is a time-consuming procedure as it requires computing an expensive inverse function, especially if a high accuracy is desired or if the input values are very small. Instead, we employ a direct approach using $\max(a, b) = \frac{1}{2}(a+b+\sqrt{(a-b)^2})$, where

the square-root can be approximated by a polynomial. To compute the maximum value for a kernel $f = k \times k$, we iterate $\log(f)$ times $\boldsymbol{c}_{i+1} = \max(\boldsymbol{c}_i, \mathsf{RotL}_{2^i}(\boldsymbol{c}_i))$. As each iteration consumes all levels, we use $\mathsf{DBootstrap}(\cdot)$ $\log(f)$ times. Hence, we suggest using the average-pooling instead, which is more efficient and precise, e.g., Dowlin et al. [112] show that low-degree approximations of max-pooling will converge to a scalar multiple of the mean of k values. We provide microbenchmarks of both max and average-pooling in Appendix C.5.1.

C.4 Technical details of Distributed Bootstrapping with Arbitrary Linear Transformations ($\mathsf{DBootstrapALT}(\cdot)$)

A linear transformation $\phi(\cdot)$ over a vector of n elements can be described by a $n \times n$ matrix. The evaluation of a matrix-vector multiplication requires a number of rotations proportional to the square-root of its non-zero diagonals, thus, this operation becomes prohibitive when the number of non-zero diagonals is large.

Such a linear transformation can be, however, efficiently carried out *locally* on a secret-shared plaintext, as $\phi(msg + M) = \phi(msg) + \phi(M)$ due to the linearity of $\phi(\cdot)$. Moreover, because of the magnitude of $msg + M$ (100 to 200 bits), arbitrary precision complex arithmetic with sufficient precision should be used for $\mathsf{Encode}(\cdot)$, $\mathsf{Decode}(\cdot)$, and $\phi(\cdot)$ to preserve the lower bits. The collective bootstrapping protocol in [209] is performed through a conversion of an encryption to secret-shared values and a re-encryption in a refreshed ciphertext. We leverage this conversion to perform the aforementioned linear transformation in the secret-shared domain, before the refreshed ciphertext is reconstructed. This is our $\mathsf{DBootstrapALT}(\cdot)$ protocol (Protocol 5).

When the linear transformation is simple, i.e., it does not involve a complex permutation or requires a small number of rotations, the $\mathsf{Encode}(\cdot)$ and $\mathsf{Decode}(\cdot)$ operations in Line 8, Protocol 5 can be skipped. Indeed, those two operations are carried out using arbitrary precision complex arithmetic. In such cases, it is more efficient to perform the linear transformation directly on the encoded plaintext.

Security Analysis of $\mathsf{DBootstrapALT}(\cdot)$. This protocol is a modification of the $\mathsf{DBootstrap}(\cdot)$ protocol of Mouchet et al. [209], with the difference that it includes a product of a public matrix. Both $\mathsf{DBootstrap}(\cdot)$ and $\mathsf{DBootstrapALT}(\cdot)$ for CKKS differ from the BFV version proposed in [209] in which the shares are not unconditionally hiding, but statistically or computationally hiding due to the incomplete support of the used masks. Therefore, the proof follows analogously the passive adversary security proof of the BFV $\mathsf{DBootstrap}(\cdot)$ protocol in [209], with the addition of Lemma 1 which guarantees the statistical indistinguishablity of the shares in $\mathbb{C}$. While the RLWE problem and Lemma 1 do not rely on the same security assumptions, the first one being computational and the second one being statistical, given the same security parameter, they share the same security bounds. Hence, $\mathsf{DBootstrap}(\cdot)$ and $\mathsf{DBootstrapALT}(\cdot)$ provide the same security as the original protocol of Mouchet et al. [209].

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

We refer to Algesheimer et. al [29, Section 3.2], and Schoenmakers and Tuyls [247, Appendix A], for the proof of the statistical λ-indistinguishability.

We recall that an encoded message msg of $\mathscr{N}/2$ complex numbers with the CKKS scheme is an integer polynomial of $\mathbb{Z}[X]/(X^{\mathscr{N}}+1)$. Given that $||msg|| < 2^{\delta}$, and a second polynomial M of $\mathscr{N}$ integer coefficients with each coefficient uniformly sampled and bounded by $2^{\lambda+\delta}-1$ for a security parameter λ, Lemma 1 suggests that $\Pr[||msg^{(i)} + M^{(i)}|| \geq 2^{\lambda+\delta}] \leq 2^{-\lambda}$, for $0 \leq i < \mathscr{N}$ and where i denotes the i^{th} coefficient of the polynomial. That is, the probability of a coefficient of $msg + M$ to be distinguished from a uniformly sampled integer in $[0, 2^{\lambda+\delta})$ is bounded by $2^{-\lambda}$. Hence, during Protocol 5 each party samples its polynomial mask M with uniform coefficients in $[0, 2^{\lambda+\delta})$. The parties, however, should have an estimate of the magnitude of msg to derive δ, and a probabilistic upper-bound for the magnitude can be computed by the circuit and the expected range of its inputs.

In Protocol 5, the masks M_i are added to the ciphertext of R_{Q_ℓ} during the decryption to the secret-shared domain. To avoid a modular reduction of the masks in R_{Q_ℓ} and ensure a correct re-encryption in R_{Q_L}, the modulus Q_ℓ should be large enough for the additions of N masks. Therefore, the ciphertext modulus size should be greater than $(N+1) \cdot ||M||$ when the bootstrapping is called. For example, for $N = 10$, a Q_L composed of a 60 bits modulus, a message msg with $||msg|| < 2^{55}$ (taking the scaling factor Δ into account) and $\lambda = 128$, we should have $||M_i|| \geq 2^{183}$ and $Q_\ell > 11 \cdot 2^{183}$. Hence, the bootstrap should be called at Q_3 because $Q_2 \approx 2^{180}$ and $Q_3 \approx 2^{240}$. Although the aforementioned details suggest that $\mathsf{DBootstrapALT}(\cdot)$ is equivalent to a depth 3 to 4 circuit, depending on the parameters, it is still compelling, as it enables us to refresh a ciphertext and apply an arbitrary complex linear transformation at the same time. Thus, its cost remains negligible compared to a centralized bootstrapping where any transformation is applied via rotations.

C.5 Supplementary Experimental Results

We provide further experimental results of POSEIDON, that were left out of the main text due to space constraints. We provide the microbenchmarks and execution times of various neural network architectures.

C.5.1 Microbenchmarks

We present microbenchmark timings for the various functionalities and sub-protocols of POSEIDON in Table C.3. These are measured in an experimental setting with $N = 10$ parties,

Topology	**Local Computation:**FF	**Local Computation:**BP	**REDUCE** (s)	Comm.	**Total**
(6, 1, 1, 2)	0.40	0.36	0.05	0.47	1.28
(6, 2, 2, 2)	0.44	0.43	0.04	0.52	1.43
(16, 2, 2, 8)	0.48	0.42	0.03	0.54	1.47
(16, 4, 4, 8)	0.47	0.45	0.04	0.51	1.47
(32, 8, 8, 8)	0.57	0.50	0.04	0.45	1.56
(32, 16, 16, 8)	0.55	0.52	0.03	0.47	1.57
(64, 8, 8, 8)	0.55	0.50	0.04	0.45	1.54
(64, 32, 32, 8)	0.55	0.62	0.04	0.43	1.64
(128, 32, 32, 8)	0.60	0.63	0.04	0.38	1.65
(128, 64, 64, 8)	0.78	0.80	0.05	0.56	2.19
(256, 64, 64, 8)	1.04	1.36	0.06	0.38	2.84
(256, 128, 128, 8)	2.01	2.62	0.11	0.61	5.35

Table C.2: *Execution times (in seconds) per-global-iteration of various neural network architectures with batch size $B = 120$, $N = 10$ parties.* ***Local Computation:****FF,* ***Local Computation:****BP, Comm. stand for* ***Local Computation:****feed-forward,* ***Local Computation:*** *backpropagation, and communication respectively.*

a dimension of $d = 32$ features, $h = 64$ neurons in a layer or kernel size $k = 3 \times 3$, and degree $d_a = 3$ for the approximated activation functions for FC, CV, FC backpropagation, CV backpropagation, and average-pooling benchmarks. These benchmarks represent the processing of 1 sample per party, thus $b = 1$. For max-pooling, we achieve a final precision of 7 bits with a square-root approximated by a Chebyshev interpolant of degree $d_a = 31$. We observe that max-pooling is 6 times slower than average-pooling, has a lower precision, and needs more communication due to the large number of $\mathsf{DBootstrap}(\cdot)$ operations. For 12-bits precision, max-pooling takes 4.72s. This supports our choice of using average-pooling instead of max-pooling in the encrypted domain. The communication column shows the overall communication between the parties in MB. As several HE-based solutions [112, 148, 126], use square activation functions, we also benchmark them and compare them with the approximated activation functions with $d_a = 3$.

We note that **Setup Phase** stands for the offline phase and it incorporates the collective generation of the encryption, decryption, evaluation, and rotation keys based on the protocols presented in [209]. Most of the time and bandwidth are consumed by the generation of the rotation keys needed for the training protocol. We refer the reader to [209, 200] for more information about the generation of these keys. Although we present the **Setup Phase** microbenchmark to hint about the execution time and communication overhead of this offline phase, we note that it is a non-trivial task to extrapolate its costs for a generic neural network structure. **Model-Update Phase** indicates the reducing step for 1 weight matrix (updating the weight matrix in root) and collectively refreshing it.

Functionality	Execution time (s)	Comm. (MB)
ASigmoid/ASmoothRelu	0.050	-
ASigmoidD/ASmoothReluD	0.022	-
Square / SquareD	0.01 / 0.006	-
ASoftmax	0.07	-
DBootstrap(·)	0.09	6.5
DBootstrapALT(·) ($\log_2(h)$ rots)	0.18	6.5
DBootstrapALT(·) with Average Pool	0.33	6.5
MaxPooling	2.08	19.5
FC layer / FC layer-backprop	0.09 / 0.13	-
CV layer / CV layer-backprop	0.03 / 0.046	-
DKeySwitch	0.07	23.13
Setup Phase	18.19	3.8k
Local Computation Phase (only communication)	0.03	18.35
Aggregate Phase	0.09	7.8
Model-Update Phase	0.1	6.5

Table C.3: *Microbenchmarks of different functionalities for* $N = 10$ *parties,* $d = 32$, $h = 64$, $\mathcal{N} = 2^{13}$, $d_a = 3$, $k = 3 \times 3$.

We show how to use these microbenchmarks to *roughly* estimate the *online* execution time and communication overhead of one global iteration for a chosen neural network structure. We combine the results of Table C.3 for layers/kernels with specific size, fixed $\mathcal{N}$, d_a, and N, with those of Table 4.2 that show POSEIDON's linear scalability with N for the operations requiring communication, linear scalability with $\mathcal{N}$, and logarithmic scalability with d. We scale the execution time of each functionality for the various parameters depending on the theoretical complexity. Here exemplify the time for computing one global iteration with $N = 50$ parties, for a CNN with 32×32 input images, 1 CV layer with kernel size $k = 6 \times 6$, 1 average-pooling layer with $k = 3 \times 3$, and 1 FC layer with $h = 128$ neurons. We observe that the number of parties N is 5 times bigger than the setting of Table C.3, thus yields one round of communication of **Local Computation Phase** and **Aggregate Phase** as $0.03 \times 5 = 0.15$s and $0,09 \times 5 = 0.45$s, respectively. The **Model-Update Phase** microbenchmark is calculated for 1 weight matrix, thus with 2 weight matrices, **Model-Update Phase** will consume 0.2s. For the LGD computation, we start with the CV layer with $k = 6 \times 6$ kernel size. We remind that CV layers are represented by FC layers, thus the kernel size affects the run-time logarithmically; we multiply the CV layer execution time by 2 ($0.03 \times 2 = 0.06$) followed by an activation execution time of 0.05s. For more than 1 filter per CV layer, this number should be multiplied by the number of filters (assuming no parallelization). Then, we use $\mathsf{DBootstrapALT}(\cdot)$ with average pooling to refresh the ciphertext, compute the pooling together with the backpropagation values yielding an execution time of 0.33s scaled to 50 parties as $0.33 \times 5 = 1.65$s. Lastly, since its execution time scales logarithmically with the number of neurons, the FC layer will be executed in $0.09/\log_2(64) * \log_2(128) = 0.105$s followed by another activation of 0.05s. A similar approach is then used for the backward pass and with FC layer-backprop, CV layer-backprop, and using the derivatives of the activation functions. The microbenchmarks are calculated using 1 sample per-party; thus, to extrapolate the time for $b > 1$ *without any parallelization*, the total time for the forward and backward passes should be multiplied by b. Finally, as this example is a CNN, we already refresh the ciphertexts after each CV layer both in the forward and backward pass, to compute pooling or to re-arrange the slots. To extrapolate the times for MLPs, the number of bootstrappings are calculated as described in Section 4.3.6 and this is multiplied by the $\mathsf{DBootstrap}(\cdot)$ benchmark. Extrapolating the communication overhead for a global iteration is straightforward: As the communication scales linearly with the number of parties, we scale the overheads given in Table C.3 with N. For example, with $N = 50$ parties, **Local Computation Phase** and **Aggregate Phase** consume $18.32 \times 5 = 91.6$MB and $7.8 \times 5 = 39$MB, respectively. Similarly, the total number of $\mathsf{DBootstrap}(\cdot)$, and its variants, should be multiplied by 5. Lastly, in this example, the weight or kernel matrices fit in one ciphertext ($\mathcal{N}/2 = 4,096$ slots); if more than 1 cipher per weight matrix is needed, the aforementioned numbers should be multiplied by the number of ciphertexts.

C.5.2 Benchmarks on Various Neural Network Topologies

We provide execution times of different network topologies in Table C.2. "Topology" represents number of features (d), hidden neurons in each layer (h_1, h_2), and number of output labels

(h_3) as (d, h_1, h_2, h_3). We use local batch size $b = 12$ and global batch size $B = 120$ for $N = 10$ parties. We use ASigmoid with $d_a = 3$ as an activation function. The execution times indicate the time required for one global iteration, i.e., a processing of the global batch, and we report forward pass, backpropagation and the number of communications in separate columns. The "Communication" column includes the communication required for the **Aggregate Phase** phase and for DBootstrap$(\cdot)$ operations. We provide the feed-forward and backpropagation times in **Local Computation Phase** separately.

C.5.3 Benchmarks on Various Convolutional Neural Network Topologies

We provide extrapolated execution times of different CNN topologies in Table C.4. As we introduce several operations (derivative of pooling) in the forward pass to bootstrapping function, we do not separate between forward pass and backpropagation times, and we introduce the overall execution times. "Topology" represents the padded (power-of-two) number of features (d), kernel size for CV layer $(CV[n \times n])$, kernel size for average pooling layer $(P[n \times n])$, and h number of neurons in the last FC layer connected to h_ℓ output layers $(FC[h : h_\ell])$ as $(d, CV[n \times n], P[n \times n], FC[h : h_\ell])$.

Topology	Execution Time (s)
$(256, CV[2 \times 2], P[2 \times 2], FC[16 : 2])$	1.42
$(512, CV[2 \times 2], P[2 \times 2], FC[16 : 2])$	1.52
$(512, CV[2 \times 2], P[2 \times 2], FC[32 : 2])$	1.88
$(784, CV[2 \times 2], P[2 \times 2], FC[32 : 2])$	2.12
$(784, CV[2 \times 2], P[2 \times 2], FC[32 : 10])$	2.56
$(784, CV[2 \times 2], P[2 \times 2], CV[2 \times 2], P[2 \times 2], FC[32 : 10])$	3.88

Table C.4: *Execution times per-global-iteration of various CNN architectures, with batch size* $B = 120$, $N = 10$ *parties.*

D Privacy-Preserving Federated Recurrent Neural Networks

D.1 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 [132]. The dataset contains 12 files, each corresponding to a distribution company and to a specific time period. Table D.1 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 ($d = 6$). 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 necessarily know the range of the energy consumption values for other data holders).

Stock Prices (Stock): This dataset contains historical daily stock prices and statistics for 10 companies, and was collected by the Yahoo! Finance website [261]. Similar to the HEC dataset, we perform Min-Max scaling on each file separately. Table D.2 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 datasets for the forecasting tasks, we pre-process and slice them using a sliding window of size T for varying timesteps $T = [5, 10, 20]$ for our model performance analysis.

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 [79]. 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 [236] 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).

Company Name	File Name [132]	#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 D.1: *Detailed description of the HEC dataset.*

Company Name	File Name [261]	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 D.2: *Detailed description of the Stock dataset.*

Algorithm 10 Matrix Multiplication

Inputs: A, B, two $n \times n$ matrices, expressed as $\ell \times \ell$ matrices, where each entry is a sub-matrix of dimension $(n/\ell) \times (n/\ell)$.

Outputs: $C \leftarrow A \otimes B$

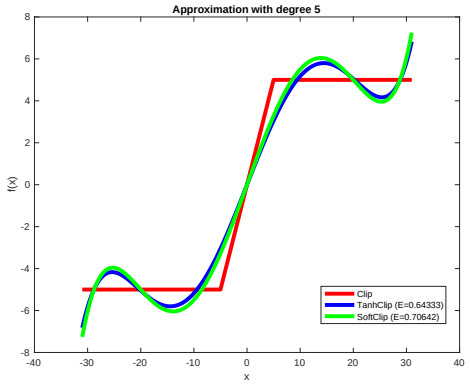
for $i = 1 \rightarrow \ell$ **do**
  for $j = 1 \rightarrow \ell$ **do**
    $\mathbf{b}_{i,j,1} \leftarrow \sigma(B_{i,j})$
    for $v = 2 \rightarrow n/\ell$ **do**
      $\mathbf{b}_{i,j,v} \leftarrow \psi_v(\mathbf{b}_{i,1})$
    end for
  end for
end for
for $i = 1 \rightarrow \ell$ **do** ▷ Iterate on each entry of A.
  for $j = 1 \rightarrow \ell$ **do**
    $\mathbf{a}_1 \leftarrow \tau(A_{i,j})$ ▷ For each entry of A, compute the linear transformations τ and ϕ.
    for $v = 2 \rightarrow n/\ell$ **do**
      $\mathbf{a}_v \leftarrow \phi_v(\mathbf{a}_1)$
    end for
    for $v = 1 \rightarrow \ell$ **do** ▷ Product between an entry of A and each corresponding row of B.
      $d \leftarrow \mathbf{a}_1 \odot \mathbf{b}_{j,1}$
      for $k = 2 \rightarrow n/\ell$ **do**
        $d \leftarrow d + \mathbf{a}_k \odot \mathbf{b}_{j,k}$
      end for
      $C_{i,v} \leftarrow C_{i,v} + d$ ▷ Aggregate the intermediate results on C.
    end for
  end for
end for

D.2 Matrix Multiplication Protocol

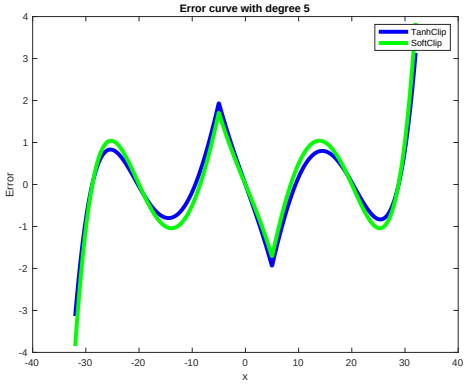
Algorithm 10 presents the plaintext operations required for the matrix multiplication protocol with multi-dimensional packing (see Section 5.3.1). This algorithm is described in the plaintext space for clarity; it is the same as Algorithm 7 with further details about the linear transformations employed. The first outer loop (Line 1) pre-computes the linear transformations on the entries (sub-matrices) of the right-matrix B while the second one (Line 9) calculates the entries of the left-matrix A. The inner loop starting at Line 15 computes the product between one entry of A and each corresponding row of B. Finally, it aggregates the intermediate results on the output C (Line 20).

D.3 Approximation of the proposed clipping functions

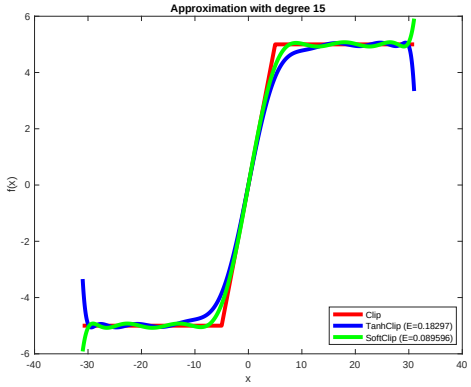
Figure D.1 displays the approximation and error curves of $\mathsf{TanhClip}(\cdot)$ and $\mathsf{SoftClip}(\cdot)$ (compared to the baseline clipping function ($\mathsf{Clip}(\cdot)$)) by employing the Minimax method with degrees $\mathfrak{p} = 5$ and $\mathfrak{p} = 15$ in the approximation interval $[-30, 30]$ 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 $\mathsf{TanhClip}(\cdot)$ is slightly smaller than $\mathsf{SoftClip}(\cdot)$ (0.643 vs.0.706). For higher degrees ($\mathfrak{p} = 15$), however, $\mathsf{SoftClip}(\cdot)$ achieves a better approximation (i.e., $E = 0.089$ vs. $E = 0.183$ for $\mathsf{TanhClip}(\cdot)$). Moreover, the divergence of $\mathsf{SoftClip}(\cdot)$ near the limits of the approximation interval is slower than $\mathsf{TanhClip}(\cdot)$ (see Figure D.1c); 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.



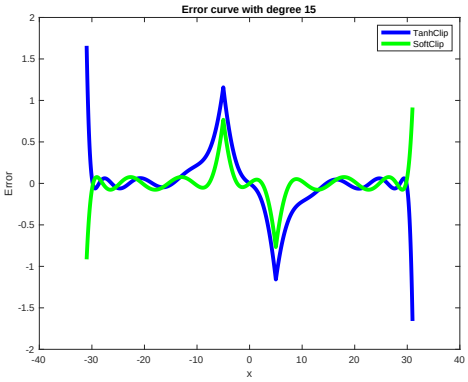
(a) Approximation curves with $\mathfrak{p} = 5$



(b) Error curves with $\mathfrak{p} = 5$



(c) Approximation curves with $\mathfrak{p} = 15$



(d) Error curves with $\mathfrak{p} = 15$

Figure D.1: *Approximation and error curves of* $\mathsf{TanhClip}(\cdot)$ *and* $\mathsf{SoftClip}(\cdot)$ *compared to baseline clipping function* $\mathsf{Clip}(\cdot)$ *with degrees* $\mathfrak{p} = 5$ *and* $\mathfrak{p} = 15$ *in the interval* $[-30,30]$ *for a clipping threshold of* $|m| = 5$. *Error curves are plotted in the range of* $[-32,32]$. *E in the legends of the approximation plots indicates the average absolute error per approximation in the interval* $[-30,30]$.

E Privacy-Preserving Federated Neural Network Learning for Disease-Associated Cell Classification

E.1 Data Preprocessing and Parameter Selection

Our data preprocessing is similar to the one used in CellCnn [32]. To address the distributed setting, we split individual donors in the training set to N institutions. Each party then generates the multi-cell inputs similar to CellCnn (Figure 6.1) by selecting c cells per sample, and z samples per class or per patient, depending on the experiment. As a result, each multi-cell input sample has a size of $c \times m$, where m is the number of markers; and the total training set per party has $o \times z$ multi-cell inputs, where o is the number of labels when the data is generated in a per class-basis. The total training set has $p \times z$ multi-cell inputs, where p is the number of patients in that institution when the data is generated in a per patient-basis.

For accuracy evaluation, we use two test datasets for each experimental setting: One is generated by multi-cell inputs of c cells and z samples drawn from test set as in training set, and one is generated with the g of cells per individual to predict the phenotype where g is the minimum of all available cells per individual in the test set. Lastly, for a fair comparison, we use the same test set generated in all settings per dataset.

For all experimental settings, we scale and standardize the marker distributions, based on the training data.

Below, we give the parameters for each experimental setting.
RRMS/NIND experiments. For RRMS and NIND experiments in Figures 6.3 and 6.4, we generate two training datasets: (i) multi-cell inputs with 100 cells were drawn for each class label to generate a dataset of 30000 samples (phenotype-based) and (ii) multi-cell inputs with 2000 cells were drawn from each patient to generate 480 samples (patient-based). We report the median accuracy in Figures 6.3 and 6.4, for patient-based multi-cell input generation in Table E.1.

The size of the test set for multi-cell inputs, is set to 10000 for the phenotype-based multi-cell generation, and to 96 for patient-based multi-cell generation setting. The test set for

phenotype classification is 12 donors for all RRMS and NIND experiments.
CMV experiments. We transform the marker measurement with the inverse hyperbolic sine function with a cofactor of 5. The training and test datasets respectively comprise 14 and 6 donors.

For the CellCnn results in Table E.1, 200 cells were drawn for each class label to generate one multi-cell sample and 2000 samples are generated per label. For all distributed settings, we also generate 200 cells per sample and gradually decrease the number of samples bagged in each party for a fair comparison.

The size of the test set including multi-cell inputs is set to 4000 and the size of the test set for phenotype classification is 6 donors.
AML Experiments. For the AML experiments in Table E.1, we draw 200 cells per class label to generate multi-cell samples and 1000 samples generated per label. Note that there are 3 class labels for this set of experiments. For all distributed settings, we generate 200 cells per sample and gradually decrease the number of samples, as in other experimental settings. The size of the test set including multi-cell inputs, is set to 3000 and the size of the test set for phenotype classification is 6 donors.
Machine Learning Parameters. For all accuracy experiments, we use 8 filters, average pooling, 1 local iteration per party before aggregating local gradients, identity activation after convolution, and an approximated sigmoid activation for the dense layer. For the baseline (CellCnn), we rely on their original optimizer, ADAM, and for PRICELL, we use SGD with momentum (μ) to enable efficient training of the neural network under encryption. We vary $\mu = 0.5 - 0.9$ and the learning rate $\eta = 0.0001 - 0.01$ for the distributed setting.

For the RRMS and NIND experiments, we use a batch size of 64 for the baseline (CellCnn) and gradually decrease the batch size proportionally to the number of parties when data is distributed. For example, when the number of parties is 4, the local batch size is 16. We use the approximated sigmoid activation in $[-1,1]$ with a polynomial degree of 3 for the dense layer, and 30 epochs.

For the CMV experiments, we use a batch size of 200 for the baseline (CellCnn) and gradually decrease the batch size proportionally to the number of parties when the data is distributed. For example, when the number of parties is 2, the local batch size is 100. We use an approximated sigmoid activation in $[-3,3]$ with a polynomial degree of 3 for the dense layer, and 20 epochs.

For all AML experiments, we use a batch size of 200 for the baseline (CellCnn) and 100 for the local batch size in the distributed setting with 2 parties. We use an approximated sigmoid activation in $[-3,3]$ with a polynomial degree of 3 for the dense layer, and 20 epochs.

Lastly, for the Local training in Figures 6.2, 6.3, and 6.4 or the Local row in Table E.1, we use the original CellCnn architecture, with the same baseline parameters and average the accuracy, precision, and recall over N local parties' models.

Security Parameters. Unless otherwise stated, all experiments use a cyclotomic polynomial ring of dimension $\mathcal{N} = 2^{15}$ and an initial level $L = 10$, which provides 2^{14} slots per ciphertext and allows for a depth-10 circuit before bootstrapping is needed (10 operations to be carried out before bootstrapping). For the inference times given in Table 6.1, we start with an initial level of 4 as we do not need the backpropagation. This enables a more efficient forward pass as operations carried out on a ciphertext with a lower level are less expensive. All our cryptographic parameters ensure at least 128-bit security level during the training and up to 256-bit security during the inference.

E.2 Summary of Experiments

In Table E.1, we show the median accuracy, precision, recall, and F-score values of 10 runs for RRMS and NIND experiments with patient-based sub-sampling shown in Figures 6.3c, 6.3d and Figures 6.4c, 6.4d, and CMV experiments for phenotype-based sub-sampling shown in Figure 6.2. We also report these metrics for the AML classification for the centralized and two-party PRICELL settings.

We note that for 3-class classification, i.e., AML, we rely on macro-averaging on metrics, and we calculate the F-score in Table E.1 over the averaged precision and recall for the Local-training experimental setting.

Our results show that PRICELL achieves an accuracy comparable to the centralized and non-private solutions. The accuracy achieved by PRICELL remains almost the same as the centralized one, and the slight decrease in phenotype classification in NIND classification is due to the limited number of samples in the test set for this task, i.e., there are only 12 patients in the NIND phenotype test set, which results in accuracy decrease of 4% in the median value when the trained model misses only one patient classification.

Lastly, we note that the differences in the precision and recall values are due to the nature of the preprocessing and training mechanism: the random selection of multi-cell inputs generates higher or lower precision and recall values depending on the eventual selection, even in the centralized and no privacy-protection solution.

E.3 Downstream Analysis

The original CellCnn [32] study aims at detecting the rare disease-associated cell subsets via learned filter weights. The final filter weights are used to select phenotype-associated cell subsets via a filter response, i.e., the weighted sum of the abundance profile for each cell. As the cell subset selected by a filter can contain more than one cell type, the authors perform a density-based clustering of the group of cells with high cell-filter responses.

We perform an analogous analysis to evaluate the effect of our introduced changes in the

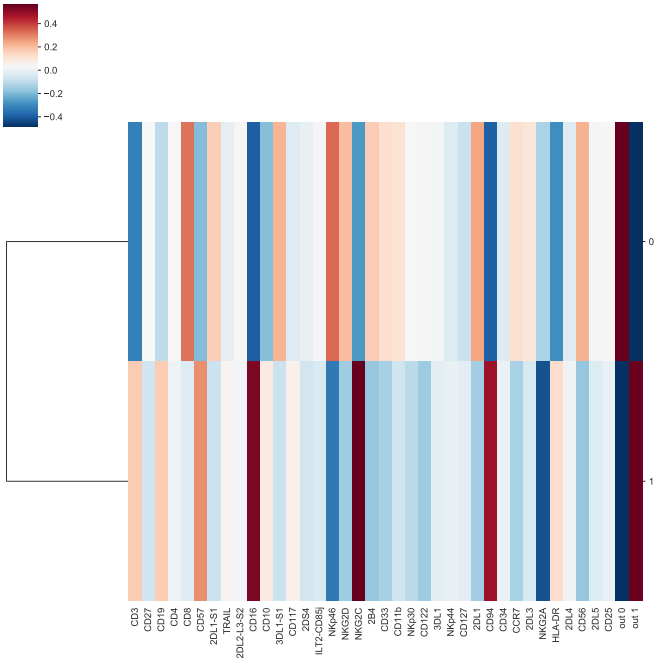
original neural network architecture, namely the average pooling, the approximated activation functions, and the optimizer. We introduce these changes in CellCnn's original implementation, simulate our encryption on centralized data, and conduct further analysis by using their downstream analysis [32]. We use the CMV infection dataset with $c = 200$ cells per multi-cell input and $z = 1000$ samples per phenotype to generate the training dataset. The test set is generated as explained in the Data Preprocessing and Parameter Selection section. We train 20 models for CellCnn and 20 models for PRICELL simulation and take the best 3 models for each approach based on the validation accuracy, as in the original work [32]. In all model training, we use 20 epochs with early-stopping and varying numbers of filters in each model training.

In Figure E.1, we show the consensus filters, i.e., one representative filter per class (phenotype) that has minimum distance to all other members of the hierarchically clustered filters, based on a threshold of 0.2, found by CellCnn and PRICELL simulation, respectively. In both Figures E.1a and E.1b, we observe that the filter which is positively associated (second filter) with previous CMV infection gives more weights to the CD16, CD57, NKG2C, and CD94 markers. We note that while the trend of consensus filters is similar, the distribution of the consensus filters, i.e., the final filter weights and the scale of the values, differs between CellCnn and PRICELL. This is due to our approximated activation function that affects the final values of the filter weights but does not affect the interpretation from the consensus filters. Similar results were found for the repetition of these experiments, which suggests that, as in the original work [32], our encrypted model is able to find natural killer (NK) cell populations associated with prior CMV infection.

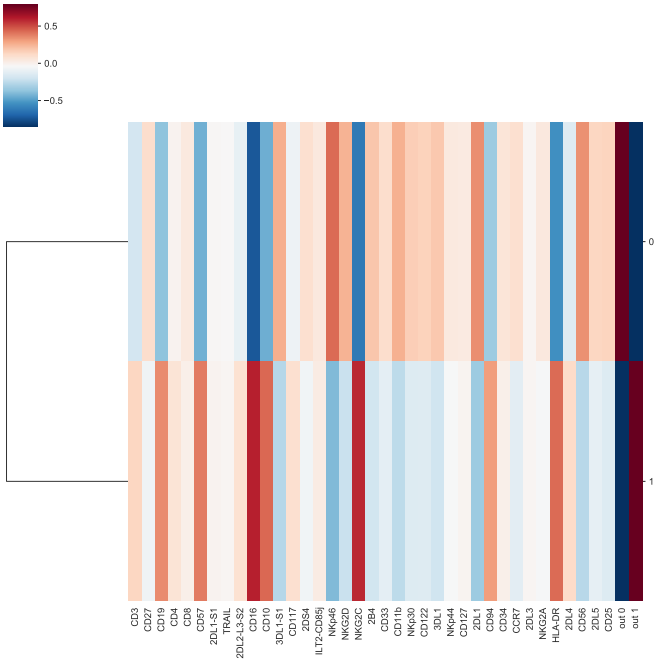
In Figure E.2, we show the boxplot of the selected cell population frequencies from the test samples of the CMV- and CMV+ classes by using the positively associated filter. Although CellCnn has higher discriminative frequencies, PRICELL simulation is able to select CMV+ cell populations with the positively associated filter.

Finally, we show in Figure E.3 the marker expression profiles for all cells vs. cell population selected by the positively associated filter learned by CellCnn training (Figure E.3a) and by PRICELL simulation(Figure E.3b). In both CellCnn and PRICELL training, we again observe that the positively associated filter weighs CD16, CD57, NKG2C, and CD94 markers more than the others.

In summary, we show that the PRICELL training does not affect the further findings of an existing work that performs training on a centralized data without integrating a privacy-preserving mechanism.

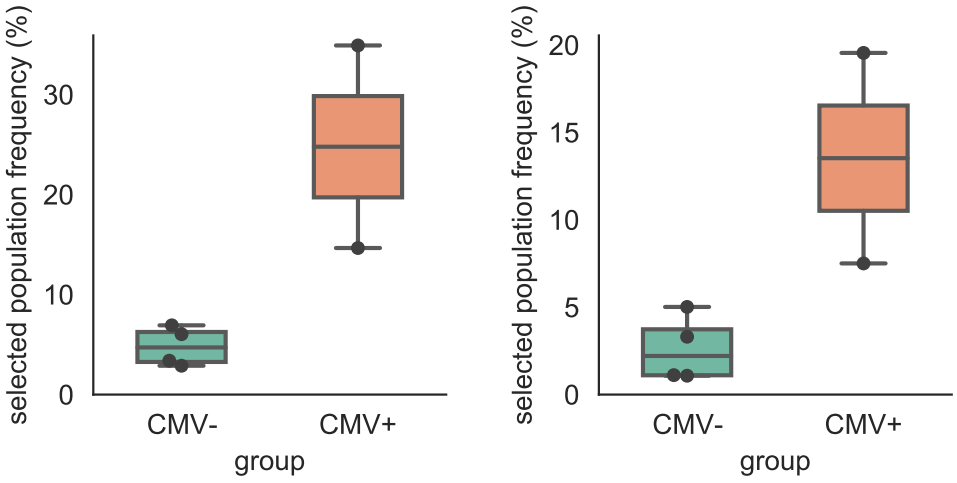


(a) Consensus filters found by CellCnn



(b) Consensus filters found by PRICELL simulation

Figure E.1: *Comparison of the consensus filters (one representative filter per class label) learned by (a) CellCnn original architecture, and by (b)* PRICELL*'s adapted architecture for encrypted training on CMV dataset.*

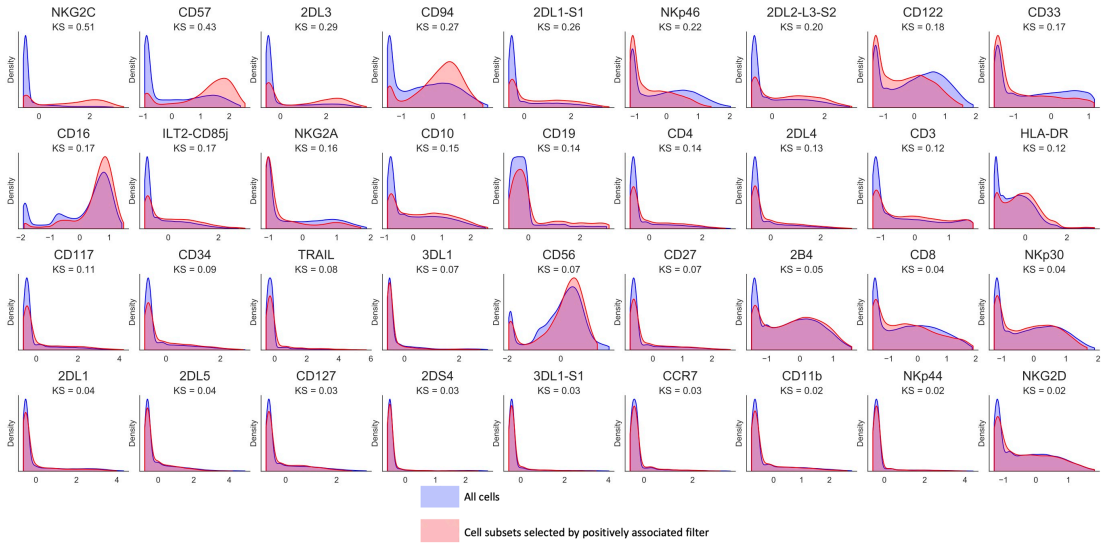


(a) CellCnn **(b)** PRICELL simulation

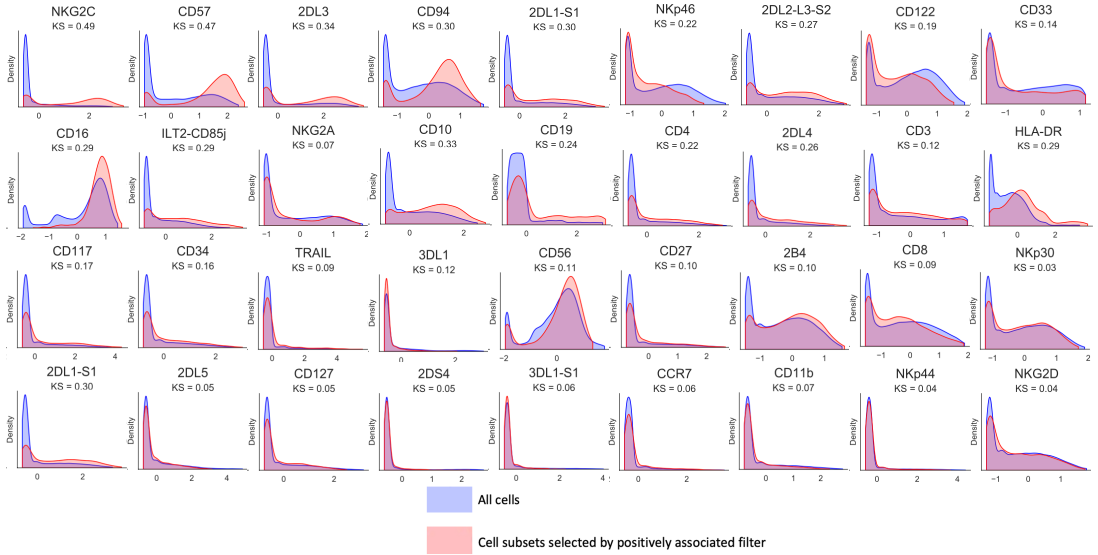
Figure E.2: *Comparison of the selected cell population frequencies from the test samples of the CMV- and CMV+ classes by using the positively associated filter learned by (a) CellCnn original architecture, and by (b)* PRICELL*'s adapted architecture for encrypted training.*

Setting/Metrics	Accuracy	Precision	Recall	F-score
RRMS (multi-cell / phenotype classification)				
CellCnn	0.65 / 0.67	0.62 / 0.71	0.61 / 0.71	0.66 / 0.71
Local (N=2)	0.59 / 0.62	0.59 / 0.68	0.62 / 0.64	0.59 / 0.66
PriCell (N=2)	0.65 / 0.67	0.66 / 0.75	0.62 / 0.64	0.64 / 0.67
Local (N=4)	0.55 / 0.55	0.57 / 0.61	0.60 / 0.61	0.58 / 0.62
PriCell (N=4)	0.64 / 0.67	0.63 / 0.73	0.61 / 0.64	0.61 / 0.69
Local (N=6)	0.53 / 0.52	0.53 / 0.58	0.55 / 0.54	0.53 / 0.57
PriCell (N=6)	0.64 / 0.67	0.68 / 0.80	0.61 / 0.64	0.63 / 0.69
NIND (multi-cell / phenotype classification)				
CellCnn	0.72 / 0.75	0.82 / 0.83	0.75 / 0.71	0.76 / 0.77
Local (N=2)	0.57 / 0.58	0.65 / 0.65	0.59 / 0.52	0.62 / 0.63
PriCell (N=2)	0.72 / 0.75	0.73 / 0.75	0.80 / 0.86	0.78 / 0.80
Local (N=4)	0.55 / 0.55	0.66 / 0.68	0.51 / 0.50	0.59 / 0.58
PriCell (N=4)	0.72 / 0.71	0.75 / 0.73	0.84 / 0.71	0.78 / 0.75
Local (N=6)	0.53 / 0.52	0.63 / 0.64	0.57 / 0.56	0.59 / 0.57
PriCell (N=6)	0.71 / 0.71	0.73 / 0.75	0.76 / 0.71	0.75 / 0.75
CMV (multi-cell / phenotype classification)				
CellCnn	0.80 / 0.75	0.72 / 0.58	0.98 / 1.00	0.83 / 0.73
Local (N=2)	0.54 / 0.58	0.52 / 0.42	0.50 / 0.50	0.52 / 0.47
PriCell (N=2)	0.79 / 0.75	0.71 / 0.58	0.98 / 1.00	0.83 / 0.73
Local (N=3)	0.59 / 0.55	0.56 / 0.40	0.64 / 0.67	0.60 / 0.49
PriCell (N=3)	0.79 / 0.75	0.76 / 0.58	0.84 / 1.00	0.80 / 0.73
Local (N=5)	0.50 / 0.52	0.44 / 0.31	0.57 / 0.55	0.50 / 0.39
PriCell (N=5)	0.78 / 0.75	0.69 / 0.58	0.97 / 1.00	0.82 / 0.73
AML (multi-cell / phenotype classification)				
CellCnn	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00
Local (N=2)	0.98 / 1.00	0.98 / 1.00	0.98 / 1.00	0.98 / 1.00
PriCell (N=2)	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00	1.00 / 1.00

Table E.1: *Classification performance (accuracy, precision, recall, and F-score) of the models obtained with* PriCell*, original CellCnn, and local training without collaboration for RRMS, NIND, CMV, and AML classification tasks. All models are tested on two datasets for multi-cell and phenotype classification respectively, separated with '/'.*



(a) CellCnn



(b) PRICELL simulation

Figure E.3: *Comparison of the histograms of univariate z-transformed marker expression profiles for all cells and for the cell population selected by the positively associated filter learned by (a) CellCnn original architecture, and by (b)* PRICELL*'s adapted architecture for encrypted training on CMV dataset. The distributions show that* PRICELL *training does not affect the findings of the non-privacy preserving training.*

Bibliography

[1] Microsoft SEAL (release 3.3). https://github.com/Microsoft/SEAL. (Accessed: 2022-01-06).

[2] Art. 25 GDPR Data protection by design and by default. https://gdpr-info.eu/art-25-gdpr/. (Accessed: 2023-02-02).

[3] 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).

[4] Convolutional Neural Networks. https://cs231n.github.io/convolutional-networks/. (Accessed: 2022-01-06).

[5] Cothority network library. https://github.com/dedis/onet. (Accessed: 2022-01-06).

[6] Data61. MP-SPDZ - Versatile framework for multi-party computation. https://github.com/data61/MP-SPDZ. (Accessed: 2022-01-06).

[7] Go Programming Language. https://golang.org. (Accessed: 2022-01-06).

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

[9] IDASH PRIVACY & SECURITY WORKSHOP 2021 - Secure Genome Analysis Competition). http://www.humangenomeprivacy.org/2021/, 2021. (Accessed: 2023-01-01).

[10] CSAW Applied Research Competition). https://www.csaw.io/research, 2021. (Accessed: 2023-01-01).

[11] The 21st Privacy Enhancing Technologies Symposium. https://petsymposium.org/index.php, 2021. (Accessed: 2022-01-06).

[12] The Network and Distributed System Security Symposium (NDSS). https://www.ndss-symposium.org/ndss2021/, 2021. (Accessed: 2022-01-06).

[13] Patterns, Volume 3, Issue 5. https://www.cell.com/patterns/issue?pii=S2666-3899(21)X0006-2, 2022. (Accessed: 2023-01-01).

[14] PriCell. https://doi.org/10.5281/zenodo.6330988, 2023. (Accessed: 2023-01-01).

[15] PriCell (Privacy-Preserving CellCNN). https://github.com/ldsec/cellCNN, 2023. (Accessed: 2023-01-01).

[16] Tune Insight. Orchestrating secure data collaborations. https://tuneinsight.com/, 2023. (Accessed: 2023-01-01).

[17] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[18] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[19] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad. State-of-the-art in artificial neural network applications: A survey. *Elsevier Heliyon*, 4(11):e00938, 2018.

[20] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Comput. Surv.*, 51(4), July 2018.

[21] M. Adnan, S. Kalra, J. C. Cresswell, G. W. Taylor, and H. R. Tizhoosh. Federated learning and differential privacy for medical image analysis. *Scientific Reports*, 12(1):1953, Feb 2022.

[22] N. Agrawal, A. S. Shamsabadi, M. J. Kusner, and A. Gascón. QUOTIENT: Two-party secure neural network training and prediction. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[23] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkowich, D. Murik, H. Shaul, and O. Soceanu. Helayers: A tile tensors framework for large neural networks on encrypted data. *CoRR*, abs/2011.01805, 2020.

[24] E. Aharoni, N. Drucker, G. Ezov, H. Shaul, and O. Soceanu. Complex encoded tile tensors: Accelerating encrypted analytics. *IEEE Security & Privacy*, 20(5):35–43, 2022.

[25] A. Akavia, H. Shaul, M. Weiss, and Z. Yakhini. Linear-regression on packed encrypted data in the two-server model. In *ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC)*, 2019.

[26] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[27] M. Albrecht et al. Homomorphic Encryption Security Standard. Technical report, HomomorphicEncryption.org, November 2018.

[28] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9:169 – 203, 2015.

[29] S. V. Algesheimer J., Camenisch J. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *Yung M. (eds) Advances in Cryptology — CRYPTO 2002*, pages 417–432, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[30] D. Alistarh, Z. Allen-Zhu, and J. Li. Byzantine stochastic gradient descent. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[31] Y. Aono, T. Hayashi, L. Trieu Phong, and L. Wang. Scalable and secure logistic regression via homomorphic encryption. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, 2016.

[32] E. Arvaniti and M. Claassen. Sensitive detection of rare disease-associated cell subsets via representation learning. *Nature Communications*, 8(1):14825, Apr 2017.

[33] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *Advances in Cryptology – EUROCRYPT 2012*, pages 483–501, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[34] A. A. Badawi, L. Hoang, C. F. Mun, K. Laine, and K. M. M. Aung. Privft: Private and fast text classification with homomorphic encryption. *CoRR*, abs:1908.06972, 2019.

[35] M. Bakshi and M. Last. Cryptornn - privacy-preserving recurrent neural networks using homomorphic encryption. In S. Dolev, V. Kolesnikov, S. Lodha, and G. Weiss, editors, *Cyber Security Cryptography and Machine Learning (CSCML)*, pages 245–253, Cham, 2020. Springer International Publishing.

[36] M. Barni, P. Failla, R. Lazzeretti, A.-R. Sadeghi, and T. Schneider. Privacy-preserving ecg classification with branching programs and neural networks. *IEEE Transactions on Information Forensics and Security (TIFS)*, 6(2):452–468, 2011.

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

[38] B. Berger and H. Cho. Emerging technologies towards enhancing privacy in genomic data sharing. *Genome Biology*, 20(1):128, Jul 2019.

[39] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski. ngraph-he2: A high-throughput framework for neural network inference on encrypted data. In *ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC)*, 2019.

[40] F. Boemer, Y. Lao, and C. Wierzynski. ngraph-he: A graph compiler for deep learning on homomorphically encrypted data. *CoRR*, abs/1810.10121, 2018.

[41] D. Bogdanov, L. Kamm, S. Laur, and V. Sokk. Rmind: a tool for cryptographically secure statistical analysis. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 15(3):481–495, 2018.

[42] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for federated learning on user-held data. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[43] L. Bonomi, X. Jiang, and L. Ohno-Machado. Protecting patient privacy in survival analyses. *Journal of the American Medical Informatics Association : JAMIA*, 27, 11 2019.

[44] C. Bonte and F. Vercauteren. Privacy-preserving logistic regression training. *BMC Medical Genomics*, 11, 2018.

[45] C. Bonte and F. Vercauteren. Privacy-preserving logistic regression training. *BMC Medical Genomics*, 11, 10 2018.

[46] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In A. Canteaut and F.-X. Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 587–617, Cham, 2021. Springer International Publishing.

[47] A. Boulemtafes, A. Derhab, and Y. Challal. A review of privacy-preserving techniques for deep learning. *Neurocomputing*, 384:21–45, 2020.

[48] C. Boura, N. Gama, and M. Georgieva. Chimera: a unified framework for b/fv, tfhe and heaan fully homomorphic encryption and predictions for deep learning. *IACR Cryptol. ePrint Arch.*, 2018:758, 2018.

[49] J. W. S. Brown, O. Ohrimenko, and R. Tamassia. Haze: Privacy-preserving real-time traffic statistics. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, page 540–543, New York, NY, USA, 2013. Association for Computing Machinery.

[50] A. Brutzkus, O. Elisha, and R. Gilad-Bachrach. Low latency privacy preserving inference, 2018.

[51] E. Buchmann, K. Böhm, T. Burghardt, and S. Kessler. Re-identification of smart meter data. *Personal and Ubiquitous Computing*, 17(4):653–662, Apr 2013.

[52] P. Bunn and R. Ostrovsky. Secure two-party k-means clustering. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 486–497, New York, NY, USA, 2007. ACM.

[53] M. Byali, H. Chaudhari, A. Patra, and A. Suresh. FLASH: Fast and robust framework for privacy-preserving machine learning. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2020:459–480, 2020.

[54] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptol. ePrint Arch.*, 2017:35, 2017.

[55] S. Chandar, C. Sankar, E. Vorontsov, S. E. Kahou, and Y. Bengio. Towards non-saturating recurrent units for modelling long-term dependencies. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, 2019.

[56] S. Chatel, A. Pyrgelis, J. R. Troncoso-Pastoriza, and J.-P. Hubaux. Privacy and integrity preserving computations with crisp. In *USENIX Security Symposium*, pages 2111–2128, 2021.

[57] H. Chaudhari, R. Rachuri, and A. Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. In *27th Annual Network and Distributed System Security Symposium, NDSS*, pages 23–26, 2020.

[58] J. Chen, M. Edupalli, B. Berger, and H. Cho. Secure and federated linear mixed model association tests. *bioRxiv*, 2022.

[59] M. Chen, R. Mathews, T. Ouyang, and F. Beaufays. Federated learning of out-of-vocabulary words. *CoRR*, abs/1903.10635, 2019.

[60] T. Chen and S. Zhong. Privacy-preserving backpropagation neural network learning. *IEEE Transactions on Neural Networks*, 20(10):1554–1564, Oct 2009.

[61] W. Chen, K. Sotiraki, I. Chang, M. Kantarcioglu, and R. A. Popa. HOLMES: A platform for detecting malicious inputs in secure collaborative computation, 2021.

[62] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Springer International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2017.

[63] J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee. Numerical method for comparison on homomorphically encrypted numbers. In S. D. Galbraith and S. Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 415–445, Cham, 2019. Springer International Publishing.

[64] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Tfhe: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, Jan 2020.

[65] I. Chillotti, M. Joye, and P. Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. Cryptology ePrint Archive, Paper 2021/091, 2021. https://eprint.iacr.org/2021/091.

[66] H. Cho, D. Froelicher, J. Chen, M. Edupalli, A. Pyrgelis, J. R. Troncoso-Pastoriza, J.-P. Hubaux, and B. Berger. Secure and federated genome-wide association studies for biobank-scale datasets. *bioRxiv*, 2022.

[67] H. Cho, D. Wu, and B. Berger. Secure genome-wide association analysis using multiparty computation. *Nature Biotechnology*, 36:547–551, 2018.

[68] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111. Association for Computational Linguistics, Oct. 2014.

[69] C. A. Choquette-Choo, N. Dullerud, A. Dziedzic, Y. Zhang, S. Jha, N. Papernot, and X. Wang. Capc learning: Confidential and private collaborative learning. In *9th International Conference on Learning Representations, ICLR 2021,Virtual Event, Austria, May 3-7, 2021*, 2021.

[70] O. Choudhury, A. Gkoulalas-Divanis, T. Salonidis, I. Sylla, Y. Park, G. Hsu, and A. Das. *Differential Privacy-enabled Federated Learning for Sensitive Health Data.* 2019.

[71] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. yu, G. Bradski, A. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. volume 19, pages 281–288, 01 2006.

[72] V. N. Chunekar and H. P. Ambulgekar. 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*, pages 893–895, 2009.

[73] A. E. Cicek, M. E. Nergiz, and Y. Saygin. Ensuring location diversity in privacy-preserving spatio-temporal data publishing. *The VLDB Journal*, 23(4):609–625, 2014.

[74] S. Constable, Y. Tang, S. Wang, X. Jiang, and S. Chapin. Privacy-preserving gwas analysis on federated genomic datasets. *BMC Medical Informatics and Decision Making*, 15:S2, 2015.

[75] H. Corrigan-Gibbs and D. Boneh. Prio: Private, Robust, and Computation of Aggregate Statistics. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[76] J. L. Crawford, C. Gentry, S. Halevi, D. Platt, and V. Shoup. Doing real work with fhe: The case of logistic regression. In *ACM WAHC*, 2018.

[77] A. Dalskov, D. Escudero, and M. Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2020(4):355 – 375, 2020.

[78] Y. Dandi, L. Barba, and M. Jaggi. Implicit gradient alignment in distributed and federated learning. AAAI Conference on Artificial Intelligence, pages 6454–6462, Palo Alto, 2022. ASSOC ADVANCEMENT ARTIFICIAL INTELLIGENCE.

[79] I. Dataset. Oecd data. https://data.oecd.org/price/inflation-cpi.htm#indicator-chart, 2023. (Accessed: 2023-03-03).

[80] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz. Chet: An optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 142–156, New York, NY, USA, 2019. Association for Computing Machinery.

[81] E. Dauterman, M. Rathee, R. Popa, and I. Stoica. Waldo: A private time-series database from function secret sharing. In *2022 2022 IEEE Symposium on Security and Privacy (S&P)*, pages 668–686, Los Alamitos, CA, USA, may 2022. IEEE Computer Society.

[82] M. De Cock, R. Dowsley, A. Nascimento, D. Reich, and A. Todoki. Privacy-preserving classification of personal text messages with secure multi-party computation: An application to hate-speech detection, 06 2019.

[83] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng. Large scale distributed deep networks. In *NIPS*. 2012.

[84] D. Delen and R. Sharda. Artificial neural networks in decision support systems. *Springer Handbook on Decision Support Systems*, 2008.

[85] J. Deng, Y. Wang, J. Li, C. Wang, C. Shang, H. Liu, S. Rajasekaran, and C. Ding. TAG: Gradient attack on transformer-based language models. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3600–3610, Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics.

[86] R. Dey and F. M. Salem. Gate-variants of gated recurrent unit (GRU) neural networks. In *MWSCAS*, pages 1597–1600, 2017.

[87] D. I. Dimitrov, M. Balunović, N. Jovanović, and M. Vechev. Lamp: Extracting text from gradients with language model priors. *CoRR*, abs/2202.08827, 2022.

[88] V. D'Amato, S. Levantesi, and G. Piscopo. Deep learning in predicting cryptocurrency volatility. *Physica A: Statistical Mechanics and its Applications*, 596:127158, 2022.

[89] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 1985.

[90] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[91] Epileptic Seizure Recognition Dataset. https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition. (Accessed: 2022-01-06).

[92] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[93] M. N. Fekri, K. Grolinger, and S. Mir. Distributed load forecasting using smart meter data: Federated learning with recurrent neural networks. *International Journal of Electrical Power & Energy Systems*, 137:107669, 2022.

[94] B. Feng, Q. Lou, L. Jiang, and G. C. Fox. Cryptogru: Low latency privacy-preserving text analysis with gru. *CoRR*, abs/2010.11796, 2021.

[95] Q. Feng, D. He, Z. Liu, H. Wang, and K.-K. R. Choo. Securenlp: A system for multi-party privacy-preserving natural language processing. *IEEE Transactions on Information Forensics and Security (TIFS)*, 15:3709–3721, 2020.

[96] M. Finck and F. Pallas. They who must not be identified—distinguishing personal from non-personal data under the GDPR. *International Data Privacy Law*, 10(1):11–36, 03 2020.

[97] M. Franklin and S. Haber. Joint encryption and message-efficient secure computation. *Journal of Cryptology*, 9(4):217–232, Sep 1996.

[98] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM CCS*, 2015.

[99] D. Froelicher, H. Cho, M. Edupalli, J. S. Sousa, J. Bossuat, A. Pyrgelis, J. R. Troncoso-Pastoriza, B. Berger, and J. Hubaux. Scalable and privacy-preserving federated principal component analysis. In *2023 2023 IEEE Symposium on Security and Privacy (SP)*, pages 888–905, Los Alamitos, CA, USA, may 2023. IEEE Computer Society.

[100] D. Froelicher, P. Egger, J. Sousa, J. L. Raisaro, Z. Huang, C. Mouchet, B. Ford, and J.-P. Hubaux. Unlynx: A decentralized system for privacy-conscious data sharing. *PETS*, 2017.

[101] D. Froelicher, J. R. Troncoso-Pastoriza, A. Pyrgelis, S. Sav, J. A. Gomes de Sá e Sousa, J.-P. Hubaux, and J.-P. Bossuat. System and method for privacy-preserving distributed training of machine learning models on distributed datasets, 2021.

[102] D. Froelicher, J. R. Troncoso-Pastoriza, A. Pyrgelis, S. Sav, J. S. Sousa, J.-P. Bossuat, and J.-P. Hubaux. Scalable privacy-preserving distributed learning. *PETS*, 2021.

[103] D. Froelicher, J. R. Troncoso-Pastoriza, J. L. Raisaro, M. A. Cuendet, J. S. Sousa, H. Cho, B. Berger, J. Fellay, and J.-P. Hubaux. Truly privacy-preserving federated analytics for precision medicine with multiparty homomorphic encryption. *Nature Communications*, 12(1):5910, Oct 2021.

[104] D. Froelicher, J. R. Troncoso-Pastoriza, J. S. Sousa, and J. Hubaux. Drynx: Decentralized, secure, verifiable system for statistical queries and machine learning on distributed datasets. *IEEE Transactions on Information Forensics and Security (TIFS)*, 15:3035–3050, 2020.

[105] D. J. Froelicher. Privacy-preserving federated analytics using multiparty homomorphic encryption. page 179, 2021.

[106] E. Galli, F. J. Hartmann, B. Schreiner, F. Ingelfinger, E. Arvaniti, M. Diebold, D. Mrdjen, F. van der Meer, C. Krieg, F. A. Nimer, N. S. R. Sanderson, C. Stadelmann, M. Khademi, F. Piehl, M. Claassen, T. Derfuss, T. P. Olsson, and B. Becher. GM-CSF and CXCR4 define a t helper cell signature in multiple sclerosis. *Nature medicine*, 25:1290 – 1300, 2019.

[107] N. Gandhi, S. Mishra, S. K. Bharti, and K. Bhagat. Leveraging towards privacy-preserving using federated machine learning for healthcare systems. In *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages 1–6, 2021.

[108] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Privacy Enhancing Technologies (PETS)*, 2017.

[109] A. Gaye, Y. Marcon, J. Kutschke, P. Laflamme, A. Turner, E. Jones, J. Minion, A. Boyd, C. Newby, M.-L. Nuotio, R. Wilson, O. Butters, B. Murtagh, I. Demir, D. Doiron, L. Giepmans, S. Wallace, I. Budin-Ljøsne, C. Schmidt, and P. Burton. Datashield: Taking the analysis to the data, not the data to the analysis. *International Journal of Epidemiology*, page dyu188, 2014.

[110] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller. Inverting gradients - how easy is it to break privacy in federated learning? In *Advances in Neural Information Processing Systems*, NeurIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[111] I. Giacomelli, S. Jha, M. Joye, C. D. Page, and K. Yoon. Privacy-preserving ridge regression with only linearly-homomorphic encryption. In *Springer International Conference on Applied Cryptography and Network Security (ACNS)*, 2018.

[112] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *ICML*, 2016.

[113] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[114] L. Gomes. Quantum computing: Both here and not here. *IEEE Spectrum*, 55(4):42–47, 2018.

[115] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[116] V. Goyal, A. Polychroniadou, and Y. Song. Sharing transformation and dishonest majority mpc with packed secret sharing. Cryptology ePrint Archive, Paper 2022/831, 2022. https://eprint.iacr.org/2022/831.

[117] D. Groos and E.-B. van Veen. Anonymised data and the rule of law. *European Data Protection Law Review*, 2020.

[118] S. Halevi and V. Shoup. HElib - An Implementation of homomorphic encryption. https://github.com/shaih/HElib/. (Accessed: 2022-01-06).

[119] S. Halevi and V. Shoup. Algorithms in helib. In *Annual International Cryptology Conference (CRYPTO)*. Springer, 2014.

[120] S. Halevi and V. Shoup. Faster homomorphic linear transformations in helib. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 93–120, Cham, 2018. Springer International Publishing.

[121] A. Hard, K. Rao, R. Mathews, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage. Federated learning for mobile keyboard prediction. *CoRR*, abs/1811.03604, 2018.

[122] L. Hardinata, B. Warsito, and Suparti. Bankruptcy prediction based on financial ratios using jordan recurrent neural networks: a case study in polish companies. *Journal of Physics: Conference Series*, 1025, 2018.

[123] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. pages 1026–1034, 2015.

[124] J. C. Heck and F. M. Salem. Simplified minimal gated unit variations for recurrent neural networks. In *MWSCAS*, pages 1593–1596, 2017.

[125] M. Henaff, A. Szlam, and Y. LeCun. Recurrent orthogonal networks and long-memory tasks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 2034–2042. JMLR.org, 2016.

[126] E. Hesamifard, H. Takabi, M. Ghasemi, and R. Wright. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2018:123–142, 06 2018.

[127] B. Hie, H. Cho, and B. Berger. Realizing private and practical pharmacological collaboration. *Science*, 362(6412):347–350, 2018.

[128] B. Hitaj, G. Ateniese, and F. Perez-Cruz. Deep models under the GAN: Information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 603–618, New York, NY, USA, 2017. Association for Computing Machinery.

[129] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[130] S.-K. Hong, K. Gurjar, H.-S. Kim, and Y.-S. Moon. A survey on privacy preserving time series data mining. In *3rd International Conference on Intelligent Computational Systems*, 2013.

[131] A. Horowitz, D. Strauss-Albee, M. Leipold, J. Kubo, N. Nemat-Gorgani, O. Dogan, C. Dekker, S. Mackey, H. Maecker, G. Swan, M. Davis, P. Norman, L. Guethlein, M. Desai, P. Parham, and C. Blish. Genetic and environmental determinants of human nk cell diversity revealed by mass cytometry. *Science translational medicine*, 5:208ra145, 2013.

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

[133] C. Huang, J. Huang, and X. Liu. Cross-silo federated learning: Challenges and opportunities. *CoRR*, abs/2206.12949, 2022.

[134] Z. Huang, W. jie Lu, C. Hong, and J. Ding. Cheetah: Lean and fast secure Two-Party deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 809–826, Boston, MA, Aug. 2022. USENIX Association.

[135] Z. Huang, G. Zweig, M. Levit, B. Dumoulin, B. Oguz, and S. Chang. Accelerating recurrent neural network training via two stage classes and parallelization. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 326–331, 2013.

[136] F. Intoci, S. Sav, A. Pyrgelis, J.-P. Bossuat, J. R. Troncoso-Pastoriza, and J.-P. Hubaux. slytHErin: an agile framework for encrypted deep neural network inference. *CoRR*, abs/2305.00690, 2023.

[137] G. Iuhasz, M. Tirea, and V. Negru. Neural network predictions of stock price fluctuations. pages 505–512, 09 2012.

[138] K. A. Jagadeesh, D. J. Wu, J. A. Birgmeier, D. Boneh, and G. Bejerano. Deriving genomic diagnoses without revealing patient genomes. *Science*, 357(6352):692–695, 2017.

[139] G. Jagannathan and R. N. Wright. Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 593–599, New York, NY, USA, 2005. ACM.

[140] J. Jang, Y. Lee, A. Kim, B. Na, D. Yhee, B. Lee, J. H. Cheon, and S. Yoon. Privacy-preserving deep sequential model with matrix homomorphic encryption. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, page 377–391, New York, NY, USA, 2022. Association for Computing Machinery.

[141] B. Jayaraman and D. Evans. Evaluating differentially private machine learning in practice. In *USENIX Security*, 2019.

[142] B. Jayaraman, L. Wang, D. Evans, and Q. Gu. Distributed learning without distress: Privacy-preserving empirical risk minimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[143] X. Jiang, M. Kim, K. Lauter, and Y. Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1209–1222, New York, NY, USA, 2018. ACM.

[144] Y. Jiang, J. Hamer, C. Wang, X. Jiang, M. Kim, Y. Song, Y. Xia, N. Mohammed, M. N. Sadat, and S. Wang. Securelr: Secure logistic regression model via a hybrid cryptographic protocol. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 2019.

[145] X. Jin, P.-Y. Chen, C.-Y. Hsu, C.-M. Yu, and T. Chen. Cafe: Catastrophic data leakage in vertical federated learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 994–1006. Curran Associates, Inc., 2021.

[146] M. I. Jordan. Chapter 25 - serial order: A parallel distributed processing approach. In J. W. Donahoe and V. Packard Dorsel, editors, *Neural-Network Models of Cognition*, volume 121 of *Advances in Psychology*, pages 471–495. North-Holland, 1997.

[147] N. P. Jouppi et al. In-datacenter performance analysis of a tensor processing unit. *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12, 2017.

[148] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. Gazelle: A low latency framework for secure neural network inference. *USENIX Security*, 2018.

[149] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021.

[150] G. Kaissis, M. Makowski, D. Rückert, and R. Braren. Secure, privacy-preserving and federated machine learning in medical imaging. *Nature Machine Intelligence*, 2, 2020.

[151] L. Kamm, D. Bogdanov, S. Laur, and J. Vilo. A new way to protect privacy in large-scale genome-wide association studies. *Bioinformatics (Oxford, England)*, 29, 03 2013.

[152] B. Kanuparthi, D. Arpit, G. Kerg, N. Ke, I. Mitliagkas, and Y. Bengio. h-detach: Modifying the lstm gradient towards better optimization. In *7th International Conference for Learning Representations (ICLR)*, 2019.

[153] S. P. Karimireddy, L. He, and M. Jaggi. Learning from history for byzantine robust optimization. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5311–5319. PMLR, 18–24 Jul 2021.

[154] S. P. Karimireddy, L. He, and M. Jaggi. Byzantine-robust learning on heterogeneous datasets via bucketing. In *International Conference on Learning Representations*, 2022.

[155] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5132–5143. PMLR, 13–18 Jul 2020.

[156] M. Keller and K. Sun. Secure quantized training for deep learning. In *International Conference on Machine Learning*, pages 10912–10938. PMLR, 2022.

[157] Why we shouldn't disregard the NDA. https://www.keystonelaw.com/keynotes/why-we-shouldnt-disregard-the-nda. (Accessed: 2022-01-06).

[158] V. Khomenko, O. Shyshkov, O. Radyvonenko, and K. Bokhan. 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)*, pages 100–103, 2016.

[159] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 2018.

[160] M. Kim, X. Jiang, K. Lauter, E. Ismayilzada, and S. Shams. Secure human action recognition by encrypted neural network inference. *Nature Communications*, 13(1):4799, Aug 2022.

[161] M. Kim, J. Lee, L. Ohno-Machado, and X. Jiang. Secure and differentially private logistic regression for horizontally distributed data. *IEEE Transactions on Information Forensics and Security*, 15:695–710, 2020.

[162] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang. Secure logistic regression based on homomorphic encryption: Design and evaluation. *JMIR Medical Informatics*, 6(2):e19, 2018.

[163] A. Kipnis and E. Hibshoosh. Efficient methods for practical fully homomorphic symmetric-key encrypton, randomization and verification. *IACR Cryptol. ePrint Arch.*, 2012.

[164] S. Kirby, P. Eng, W. Danter, C. George, T. Francovic, R. R. Ruby, and K. Ferguson. Neural network prediction of obstructive sleep apnea from clinical criteria. *Chest*, 116 2:409–15, 1999.

[165] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten. Crypten: Secure multi-party computation meets machine learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *NeurIPS*, volume 34, pages 4961–4973. Curran Associates, Inc., 2021.

[166] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *CoRR*, abs:1610.02527, 2016.

[167] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016.

[168] A. Krizhevsky. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto*, 2012.

[169] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. *PETS*, 2011.

[170] D. Laehnemann et al. 12 grand challenges in single-cell data science. *Genome Biology*, 21:31, 2020.

[171] Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *ArXiv*, abs/1504.00941, 2015.

[172] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[173] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 12403–12422. PMLR, 17–23 Jul 2022.

[174] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.

[175] J. Levine, E. Simonds, S. Bendall, K. Davis, E.-A. Amir, M. Tadmor, O. Litvin, H. Fienberg, A. Jager, E. Zunder, R. Finck, A. Gedman, I. Radtke, J. Downing, D. Pe'er, and G. Nolan. Data-driven phenotypic dissection of aml reveals progenitor-like cells that correlate with prognosis. *Cell*, 162, 2015.

[176] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen. Learning to detect malicious clients for robust federated learning. *CoRR*, abs/2002.00211, 2020.

[177] W. Li, F. Milletarì, D. Xu, N. Rieke, J. Hancox, W. Zhu, M. Baust, Y. Cheng, S. Ourselin, M. J. Cardoso, and A. Feng. Privacy-preserving federated brain tumour segmentation. In H.-I. Suk, M. Liu, P. Yan, and C. Lian, editors, *International Workshop in Machine Learning in Medical Imaging (MLMI)*. Springer, 2019.

[178] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang. On the convergence of FedAvg on non-IID data. In *ICLR*, 2020.

[179] X. Lian, W. Zhang, C. Zhang, and J. Liu. Asynchronous decentralized parallel stochastic gradient descent. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 3043–3052, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

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

[181] R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In A. Kiayias, editor, *Springer Topics in Cryptology*, pages 319–339, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[182] Why NDAs often don't work when expected to do so and what to do about it. https://www.linkedin.com/pulse/why-ndas-often-dont-work-when-expected-do-so-what-martin-schweiger. (Accessed: 2022-01-06).

[183] P. Lisboa. A review of evidence of health benefit from artificial neural networks in medical intervention. *Elsevier Neural networks*, 2002.

[184] J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 619–631, New York, NY, USA, 2017. ACM.

[185] X. Liu and X. Yi. Privacy-preserving collaborative medical time series analysis based on dynamic time warping. In K. Sako, S. Schneider, and P. Y. A. Ryan, editors, *Computer Security – ESORICS 2019*, pages 439–460, Cham, 2019. Springer International Publishing.

[186] X. Liu, Y. Zheng, X. Yi, and S. Nepal. Privacy-preserving collaborative analytics on medical time series data. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2020.

[187] Y. Liu, J. J. Q. Yu, J. Kang, D. Niyato, and S. Zhang. Privacy-preserving traffic flow prediction: A federated learning approach. *IEEE Internet of Things Journal*, 7(8):7751–7763, 2020.

[188] G. Lloret-Talavera, M. Jorda, H. Servat, F. Boemer, C. Chauhan, S. Tomishima, N. Shah, and A. Peña. Enabling homomorphically encrypted inference for large dnn models. *IEEE Transactions on Computers*, PP:1–1, 04 2021.

[189] Y. Long, V. Bindschaedler, and C. A. Gunter. Towards measuring membership privacy. *CoRR*, abs/1712.09136, 2017.

[190] Y. Long, B. Wang, Z. Yang, B. Kailkhura, A. Zhang, C. A. Gunter, and B. Li. G-pate: Scalable differentially private data generator via private aggregation of teacher discriminators. *CoRR*, abs/1906.09338, 2021.

[191] A. López-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 1219–1234. Association for Computing Machinery, 2012.

[192] W.-j. Lu and J. Sakuma. More practical privacy-preserving machine learning as a service via efficient secure matrix multiplication. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '18, page 25–36, New York, NY, USA, 2018. Association for Computing Machinery.

[193] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 1–23, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[194] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In A. Singh and J. Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[195] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang. Learning differentially private recurrent language models. *CoRR*, abs/1710.06963, 2018.

[196] Top 15 deep learning applications that will rule the world in 2018 and beyond. https://medium.com/breathe-publication/top-15-deep-learning-applications-that-will-rule-the-world-in-2018-and-beyond-7c6130c43b01. (Accessed: 2022-01-06).

[197] S. Meftah, B. H. M. Tan, C. F. Mun, K. M. M. Aung, B. Veeravalli, and V. Chandrasekhar. Doren: Toward efficient deep convolutional neural networks with fully homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 16:3740–3752, 2021.

[198] L. Melis, G. Danezis, and E. De Cristofaro. Efficient private statistics with succinct sketches. In *Network and Distributed System Security Symposium (NDSS)*, 2016.

[199] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (S&P)*, pages 691–706, 2019.

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

[201] Mininet. http://mininet.org. (Accessed: 2022-01-06).

[202] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, Boston, MA, Aug. 2020. USENIX Association.

[203] P. K. Mishra, D. Rathee, D. H. Duong, and M. Yasuda. Fast secure matrix multiplications over ring-based homomorphic encryption. *Information Security Journal: A Global Perspective*, 30(4):219–234, 2021.

[204] P. Mohassel and P. Rindal. ABY3: a mixed protocol framework for machine learning. In *ACM Conference on Computer and Communications Security (CCS)*, 2018.

[205] P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (S&P)*, pages 19–38, May 2017.

[206] A. Molina-Markham, P. Shenoy, K. Fu, E. Cecchet, and D. Irwin. Private memoirs of a smart meter. In *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, BuildSys '10, page 61–66, New York, NY, USA, 2010. Association for Computing Machinery.

[207] S. Moshiri, N. E. Cameron, and D. Scuse. Static, dynamic, and hybrid neural networks in forecasting inflation. *Computational Economics*, 14(3):219–235, Dec 1999.

[208] C. Mouchet, E. Bertrand, and J.-P. Hubaux. An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. Cryptology ePrint Archive, Paper 2022/780, 2022. https://eprint.iacr.org/2022/780.

[209] C. Mouchet, J. R. Troncoso-pastoriza, J.-P. Bossuat, and J. P. Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *PETS*, 2021.

[210] T. Nagalakshmi, M. Govindarajan, and M. Ramalingam. An intelligent breast cancer forecasting system using optimized elman deep neural network. In *2022 3rd International Conference on Smart Electronics and Communication (ICOSEC)*, pages 1107–1114, 2022.

[211] K. Nandakumar, N. Ratha, S. Pankanti, and S. Halevi. Towards deep neural network training on encrypted data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[212] M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE Symposium on Security and Privacy (S&P)*, pages 739–753, 2019.

[213] C. Neill et al. A blueprint for demonstrating quantum supremacy with superconducting qubits. *Science*, 2018.

[214] Y. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

[215] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. *NIPS*, 2011.

[216] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[217] A.-M. Olteanu, K. Huguenin, R. Shokri, M. Humbert, and J.-P. Hubaux. Quantifying interdependent privacy risks with location data. *IEEE Transactions on Mobile Computing*, 16(3):829–842, 2017.

[218] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Springer International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 1999.

[219] N. Papernot, M. Abadi, Úlfar Erlingsson, I. Goodfellow, and K. Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *CoRR*, abs/1610.05755, 2017.

[220] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and U. Erlingsson. Scalable private learning with pate. In *ICLR*, 02 2018.

[221] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page III–1310–III–1318. JMLR.org, 2013.

[222] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[223] A. Patra and A. Suresh. Blaze: Blazing fast privacy-preserving machine learning. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

[224] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. Privacy-preserving deep learning: Revisited and enhanced. In L. Batten, D. S. Kim, X. Zhang, and G. Li, editors, *Applications and Techniques in Information Security*, pages 100–110, Singapore, 2017. Springer Singapore.

[225] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security (TIFS)*, 13(5):1333–1345, 2018.

[226] L. Prechelt. Early stopping - but when? In G. B. Orr and K.-R. Müller, editors, *Springer Neural Networks: Tricks of the Trade*, pages 55–69, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[227] A. Pyrgelis, E. De Cristofaro, and G. J. Ross. Privacy-friendly mobility analytics using aggregate location data. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPACIAL '16, New York, NY, USA, 2016. Association for Computing Machinery.

[228] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[229] M. A. Rahman, T. Rahman, R. Laganière, and N. Mohammed. Membership inference attack against differentially private deep learning model. *Transactions on Data Privacy*, 11:61–79, 2018.

[230] J. L. Raisaro, J. R. Troncoso-Pastoriza, M. Misbach, J. S. Sousa, S. Pradervand, E. Missiaglia, O. Michielin, B. Ford, and J.-P. Hubaux. Medco: Enabling secure and privacy-preserving exploration of distributed clinical and genomic data. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 16(4):1328–1341, July 2019.

[231] R. Ran, W. Wang, Q. Gang, J. Yin, N. Xu, and W. Wen. CryptoGCN: Fast and scalable homomorphically encrypted graph convolutional network inference. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[232] D. Rathee, M. Rathee, G. R. K. Kiran, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi. Sirnn: A math library for secure rnn inference. *2021 IEEE Symposium on Security and Privacy (S&P)*, pages 1003–1020, 2021.

[233] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma. *CrypTFlow2: Practical 2-Party Secure Inference*, page 325–342. Association for Computing Machinery, New York, NY, USA, 2020.

[234] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29:1–98, 06 2017.

[235] A. Regev, S. Teichmann, E. Lander, I. Amit, C. Benoist, E. Birney, B. Bodenmiller, P. Campbell, P. Carninci, M. Clatworthy, H. Clevers, B. Deplancke, I. Dunham, J. Eberwine, R. Eils, W. Enard, A. Farmer, L. Fugger, B. Göttgens, and N. Yosef. Science forum: The human cell atlas. *eLife*, 6, 12 2017.

[236] U. M. L. Repository. Breast cancer wisconsin (original). https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original), 1992. (Accessed: 2023-03-03).

[237] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar. Xonn: Xnor-based oblivious deep neural network inference. In *USENIX Security*, 2019.

[238] K. Rohloff. The PALISADE Lattice Cryptography Library. https://git.njit.edu/palisade/PALISADE, 2018.

[239] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[240] A. Sadilek, L. Liu, D. Nguyen, M. Kamruzzaman, S. Serghiou, B. Rader, A. Ingerman, S. Mellem, P. Kairouz, E. O. Nsoesie, J. MacFarlane, A. Vullikanti, M. Marathe, P. Eastham,

J. S. Brownstein, B. A. y. Arcas, M. D. Howell, and J. Hernandez. Privacy-first health research with federated learning. *npj Digital Medicine*, 4(1):132, Sep 2021.

[241] A. Salem, Y. Zhang, M. Humbert, M. Fritz, and M. Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *Network and Distributed System Security Symposium (NDSS)*, 2019.

[242] S. Sav, J.-P. Bossuat, J. R. Troncoso-Pastoriza, M. Claassen, and J.-P. Hubaux. Privacy-preserving federated neural network learning for disease-associated cell classification. *Patterns*, 3(5), 2022.

[243] S. Sav, A. Diaa, A. Pyrgelis, J.-P. Bossuat, and J.-P. Hubaux. Privacy-preserving federated recurrent neural networks. *CoRR*, abs/2207.13947, 2022.

[244] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux. Poseidon: Privacy-preserving federated neural network learning. In *Network and Distributed System Security Symposium (NDSS)*, 2021.

[245] S. Sav, J. R. Troncoso-Pastoriza, A. Pyrgelis, D. Froelicher, J. A. Gomes de Sá e Sousa, J.-P. Bossuat, and J.-P. Hubaux. System and method for privacy-preserving distributed training of neural network models on distributed datasets, 2022.

[246] A. M. Schäfer and H. G. Zimmermann. Recurrent neural networks are universal approximators. In *Artificial Neural Networks – ICANN 2006*, pages 632–640, 2006.

[247] T. P. Schoenmakers B. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *Vaudenay S. (eds) Advances in Cryptology - EUROCRYPT 2006*, pages 522–537, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[248] P. Schoppmann, A. Gascon, M. Raykova, and B. Pinkas. Make some room for the zeros: Data sparsity in secure distributed machine learning. In *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[249] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, Nov. 1979.

[250] M. Sheller, B. Edwards, G. Reina, J. Martin, S. Pati, A. Kotrotsou, M. Milchenko, W. Xu, D. Marcus, R. Colen, and S. Bakas. Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data. *Scientific Reports*, 10, 2020.

[251] E. Shi, T.-H. Chan, E. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, volume 2, 01 2011.

[252] Y. Shi, M.-Y. Hwang, K. Yao, and M. Larson. Speed up of recurrent neural network language models with sentence independent subsampling stochastic gradient descent. In *INTERSPEECH*, 2013.

[253] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[254] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy (S&P)*, 2017.

[255] R. Shokri, G. Theodorakopoulos, G. Danezis, J.-P. Hubaux, and J.-Y. Le Boudec. Quantifying location privacy: The case of sporadic location exposure. pages 57–76, 2011.

[256] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J.-P. Hubaux. Quantifying location privacy. In *2011 IEEE Symposium on Security and Privacy*, pages 247–262, 2011.

[257] S. Simmons, C. Sahinalp, and B. Berger. Enabling privacy-preserving gwass in heterogeneous human populations. *Cell Systems*, 3(1):54–61, Jul 2016.

[258] H. Smajlović, A. Shajii, B. Berger, H. Cho, and I. Numanagić. Sequre: a high-performance framework for secure multiparty computation enables biomedical data sharing. *Genome Biology*, 24(1):5, Jan 2023.

[259] S. Song, K. Chaudhuri, and A. D. Sarwate. Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 245–248, 2013.

[260] T. Stevens, C. Skalka, C. Vincent, J. Ring, S. Clark, and J. Near. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1379–1395, 2022.

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

[262] I. Stoica, D. Song, R. A. Popa, D. A. Patterson, M. W. Mahoney, R. H. Katz, A. D. Joseph, M. I. Jordan, J. M. Hellerstein, J. E. Gonzalez, K. Goldberg, A. Ghodsi, D. E. Culler, and P. Abbeel. A Berkeley view of systems challenges for AI. *CoRR*, abs/1712.05855, 2017.

[263] Z. Sultana, D. M. A. Khan, and N. Jahan. Early breast cancer detection utilizing artificial neural network. *WSEAS TRANSACTIONS ON BIOLOGY AND BIOMEDICINE*, 18:32–42, 03 2021.

[264] S. Tan, B. Knott, Y. Tian, and D. J. Wu. Cryptgpu: Fast privacy-preserving machine learning on the gpu. *2021 IEEE Symposium on Security and Privacy (S&P)*, pages 1021–1038, 2021.

[265] X. Tang, Y. Huang, J. Lei, H. Luo, and X. Zhu. The single-cell sequencing: New developments and medical applications. *Cell & Bioscience*, 9, 2019.

[266] B. Terhal. Quantum supremacy, here we come. *Nature Physics*, 14(06), 2018.

[267] Top applications of deep learning across industries. https://www.mygreatlearning.com/blog/deep-learning-applications/. (Accessed: 2022-01-06).

[268] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu. 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*, pages 480–501. Springer, 2020.

[269] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu. Data poisoning attacks against federated learning systems. In L. Chen, N. Li, K. Liang, and S. Schneider, editors, *Computer Security – ESORICS 2020*, pages 480–501, Cham, 2020. Springer International Publishing.

[270] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction {APIs}. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 601–618, 2016.

[271] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction APIs. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, page 601–618, USA, 2016. USENIX Association.

[272] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou. A hybrid approach to privacy-preserving federated learning. In *ACM Workshop on Artificial Intelligence and Security (AISec)*, 2019.

[273] M. Uddin, Y. Wang, and M. Woodbury-Smith. Artificial intelligence for precision medicine in neurodevelopmental disorders. *NPJ Digital Medicine*, 2, 2019.

[274] T. van Elsloo, G. Patrini, and H. Ivey-Law. Sealion: a framework for neural network inference on encrypted data. *CoRR*, abs/1904.12840, 2019.

[275] S. Vieira, W. H. Pinaya, and A. Mechelli. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience & Biobehavioral Reviews*, 74:58–75, 2017.

[276] A. Vizitu, C. Nită, A. Puiu, C. Suciu, and L. Itu. Applying deep neural networks over homomorphic encrypted medical data. *Computational and Mathematical Methods in Medicine*, 2020:1–26, 2020.

[277] D. Vizár and S. Vaudenay. Cryptanalysis of chosen symmetric homomorphic schemes. *Studia Scientiarum Mathematicarum Hungarica*, 52:288–306, 06 2015.

[278] S. Wagh, D. Gupta, and N. Chandran. Securenn: 3-party secure computation for neural network training. *Privacy Enhancing Technologies (PoPETS)*, 2019.

[279] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin. FALCON: Honest-majority maliciously secure framework for private deep learning. *PETS*, 2021.

[280] A. Wainakh, F. Ventola, T. Müßig, J. Keim, C. Garcia Cordero, E. Zimmer, T. Grube, K. Kersting, and M. Mühlhäuser. User-level label leakage from gradients in federated learning. *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2022:227–244, 04 2022.

[281] J. Wang and G. Joshi. Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms. *CoRR*, abs:1808.07576, 2018.

[282] T. Wang, J. Bai, and S. Nabavi. Single-cell classification using graph convolutional networks. *BMC Bioinformatics*, 22, 07 2021.

[283] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 2512–2520, 2019.

[284] S. Warnat-Herresthal, H. Schultze, K. Shastry, S. Manamohan, S. Mukherjee, V. Garg, R. Sarveswara, K. Händler, P. Pickkers, N. A. Aziz, S. Ktena, F. Tran, M. Bitzer, S. Ossowski, N. Casadei, C. Herr, D. Petersheim, U. Behrends, F. Kern, and T. Velavan. Swarm learning for decentralized and confidential clinical machine learning. *Nature*, 594, 2021.

[285] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. V. Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Transactions on Information Forensics and Security (TIFS)*, 15:3454–3469, 2020.

[286] M. Wen, R. Xie, K. Lu, L. Wang, and K. Zhang. Feddetect: A novel privacy-preserving federated learning framework for energy theft detection in smart grid. *IEEE Internet of Things Journal*, pages 1–1, 2021.

[287] D. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Scalable anonymous group communication in the anytrust model. In *Technical Report*, 2012.

[288] N. Wu, F. Farokhi, D. Smith, and M. A. Kaafar. The value of collaboration in convex machine learning with differential privacy. *CoRR*, abs/1906.09679, 2019.

[289] Y. Yang, P. Gohari, and U. Topcu. On the privacy risks of deploying recurrent neural networks in machine learning. *CoRR*, abs/2110.03054, 2021.

[290] Y.-R. Yang and W.-J. Li. Basgd: Buffered asynchronous sgd for byzantine learning. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11751–11761. PMLR, 18–24 Jul 2021.

[291] I.-C. Yeh and C. hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36(2):2473 – 2480, 2009.

[292] J. Yoon, J. Jordon, and M. van der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations (ICLR)*, 2019.

[293] L. Yu, L. Liu, C. Pu, M. Gursoy, and S. Truex. Differentially private model publishing for deep learning. In *2019 2019 IEEE Symposium on Security and Privacy (S&P)*, pages 326–343, Los Alamitos, CA, USA, may 2019. IEEE Computer Society.

[294] X. Yuan, K. Chen, J. Zhang, W. Zhang, N. Yu, and Y. Zhang. Pseudo label-guided model inversion attack via conditional generative adversarial network. *AAAI*, 2023.

[295] A. Zalcman et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505–510, 10 2019.

[296] H. Zang and J. Bolot. Anonymization of location data does not work: A large-scale measurement study. In *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, MobiCom '11, page 145–156. Association for Computing Machinery, 2011.

[297] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu. Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC'20, USA, 2020. USENIX Association.

[298] D. Zhang. Big data security and privacy protection. In *Proceedings of the 8th International Conference on Management and Computer Science (ICMCS 2018)*, pages 275–278. Atlantis Press, 2018/10.

[299] M. Zhang, Z. Ren, Z. Wang, P. Ren, Z. Chen, P. Hu, and Y. Zhang. Membership inference attacks against recommender systems. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 864–879, New York, NY, USA, 2021. Association for Computing Machinery.

[300] Q. Zhang, C. Wang, H. Wu, C. Xin, and T. V. Phuong. Gelu-net: A globally encrypted, locally unencrypted deep neural network for privacy-preserved learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 3933–3939. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[301] Y. Zhang, G. Bai, X. Li, C. Curtis, C. Chen, and R. K. L. Ko. Privcoll: Practical privacy-preserving collaborative machine learning. In *Computer Security – ESORICS 2020*, pages 399–418, Cham, 2020. Springer International Publishing.

[302] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song. The secret revealer: Generative model-inversion attacks against deep neural networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 250–258, Los Alamitos, CA, USA, jun 2020. IEEE Computer Society.

[303] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong. 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*, KDD '22, page 2545–2555, New York, NY, USA, 2022. Association for Computing Machinery.

[304] B. Zhao, K. R. Mopuri, and H. Bilen. iDLG: Improved deep leakage from gradients. *CoRR*, abs/2001.02610, 2020.

[305] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra. Federated learning with non-IID data. *CoRR*, abs/1806.00582, 2018.

[306] L. Zheng, G. Wang, F. Zhang, Q. Zhao, C. Dai, and N. Yousefi. Breast cancer diagnosis based on a new improved elman neural network optimized by meta-heuristics. *International Journal of Imaging Systems and Technology*, 30(3):513–526, 2020.

[307] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica. Helen: Maliciously secure coopetitive learning for linear models. In *IEEE Symposium on Security and Privacy (S&P)*, 2019.

[308] H. Zhu, R. S. Mong Goh, and W.-K. Ng. Privacy-preserving weighted federated learning within the secret sharing framework. *IEEE Access*, 8:198275–198284, 2020.

[309] L. Zhu, Z. Liu, and S. Han. Deep leakage from gradients. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2019.

[310] X. Zhu, C. Vondrick, C. C. Fowlkes, and D. Ramanan. Do we need more training data? *Springer IJCV*, 119(1):76–92, Aug. 2016.

[311] T. Šestanović. Jordan neural network for inflation forecasting. *Croatian Operational Research Review*, 10:23–33, 07 2019.

[312] T. Šestanović and J. Arnerić. Can recurrent neural networks predict inflation in euro zone as good as professional forecasters? *Mathematics*, 9(19), 2021.

Sinem Sav
Chemin des Avelines, 1
1004, Lausanne, Switzerland
+41 (78) 962-2494, sinem.sav@epfl.ch

EDUCATION

École Polytechnique Fédérale de Lausanne (EPFL), Switzerland 2018 -
PhD Candidate, in School of Computer and Communication Sciences
Advisor: Prof. Jean-Pierre Hubaux, Prof. Carmela Troncoso

Bilkent University, Ankara, Turkey 2016 - 2018
Master of Science, in Computer Engineering
Advisor: Prof. Erman Ayday
CGPA 3.91

Bilkent University, Ankara, Turkey 2012 - 2016
Bachelor of Science, in Computer Engineering
CGPA 3.66

RESEARCH INTEREST

Privacy enhancing technologies, applied cryptography, big data privacy, privacy-preserving machine learning, federated learning, multiparty homomorphic encryption, biomedical/genomic data privacy.

WORK EXPERIENCE

HAVELSAN Inc., Ankara, Turkey September 2016 - March 2018
Industry Project
Privacy-Preserving Medical Databases, application of Paillier cryptosystem and homomorphic operations to health informations.

HAVELSAN Inc., Ankara, Turkey April 2016 - July 2016
Software Engineer (Candidate) Command Control and Combat Systems

Simon Fraser University, BC, Canada June 2015 - September 2015
Undergraduate Research Assistant, on RNA-Design problem with simulated-annealing
Advisor: Prof. Herbert H. Tsang

TAI, Turkish Aerospace Industry Inc., Ankara, Turkey June 2014 - July 2014
Intern, IT department.

TEACHING EXPERIENCE

Teaching Assistant Fall 2014 - Present
Bilkent University, Computer Science Department, Ankara, Turkey

- Algorithms and Programming I-Java (CS-101).
- Introduction to Programming for Engineers - Java (CS-114).
- Software Architecture Design (CS-411).
- Object Oriented Programming (CS-319).

EPFL, School of Computer and Communication Sciences

- Information Security and Privacy (COM-402).
- Mobile Networks (COM-405).

- Advanced Topics on Privacy Enhancing Technologies (CS-523)

JOURNAL PUBLICATIONS

- Sinem Sav, Abdulrahman Diaa, Apostolos Pyrgelis„ Jean-Philippe Bossuat, and Jean-Pierre Hubaux
  Privacy-Preserving Federated Recurrent Neural Networks.
  Proceedings on Privacy Enhancing Technologies (PoPETs), 2023(4).
- Sinem Sav, Jean-Philippe Bossuat, Juan R. Troncoso-Pastoriza, Manfred Claassen, and Jean-Pierre Hubaux
  Privacy-Preserving Federated Neural Network Learning for Disease-Associated Cell Classification.
  Patterns, 3(5), 2022.
- David Froelicher, Juan R. Troncoso-Pastoriza, Apostolos Pyrgelis, Sinem Sav, Joao Sa Sousa, Jean-Philippe Bossuat, and Jean-Pierre Hubaux
  Scalable Privacy-Preserving Distributed Learning. *Proceedings on Privacy Enhancing Technologies (PoPETs), 2021(2).*

CONFERENCE PUBLICATIONS

- Sinem Sav, Apostolos Pyrgelis, Juan R. Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux
  POSEIDON: Privacy-Preserving Federated Neural Network Learning.
  Network and Distributed Systems Security (NDSS) Symposium, 2021.
  Selected as the best paper in CSAW'21 Applied Research Competition in Europe.
  Selected talk for PPML NeurIPS, 2020.
- Sinem Sav, David Hampson, and Herbert H. Tsang,
  SIMARD: A Simulated Annealing Based RNA Design Algorithm with Quality Pre-Selection Strategies. *IEEE Symposium Series on Computational Intelligence (SSCI), 2016.*
- Halid Emre Erhan, Sinem Sav, Stas Kalashnikov, and Herbert H. Tsang,
  Examining the Annealing Schedules for RNA Design Algorithm. *IEEE Congress on Evolutionary Computation, July 24-29, 2016.*
- David Hampson, Sinem Sav, and Herbert H. Tsang,
  Investigation of Multi-Objective Optimization Criteria for RNA Design.
  IEEE Symposium Series on Computational Intelligence (SSCI), 2016.

WORKSHOP PUBLICATIONS

- Francesco Intoci*, Sinem Sav*, Apostolos Pyrgelis, Jean-Philippe Bossuat, Juan R. Troncoso-Pastoriza, and Jean-Pierre Hubaux
  SlytHErin: An Agile Framework for Encrypted Deep Neural Network Inference
  Accepted at 5th Workshop on Cloud Security and Privacy (Cloud S&P 2023) co-located with ACNS.

PATENTS

- David Froelicher, Juan Ramón Troncoso-Pastoriza, Apostolos Pyrgelis, Sinem Sav, Joao André Gomes de Sá e Sousa, Jean-Pierre Hubaux, Jean-Philippe Bossuat
  System and method for privacy-preserving distributed training of machine learning models on distributed datasets, 2021
  Patent no: WO/2021/223873
- Sinem Sav, Juan Ramón Troncoso-Pastoriza, Apostolos Pyrgelis, David Froelicher, Joao André Gomes de Sá e Sousa, Jean-Philippe Bossuat, Jean-Pierre Hubaux
  System and method for privacy-preserving distributed training of neural network models on distributed datasets, 2022.
  Patent no: WO/2022/042848

TALKS

- Privacy-Preserving Federated Neural Network Learning for Disease-Associated Cell Classification
  - ❖ Highlight talk at 27th Annual International Conference on Research in Computational Molecular Biology (RECOMB2023), April 2023, Turkey.
- POSEIDON: Privacy-Preserving Federated Neural Network Learning
  - ❖ CAp2021: Conférence francophone en Apprentissage, June 15, 2021 (online).
  - ❖ Contributed talk for PPML NeurIPS'20, December 11, 2020 (online).
  - ❖ RISELab, UC Berkeley, 2021 (online).
- Privacy-Preserving Federated Learning with Multiparty Homomorphic Encryption
  - ❖ Workshop on Privacy Preserving systems, softwares, and tools at the Department of Mathematics and Physics of the Roma Tre University, October 24, 2022, Italy.
  - ❖ Lecture in Advanced Topics in Computer and Network Security, University of Padua, October 27, 2022, Italy.
  - ❖ Contributed talk and invited panelist at the 3rd International Workshop "Towards Auditable AI Systems: From Use Cases to Standardization & Regulation", November 24, 2022, Germany.

SERVICE

Reviewer/Sub-reviewer: PoPETS, USENIX Security, IET Information Security, BMC Medical Informatics, Computers & Security, ISMB/ECCB 2023.

STUDENT SUPERVISION

- Natalija Mitic (Ongoing), Master semester project (12 ECTS), Fall 2022.
- Francesco Intoci (Ongoing), Master semester project (12 ECTS), Spring 2022.
- Abdulrahman Diaa, Privacy-Preserving Federated Recurrent Neural Networks, Summer@EPFL, 2021.
- Xavier Oliva I Jurgens, Privacy-Preserving Federated Hyperparameter Tuning on Non-IID Data Silos: A Measurement Study, Master semester project (12 ECTS), Fall 2021.
- Shufan Wang, Privacy-Preserving Federated Neural Network Training for Disease Associated Cell Detection, Master semester project (12 ECTS), Spring 2021.
- Simon Nicolas Perriard, Privacy-Preserving Hyperparameter Tuning in Federated Learning Setting, Master semester project (12 ECTS), Spring 2021.
- Raphaël Reis Nunes, Distributed Learning with Neural Networks: a performance analysis under decentralization and server failure constrains, Bachelor semester project (8 ECTS), Spring 2020.
- Claire Marie Louise Lefrancq, Convolutional Neural Networks for Disease-Associated Cell Detection, Bachelor semester project (8 ECTS), Fall 2020.

HONORS & AWARDS

- 1st prize for the paper "POSEIDON: Privacy-Preserving Federated Neural Network Learning " in CSAW'21 Applied Research Competition (Prize: 700€).
- 2nd place for the "Homomorphic Encryption-based Secure Viral Strain Classification", iDASH21.
- Awarded with tuition waiver for Mitacs Globalink Programme, Canada.
- Awarded with tuition waiver from Bilkent University due to high ranking in University Entrance Exam.
- Bilkent University, Senior Design Project, the Best Demonstration Award.

EXTRA-CURRICULAR ACTIVITIES

Scuba Diving
- PADI Open Water Diver

Horse Riding
- Ankara AtliSpor Club September 2017-2018

Advanced Squash Player
- Bilkent University, Squash Tournament Fall 2017, 1st Place
- Bilkent University, Squash Tournament Spring 2016, 2nd Place

Radyo Bilkent, Ankara, Turkey DJ, September 2013 – September 2014
- Broadcasted 4 hours per week at Radyo Bilkent.