Applications of secure multi-party computation in Machine Learning

Angelo Saadeh

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAT022

Thèse de doctorat

Applications of Secure Multi-party Computation in Machine Learning

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 8 juin 2023, par

ANGELO SAADEH

Composition du Jury :

Catuscia Palamidessi Directrice de recherche, Inria Saclay	Présidente / Examinatrice
Anderson Nascimento Professeur associé, University of Washington	Rapporteur
Melek Önen Professeure associée, Eurecom	Rapporteure
Jan Ramon Directeur de recherche, Inria Lille	Examinateur
Daniel Augot Directeur de recherche, Inria Saclay	Directeur de thèse
Matthieu Rambaud Maître de conférences, Télécom Paris	Co-directeur de thèse

Acknowledgements

First and foremost I thank the reviewers, Melek Önen for her constructive remarks, and Anderson Nascimento for all the technical comments that enriched this thesis. I also thank the examiners Catuscia Pamiledessi and Jan Ramon for their time and interest in this thesis.

Ce parcours n'aurait pas été possible sans un certain nombre de personnes, je les remercie tous énormément. En particulier mes encadrants ; Matthieu Rambaud, pour ses efforts pour que non seulement je réussisse cette thèse mais aussi pour que je puisse accomplir tous mes projets futurs, et Daniel Augot, pour qui je n'ai pas mots tellement il m'a donné confiance, a été bienveillant et amical avec moi. Je remercie Stéphane Bressan qui a beaucoup contribué à cette thèse. Ses remarques et celles de Pierre Sennelart n'ont fait qu'améliorer mes travaux de recherche.

Je garde un excellent souvenir de mes jours à l'Inria avec mes collègues, quelle che parlano italiano et quelle che lo parlano meno bene. Je les aime très sincèrement.

Je n'aurai jamais eu le bagage nécessaire pour la réussite de cette thèse sans le soutient de mes professeurs de l'université libanaise. En particulier Fida El-Chami et Georges Abou Abdo. Je voudrais aussi remercier Ghada Kawkabani qui était mon enseignante de mathématiques au lycée. C'est grâce à elle si j'ai une grande passion pour cette discipline.

I would like to mention my friends who relieved me during difficult times and who added a lot of joy to my daily life. Some of them have been doing so for more than twenty years: Anna, Jean-Paul, Joanna, Laura, Marie-Ange, Myriam, Rodolfo, and Vincent.

Finalement, je voudrais remercier ma mère qui a toujours priorisé mon éducation. Elle a sacrifié beaucoup pour financer mes études depuis que j'étais enfant. Elle est le maillon le plus important dans mon parcours.

Contents

Résumé en français - version longue

Introduction

Le calcul multipartite sécurisé basé sur le partage de secrets, simplement appelé multipartite sécurisé ou MPC dans cette thèse, est un outil cryptographique qui permet l'exécution sécurisée de calculs sur des données de plusieurs entités sans révéler les données d'aucune des entités impliquées. Cet outil a le potentiel de changer la donne dans le domaine de l'apprentissage automatique, car il offre un moyen de garantir la confidentialité des détenteurs de données.

Le MPC a déjà été utilisé en apprentissage automatique pour analyser de grands ensembles de données afin de détecter et limiter des activités criminelles. Plus concrètement, Roseman Labs, une startup qui travaille sur le MPC, ont collaboré avec l'ONG Sustainable Rescue pour lutter contre la traite des êtres humains. Leur solution utilise le partage de secret pour tester si le nom d'une victime qui a demandé l'aide à l'ONG est inclus dans la liste des victimes potentielles des forces de l'ordre , afin que les forces de l'ordre puissent agir et retrouver les trafiquants. Garder le nom des victimes secret est important afin d'éviter que celles-ci aient peur et fuient.

Les contributions de cette thèse portent spécifiquement sur l'utilisation du calcul multipartite sécurisé dans les algorithmes d'apprentissage automatique et d'analyse de données afin de pouvoir exécuter ces algorithmes sur des bases de données combinées où chaque base de données appartient à plusieurs entités sans qu'aucune des entités ne révèle ses données à qui que ce soit.

Regression Logistique avec Confidentialité Différentielle à Deux Joueurs

Cas d'usage : La régression logistique est un type d'algorithme d'apprentissage automatique utilisé pour prédire la probabilité d'une variable dépendante (binaire). Lors de l'apprentissage d'une régression logistique sur un ensemble de données séparées, où chaque section appartient à une entité distincte appelée " joueur ", il est essentiel de s'assurer que l' " entrée secrète partielle de données " de chaque joueur reste privée. L'objectif des deux joueurs est d'obtenir le " modèle public " en sortie entrainé sur l'ensemble des données partielles et secrètes combinées. Aucun des joueurs ne connaît entièrement les données complètes ; les données des joueurs doivent rester privées.

Contribution : Cette contribution – publiée dans [SKB22] – se démarque des travaux antérieurs car elle combine pour la première fois ces trois garanties :

1. Nous satisfaisons la sécurité passive selon la définition 3 - obtenue en utilisant exclusivement un calcul multipartite basé sur le partage de secrets.

2. La sortie du modèle public a une confidentialité différentielle ϵ [DR14].

3. La sortie du modèle public a une bonne précision en raison de l'utilisation du mécanisme fonctionnel [Zha+12; CMS11].

Comme les calculs sont effectués avec un schema de partage de secrets, seule la sortie du modèle public doit être révélée. Cependant, il n'est pas nécessaire de le révéler, il pourrait être gardé secret et être utilisé pour faire des prédictions à l'aide de protocoles, dans ce cas, les joueurs devraient tous les deux s'entendre pour faire l'inférence.

Découverte de Vérité Confidentielle avec le Calcul Multipartite Sécurisé

Cas d'usage : Les algorithmes de découverte de vérité servent à corroborer des données provenant de sources en désaccord pour trouver la valeur de vérité d'une ou de plusieurs requêtes. Le vote à la majorité peut être considéré comme un type d'algorithme de découverte de vérité. De manière générale, ces algorithmes prennent en entrée les réponses des plusieurs sources sur un ensemble de requêtes, puis les comparent et les analysent afin de trouver la valeur de vérité de chacune des requêtes. Comme ce processus pourrait impliquer le partage d'informations confidentielles, il est nécessaire de disposer de protocoles qui traitent des questions de confidentialité et de respect de la vie privée. Par exemple, un créancier peut souhaiter déterminer si les demandeurs de prêt sont solvables. Le créancier souhaiterait fonder sa décision sur les différentes évaluations d'institutions financières. Toutefois, les institutions financières ne s'engagent à apporter leurs évaluations respectives à la décision que si elles ne sont révélées ni au créancier, ni aux autres institutions financières, ni à des tiers. Dans ce but, nous utilisons le calcul multipartite sécurisé.

Contribution : Dans notre solution [SSB23], les sources font un partage de secret de leurs réponses sur deux serveurs qui exécutent l'algorithme de découverte de vérité en utilisant des protocoles de calcul bipartites. Le nombre de serveurs pourrait être plus élevé, mais nous nous en tenons à deux joueurs. Nous présentons un test d'égalité sécurisé arithmétique que nous utilisons pour calculer des additions qui dépendent d'une condition, une opération récurrente dans les algorithmes de découverte de vérité. Notre test d'égalité est rapide car il correspond à une évaluation polynomiale de degré deux. Pour évaluer les performances de la solution décrite, deux algorithmes de recherche de vérité, Cosinus et 3-Estimations [Gal+10], sont exécutés de manière sécurisée, puis les résultats sont comparés à ceux des executions en clair (sans cryptographie).

Primitives pour l'Apprentissage Automatique Confidentiel

Cas d'usage : Les primitives basées sur le partage de secrets pour l'apprentissage automatique sont un ensemble de protocoles et d'algorithmes qui peuvent être utilisés pour partager en toute sécurité des données dans le but de collaborer et d'entraîner des modèles avec différents détenteurs de données.

Contribution : Cette contribution comprend des primitives de calcul bipartites sécurisées basées sur le partage de secrets pour les applications d'apprentissage automatique. Nous expliquons comment le calcul multipartite sécurisé en général affecte la précision des résultats, et nous donnons un protocole pour les tests d'égalité sécurisé lorsque les entrées sont dans un ensemble fini défini publiquement. Le test d'égalité accélère les calculs des algorithmes d'apprentissage automatique comme les machines de Boltzmann par exemple, comme expliqué dans 6.2.3.2. Nous présentons

aussi un protocol arithmétique pour calculer le signe d'un secret d'une façon sécurisée. Nous implémentons les protocoles que nous proposons dans un réseau sockets dans Python où on peut tester les différents protocoles et exécuter des algorithmes de manière sécurisée en calcul multipartite.

Chapter 1

Introduction

Machine learning techniques have become very widespread, they are frequently used to process any kind of data, including sensitive data. Preserving the privacy of data holders is a necessity, not only because their data is confidential, but also due to law and regulation guidelines that could prevent data holders from sharing some kind of data.

Secret-sharing-based secure multi-party computation, simply referred to as secure multi-party or MPC in this thesis, is a concept in cryptography, which allows multiple entities - also called parties or players - to compute a public function that takes their secret data as input. This computation is done without having them reveal their data to any other involved party. This tool has the potential to be a game-changer when applied to the field of machine learning, as it offers a way to ensure the privacy of data holders. This thesis will delve into the applications of secure multi-party computation in machine learning and data mining and assess its potential beneficial and detrimental implications.

The present introduction begins in section 1.1 by articulating the underlying problem this thesis seeks to address. Subsequently, section 1.2 examines the state-of-the-art techniques for conducting machine learning via not only secure multi-party computation, but as well as other tools, techniques, and algorithms used for privacy-preserving machine learning, such as homomorphic encryption, differential privacy, and other privacy-enhancing technologies. Having examined the relevant existing literature, section 1.3 outlines the contributions of this thesis and it is followed by an emphasis on the outline of the thesis in section 1.4.

1.1 Problem Definition

Data privacy in machine learning is the practice of protecting the privacy, confidentiality, and security of data used in machine learning applications and algorithms. It is achieved by using techniques such as data masking, data obfuscation, data encryption, data segmentation, and other tools. Privacy-preserving tools are used to protect the data from unauthorized access, prevent data leakage, and ensure that only authorized users have access to the data.

Although the scenarios of privacy-preserving machine learning may differ - and they are numerous - they all have one objective, ensure the data remains secure, private, and confidential while training and testing machine learning algorithms and ensuring that the output of the machine learning algorithm does not reveal any sensitive or confidential information.

Data security is the process of protecting data from unauthorized access, destruction, modification, or disclosure. It involves implementing physical, technical, and administrative safeguards to ensure the confidentiality, integrity, and availability of data. Examples of data security measures

include access control systems, firewalls, encryption, and antivirus software. Whereas data privacy is the right of individuals to control who has access to their personal information. It involves protecting the personal data of an individual and recognizing their right to privacy. Examples of data privacy measures include data minimization, data control, data retention, and data destruction practices. Data confidentiality on the other hand is the practice of keeping data secure by limiting access to authorized personnel. It involves controlling who can access data and preventing unauthorized individuals from viewing or modifying it. Examples of data confidentiality measures include user authentication and encryption.

The goal of privacy-preserving machine learning is to avoid problems related to data leakage, and data privacy issues during training or during using machine learning models. Data leakage is the unauthorized disclosure of information relative to private data. It occurs when a part of private data is unintentionally exposed to the public or to unauthorized users. Whereas, data privacy issues arise when personal information is used without the consent of the individual. This can be done through the misuse of data collected from individuals or the sharing of personal information without their consent. These issues are more likely to happen when multiple parties are training a model on their combined data, or when a party is using another party's data to train a machine learning algorithm.

In each scenario, the considered parties define their own privacy and security standards, and they use tools that satisfy these standards. Consequently, there is no absolute right or wrong privacy-preserving technology to use, it all depends on the regulation, rules, and specifications that ought to be respected, and the type of situation in which the data holders and the parties training the models are in.

1.1.1 Privacy-preserving Tools for Machine Learning

The first example of privacy-preserving tools for machine learning is the one around which all the contributions of this thesis are, secret-sharing-based secure multi-party computation. It is a cryptographic tool that allows multiple parties to jointly compute a function over their inputs while keeping those inputs private from each other. Formal definitions and details about secure multi-party computation are given in section 2.1 and in section 2.2 we give the basic protocols that are used to securely evaluate functions.

Another privacy-preserving machine learning tool is homomorphic encryption, which is a form of encryption that allows computation to be done directly on encrypted data without needing to decrypt it first. This means that calculations can be carried out on the data in its encrypted form, and the result is also in an encrypted form. This allows for secure computation to occur without exposing the underlying data. This has many applications in the fields of cloud computing and data privacy, as it allows computations to be done without compromising the security of the data.

A non-cryptographic tool for privacy-preserving machine learning is differential privacy; it provides a privacy guarantee for individuals by injecting noise into data, the algorithm computing over the data, or the output of the algorithm. Differential privacy helps ensure that no individual's data can be identified or linked to them and that no individual's information is leaked. Differential privacy also prevents data re-identification and gives users the ability to control access to their data. We identify however two types of differential privacy algorithms. In the first one, the amount of noise added is determined by a parameter, ϵ, which determines the level of privacy of the dataset. The lower the value of ϵ, the greater the privacy of the dataset. In the second one, a second parameter, δ is added. The parameter δ is a confidence parameter that determines the probability that the privacy guarantee of the dataset will be violated. The lower the value of δ, the more confident the user can be that their privacy will be preserved. One can conclude that ϵ-differential privacy

has a better privacy guarantee than ϵ, δ-differential privacy because in the former the parameter δ is null.

Another non-cryptographic privacy-preserving machine learning is federated learning, a type of distributed machine learning where data is stored and processed on individual devices instead of a central server. It enables distributed training of models across multiple servers without the need to share any raw data with a centralized server. This allows users to keep their data private and secure while still allowing the model to be collaboratively trained with data from other users. With federated learning, models can be trained to recognize patterns in data without the need to transfer large amounts of data between devices.

As mentioned before, no single solution can guarantee complete privacy in machine learning. The degree of privacy achieved by the aforementioned tools is contingent upon a variety of elements, such as the machine learning algorithm employed, the adversary's capabilities and resources, the regulations, and so on. Therefore, to reach higher levels of privacy, it could be necessary to strategically combine various privacy-preserving machine learning tools.

1.1.2 Real World Applications

Privacy-preserving tools in machine learning technologies could be applied in many sectors such as healthcare where medical records are highly sensitive, and preserving patient privacy is paramount. Privacy-preserving machine learning techniques like secure multi-party computation can be used to protect sensitive healthcare data from being accessed by unauthorized parties. By securely sharing and computing data from multiple parties, a secure protocol can be established to ensure that only the relevant parties have access to the data. Privacy-preserving tools can be used to analyze large sets of medical data, such as genomic data, without exposing individual identities or data points.

Another area of application is law enforcement, where using machine learning to analyze large datasets in order to detect potential criminal activity is increasing. A concrete example of such an application was done by Roseman Labs, a startup that works on secure multi-party computation. They collaborated with the NGO Sustainable Rescue to fight against human trafficking. Their solution uses secret-sharing to test if the name of a victim that sought an NGO for help is included in the law enforcement potential victims list, so that the law enforcement could act and find the traffickers. Keeping the victim's name secret would prevent them from going off-grid.

Another sector of application is finance. Indeed, credit card data, investment portfolios, and other financial records are all highly sensitive and must be kept private. Privacy-preserving machine learning algorithms can analyze financial data without exposing individual identities or data points. Secure multi-party computation can be used to securely share and compute data related to credit risk analysis and credit scoring. This helps financial institutions to accurately assess credit risk and make more informed decisions with regard to granting credit.

Finally, social networks contain highly sensitive data about users. Privacy-preserving machine learning algorithms can be used for collaborations between social media platforms and advertisers without revealing the users' data. For example, Meta proposed a product called Private Lift Measurement [Met20] which uses secure multi-party computation to help advertisers understand how their campaigns are performing while limiting what the advertiser and Meta can learn about a person.

1.2 Literature Review

The next sections present the state-of-the-art improvements of privacy-preserving tools and the way they are applied in machine learning and data mining. The non-main technologies that have

yet to be introduced will be explained when they are mentioned.

1.2.1 Privacy-Enhancing Technologies

In this section, we briefly introduce and review seminal works about privacy-enhancing technologies like differential privacy and secure multi-party computation with secret-sharing and homomorphic encryption. There are other privacy-enhancing technologies but we do not address them in this thesis.

Secure Multi-party Computation and Secret Sharing

Yao's millionaire problem is a well-known problem introduced by Andrew Yao in [Yao82]. The problem is the following: two millionaires, Alice and Bob, are interested in knowing which of them is the richest without revealing their actual wealth. The solution introduced for the first time secure two-party computation with the garbled circuit, which was detailed by Yao in [Yao86] and later by many others. Garbled circuits are a cryptographic technique for securely computing a public function on secret binary input strings without revealing them. The two parties do so by evaluating the binary circuit of the function, i.e. circuit containing AND and XOR gates.

A secret-sharing scheme is a method of distributing a secret among a group of participants, each of whom is assigned a share of the secret. The secret cannot be reconstructed unless a certain number of shares are combined. In [GMW87], the authors introduced the first secret sharing-based two-party protocol which could be extended to more than two parties. This protocol uses a cryptographic technique called oblivious transfer; it allows a party to transfer one of many pieces of information to another party who would remain oblivious to what piece (if any) has been transferred. The secret-sharing scheme used in this work is called additive secret-sharing, and it is one that we use in the thesis.

In [BGW88] the authors use Shamir's secret sharing scheme [Sha79] and suppose that all channels of communication are secure. The security model of this seminal work is often used as a reference.

Homomorphic Encryption

Homomorphic encryption is a specific form of encryption that allows computations on encrypted data. Computations on the encrypted data would yield an encrypted result which, after decryption, corresponds to the result of the operations as if they were computed on the plaintext. Several types of homomorphic encryption schemes can be defined, depending on the nature of the operations. They allow performing encryption schemes such as multiplicative encryption schemes, which make it possible to perform multiplication on encrypted data. The two encryption schemes RSA [RSA78] and ElGamal [Gam85] are multiplicative encryption schemes. Indeed, with these two encryption schemes, multiplying two encrypted elements makes it possible to obtain an encryption of the product of the two plain texts. Another encryption scheme is the additive encryption scheme which similarly makes it possible to perform an addition on encrypted data and by extension a multiplication between an encrypted value and a clear value (multiplication by a public constant). Paillier's cryptosystem [Pai99] is an example of such schemes. Lastly, fully homomorphic encryption schemes support both additions and multiplications, making any circuit evaluable over encrypted data. The first build was given by Gentry in 2009 [Gen09]. Despite significant optimizations since then, the size of the cryptographic keys and the overhead to perform operations are still a significant issue.

Differential Privacy

When a dataset is used to train a machine learning model, the model output could reveal information about the dataset that was used to train it. Differential privacy [Dwo+06] protects the data once the model is shared with another party. It prevents attacks like membership inference attacks [Sho+17].

Overfitting in machine learning is when a model is trained to a point where its performance on the training data is very good, but its performance on unseen data is poor. This is due to the model having sort of memorized the training data, resulting in poor generalization. Overfitting a model makes it vulnerable to membership inference attacks, and one way to prevent that is by using regularization techniques. Indeed, regularization techniques help to reduce the complexity of a model by adding constraints to the model during the training process. It is used to improve the generalization of a model by penalizing the weights of the model. One type of such technique is a dropout [Sri+14], which randomly removes some of the computation nodes during each training iteration. This helps the network to learn more efficiently because by dropping out some neurons, the network is forced to use different neurons to make predictions, thus avoiding overfitting. Another way to avoid overfitting and guaranteeing privacy, in general, is by using differential privacy. If the training process is differentially private, the probability of producing a given model from a training dataset that includes a particular record is the same as the probability of having the same model when this record is not included. Differentially private models are designed to be secure against model inversion attacks and the membership inference attacks developed in [Sho+17] - as these attacks only work with the outputs of the model without any extra information. The only downside is that differentially private models may significantly lower prediction accuracy when the privacy parameter ϵ is small. In this thesis, we use differential privacy to train logistic regression models in order to protect the model.

Different techniques have been suggested to achieve differential privacy. One of these is the Laplace mechanism, which involves adding random noise to the released results (output perturbation) to ensure that the addition or removal of a single record does not significantly impact the results. Another output perturbation algorithm is presented in [Zha+17], they not only attain close to optimal utility but also markedly reduce the running time of the most current private optimization methods for both ϵ-differential privacy, and the weaker version ϵ, δ-differential privacy. Another method is objective perturbation which adds noise to the objective function rather than the parameters of the constructed models. In the paper [CMS11], the authors state that experimental as well as theoretical results indicate that in general, objective perturbation provides more accurate solutions than output perturbation. In our work, when we apply differential privacy, we use the functional mechanism [Zha+12], which is a highly effective and efficient objective perturbation technique, proven to have significantly outperformed previous solutions. Another differential privacy technique consists of adding noise to the inputs [Kan+20]. By adding noise to the original training data and training with the perturbed data, they achieve ϵ, δ-differential privacy on the final model, along with some kind of privacy on the original data.

The field of differential privacy is still evolving, for example, a recent work [Ami+22] introduced a tool that allows scaling differential privacy solutions. Indeed, real-world systems have users who can submit multiple records into them, the range of potential entries may be unclear, and the systems aren't always able to manage a large amount of data. The authors of [Ami+22] highlight that not taking into consideration all three issues can drastically reduce the quality and practicality of the systems. Hence they introduce a solution that addresses these issues.

1.2.2 Privacy-preserving Machine Learning and Data Mining

In this section, we review works on privacy-preserving machine learning algorithms. We chose the machine learning and data mining algorithms that discussed in this thesis, which are linear and logistic regression, truth-finding, and restricted Boltzmann machines. We also review frameworks for privacy-preserving machine learning.

Linear and Logistic Regressions

Linear regression [MPV01] and logistic regression [KK02] are both types of supervised machine learning algorithms that utilize labeled data to make predictions. Linear regression is a type of predictive analysis that is used to identify the linear relationship between one or more independent variables and a dependent variable. It is used to predict a continuous variable, such as the price of a house or the annual income of an individual. It is used to analyze and model the relationship between the dependent variable and one or more independent variables.

Logistic regression is a type of classification algorithm that is used to predict the probability of an event occurring. It is used to predict whether an instance belongs to one class or another. Unlike linear regression, logistic regression is not used to predict a continuous variable, but instead a discrete variable (e.g. yes/no, pass/fail, etc.). It is used to classify data into different categories and is useful for predicting the probability of an event occurring.

Both linear and logistic regression are briefly explained in the background chapter in section 2.3.2. In this section, we focus on the works that have been used for the protection the data privacy in federated linear and logistic regressions i.e. trained on datasets held by different parties.

Most works on privacy-preserving machine learning using secure multi-party computation were done in the last five years and use privacy-enhancing technologies. The majority of these works however tackles only horizontally-split datasets, like the following ones.

For example, [Nas+20] introduces a user-friendly tool that can train a logistic regression on distributed datasets (horizontally split) without revealing the parties' data as the training is locally executed, and by preserving the accuracy of the results. It does not require the exchange of raw data, but it exchanges model parameters between the parties and the central server. They do not use differential privacy or cryptographic tools therefore it is able to maintain the accuracy of the results. Their main application is the medical field and more specifically genome-wide association studies that are used to unravel connections between genetic variants and diseases.

Another work [GTJ22] presents a privacy-preserving logistic regression on horizontally-split data by secret-sharing the database between multiple parties to ensure that no single party has access to all the data. The authors present an algorithm for training logistic regression using various approximation approaches such as a fixed Hessian matrix, least-square approximation for the Sigmoid function, and a matrix inversion algorithm tailored to the honest majority and dishonest majority security settings of secure multi-party computation. They say that their proposed approach is more efficient in terms of runtime and communication complexity due to the approximations used instead of computing the actual logistic regression algorithm. The paper concludes that the proposed approach is a secure and efficient solution for privacy-preserving logistic regression.

In the approach of [Wan+19], a federated linear regression on a horizontally-split dataset is trained by making each local device perform a local update step and a global aggregation step - which both consume resources. Using a theoretical bound, a control algorithm has been proposed to achieve a desirable trade-off between local update and global aggregation in order to minimize the loss function under a resource budget constraint. An older work presented in [Aon+15] describes

a homomorphic encryption scheme with a flexible encoding of plain texts that could be executed quickly in a short amount of time and with minimal communication. Specifically, in the scenario of an honest-but-curious server and a client, their system was able to process a simulated dataset of 10^8 records, each containing 20 features, within 10 minutes.

The authors of [BP20] present a federated ϵ-differentially private logistic regression on horizontally-separated data by using secure multi-party computation. Their solution is accessible to readers without prior knowledge of security, encryption, privacy, or distributed learning. The work done in [Jay+18] also combines differential privacy with secure multi-party computation. Indeed, they use two methods of differential privacy, output perturbation, and gradient perturbation. They make advances to their state-of-the-art in a distributed learning setting. Their output perturbation approach involves combining local models within a secure computation, then adding the necessary differential privacy noise before the model is revealed. Similarly, their gradient perturbation involves the data owners collaboratively training a global model through an iterative learning algorithm. At each iteration, the parties aggregate their local gradients through a secure computation, adding noise to ensure privacy before the gradient updates are revealed.

Horizontal federated learning could also be done in cryptography-free approaches. One of these approaches is presented in [Pap+17], it consists of a learning strategy and corresponding privacy analysis for safeguarding sensitive training data. The proposed technique, known as PATE, uses information gathered and transferred from teacher models, trained on separate data, to create a student model that can be made public. PATE had only been tested on basic classification problems, such as MNIST, raising doubts about its usefulness for larger-scale learning tasks and real-world datasets. In the research [Pap+18], the authors demonstrated how PATE can be implemented for problems with many output classes and unstructured, uneven training datasets with errors. They proposed new, noisier grouping techniques for teacher groups that are more precise, create less noise, and prove ϵ-differential privacy guarantees.

Another category of works addresses the case of vertically-split datasets. However, this subject is less discussed than the case of horizontal separation.

For example, the authors of [Har+17] outline a secure three-party end-to-end solution that computes a vertical federated logistic regression over messages encrypted with an additively homomorphic scheme. This system is designed to protect data and prevent the exposure of which entities the data providers have in common. Their implementation is as accurate as a non-private solution that brings all data together and can handle millions of entities with hundreds of features.

The work described in [San+04] investigates the situation in which parties wish to train a privacy-preserving linear regression on a vertically split dataset using secure multi-party computation. It presents an algorithm that allows the parties to calculate the exact regression coefficients of the global regression equation, as well as perform some basic goodness-of-fit diagnostics, all while protecting the confidentiality of their data in the semi-honest security setting.

In the reference [JZG22], the authors develop and improve upon previous research by designing a general gradient-based reconstruction attack framework for logistic regression models trained on vertically-split data. Indeed, as mentioned before models can leak information during inference, and they say that among the solutions to protect the model is differential privacy. Some existing works achieve differential privacy in the case of a vertically split dataset. The works [Taj+20] and [Wan+20] for example satisfies (ϵ, δ)-differential privacy to train their logistic regressions. As mentioned, it is a weaker privacy guarantee than ϵ-differential privacy. The work in [Wu+16] highlights the differences between the two approaches and gives information regarding the use of (ϵ, δ)-differential privacy in the context of computation on secret data. In this master's thesis [Thi+19] the reader can find how to achieve differential privacy in secure multi-party computation in general.

In [XYW21] the authors introduce a new framework for satisfying ϵ-differential privacy in multi-

party learning when data is vertically partitioned among multiple parties. Their main concept is based on the functional mechanism which adds noise to the objective function to provide differential privacy, and it is the mechanism we use in our work on differentially private logistic regression in this thesis. Their approach only requires one round – which roughly means one interaction between the players – of noise addition and secure aggregation, but their algorithm proceeds by revealing to a third party (denoted as the server) intermediate information.

Another privacy-enhancing tool used in machine learning is functional encryption [BSW11]. It is a type of encryption that allows a user to encrypt data so that only specific operations can be performed on it. It works by encrypting a data set with a key that only allows certain functions to be performed on it. The paper [Xu+21] proposes a framework that eliminates the need for peer-to-peer communication among parties for secure gradient computation in vertical settings for various widely used machine learning models including linear models, logistic regression, and support vector machines. It makes use of functional encryption schemes in order to achieve quicker training times and is suitable for larger and changing sets of parties.

Many works aimed to offer federated learning solutions that suit both vertical and horizontal data separations. For example, the authors of [Coc+15] present an information-theoretically secure linear regression with two parties. It is computed using a trusted initializer in a pre-computation that generates and distributes random elements that the two parties use during the online phase (during which the trusted initializer is absent). They also present a version of their work without a trusted initializer which makes the pre-computation phase longer. Their solution could be generalized to multiple parties wishing to train a linear regression on their combined dataset. The same authors presented later in [Coc+20] a two-party federated logistic regression algorithm that works for any data separation and that uses a trusted initializer.

the solutions presented in [MZ17] consist of new and effective protocols for preserving privacy in machine learning for linear and logistic regression. These protocols fall under the two-server model, in which data owners divide their private data between two non-colluding servers to train various models on the combined data via secure two-party computation.

The authors of [Zha+22] propose a unified federated learning framework that reconciles horizontal and vertical federated learning. Based on this framework, the authors formulate and quantify the trade-offs between privacy leakage, utility loss, and efficiency reduction. Previous solutions relied on cryptographic techniques; in 2007, the authors of [SNT07] used secure multi-party computation protocols for additions and matrix multiplications in a semi-honest security setting for horizontal and vertical partitioning. The logistic regression is computed through Newton-Raphson iterations.

Another tool used to compute a federated logistic regression is transfer learning [PY10], which is a machine learning technique where a model developed for a task is reused as the starting point for a model on a second task. It is used to save time and resources by reusing previously developed models and knowledge. Transfer learning allows the reuse of knowledge and data and is extremely useful when the amount of data to train a new model is limited. The paper [Gao+19] uses transfer learning to construct a secure, efficient, and scalable end-to-end privacy-preserving multi-party learning approach for both vertical and horizontal partitioning of data. They use secure multi-party computation and homomorphic encryption with an honest-but-curious adversary. Through their experiments on five benchmark datasets, they prove the effectiveness of this approach and also demonstrate its real-world applicability in in-hospital mortality prediction.

The paper [Aga+22] proposes a new way to securely train a logistic regression model on secret shared data. Their aim is to reduce online communication and the number of rounds needed, while still making an efficient offline phase possible. They introduce a new way to approximate the sigmoid function that leads to a secure evaluation protocol with better communication, secure spline

evaluation and secure powers computation protocols for fixed-point values, and a new comparison protocol that reduces online communication. They test their functions on a two-party logistic regression.

In the reference [Pen+22], the authors present a practical and efficient adaptation of distributed logistic regression learning that adds tractable privacy guarantees against model inversion, even in the absence of a trusted curator. They use secure multi-party computation and output perturbation. Their design is general, making no assumptions about the data partitioning scenario, the number of computing parties i.e. data owners, or the security setting in which it is applied (semi-honest or malicious). Moreover, it is easy to replace the noise mechanism with another one for scenarios where weaker privacy guarantees like ϵ, δ-differential privacy are acceptable. The work in this thesis on logistic regression, which has been published in [SKB22] is very similar to [Pen+22] except that we use objective perturbation differentially private technique instead of the output perturbation one. Even though we say that our work is for vertically split datasets, our methods can be applied to horizontally split data by changing the concatenation orientation. In their work, [Pen+22] says that in both horizontally and vertically split data scenarios, the approach of training the data on secret-shared data with differential privacy is preferable because it yields a higher accuracy, which becomes even more evident when the data is distributed among multiple data owners.

Truth Finding

Truth-finding or truth-discovery is a data mining algorithm whose objective is uncovering facts in order to gain an understanding of the truth. It involves collecting answers to queries from sources and analyzing them in order to draw conclusions about the true answer to the query. It enables practitioners to arrive at an accurate understanding of events and circumstances and can be used in a variety of ways, including in court cases, or business affairs.

Naive majority voting can be considered a truth-finding algorithm because multiple sources would give their opinion on the veracity of a statement, and if the majority finds the statement to be true then it would be considered as such. The early works in truth-finding found in [YHY08; Gal+10], show that majority voting is not the best solution to corroborate data when different sources provide conflicting information on it. Particularly in [Gal+10] three alternative algorithms are given. Interestingly, further studies [Li+12; Ber15] show that no single truth-finding algorithm performs well in all scenarios and benchmarks. In this thesis, we focus on achieving privacy in two of the three pioneering algorithms given in [Gal+10], they are called Cosine and 3-Estimates.

Since their introduction, cryptographic privacy-preserving tools like secure multi-party computation and homomorphic encryption have been used for federated tasks, including truth-finding. The current state-of-the-art multi-party computation protocols used in platforms already mentioned before like ABY^3 [MR18] and others specific to machine learning tasks [Kno+21; Ryf+18] could be used to compute functions securely, robustly, and efficiently. Alternatively, homomorphic encryption - also used for machine learning tasks like [Zam21] as mentioned earlier - can be used for scenarios where no communications take place during the computations; only one party needs to be doing them.

Concerning applications similar to truth-finding, cryptography has been used for e-voting: for example, secure multi-party in [NBK15] and homomorphic encryption in [Chi+16]. Both tools have even been combined in [Jun+13] in order to achieve a privacy-preserving aggregation without secure communication channels. These results are for simple majority voting and do not consider other complex truth-finding algorithms, which are usually more complex than performing a simple vote count.

Privacy-preserving truth-finding algorithms were not common until 2015 with [Mia+15] and

afterward with [Zhe+20; ZDW18] and other similar works; all these works consider the same specific problem of mobile crowd-sensing systems, and they all use homomorphic encryption and one specific truth-finding algorithm: CRH [Li+16]. The work presented in this thesis is more general, it proposes secure multi-party computation protocols and implementation techniques that can be applied to various truth-finding algorithms. To our knowledge, secure multi-party computation has not been used to securely evaluate truth-finding algorithms.

Restricted Boltzmann Machines

Neural networks for classification [FC02] are a type of machine learning algorithm that can be used to classify data into different categories. Neural networks work by combining a large number of input variables (features) and using them to identify patterns that can be used to predict an output (class). They are used in a variety of applications such as image recognition, text classification, and handwriting recognition.

Restricted Boltzmann machines [FI12] is a type of generative artificial neural network used for unsupervised learning. They work by creating a probabilistic graphical model that can learn a probability distribution over its inputs. They use a two-layer architecture, consisting of visible and hidden layers of neurons that are connected to each other, with the hidden neurons receiving input from the visible neurons and vice versa. The connections between the neurons are weighted and the weights are adjusted during training. The network is trained by adjusting the weights to maximize the probability of observing the correct output for a given input. This guide [Hin12] is an effort by the machine learning team at the University of Toronto to share their experience in training restricted Boltzmann machines using the contrastive divergence learning procedure which is used to usually train energy models.

One of the first privacy-preserving restricted Boltzmann machines is presented in [SMH07; LZJ14]. It can be used to exchange information without disclosing personal data from different organizations. They evaluate the accuracy and effectiveness of their algorithms and show that the accuracy of their approach is very close to that of the original model. In their privacy-preserving scheme, they use the ElGamal encryption scheme [Gam85] - which is a typical public encryption method with a homomorphic property - in a semi-honest setting.

The paper [NHC22] introduces the first endeavor to develop private generative machine learning models based on neural networks. Specifically, their ϵ, δ-differentially private generative model enables the production and distribution of secure high-dimensional data with guaranteed privacy. Differential privacy is applied to the input layer of data processing as a form of data synthesis that is kept private. In other words, this solution works by either hiding or changing sensitive features of the data, while still maintaining its utility for deep learning models. This data or generative model can then be used for various analyses, protecting the privacy of sample-level participants.

Frameworks for Privacy-preserving Machine Learning

Gazelle [JVC18] is a secure, scalable, and low-latency system for neural network inference, that combines homomorphic encryption and traditional two-party computation techniques (like garbled circuits). Their contributions include a homomorphic encryption library with efficient implementations of basic operations, homomorphic linear algebra kernels to map neural network layers to optimized matrix-vector multiplication and convolution routines, and optimized encryption switching protocols for seamlessly converting between homomorphic and garbled circuit encodings for complete neural network inference. Their work satisfies the cryptographic standard of ideal (real) security.

Another privacy-preserving framework is PySyft [Ryf+18], we use it in this thesis for some implementation. This work presents the first reliable, general framework for privacy-preserving deep learning, built over PyTorch. The framework enables the use of federated learning, secure multiparty computation, and differential privacy from an intuitive interface, while still exposing a familiar deep learning API to the end-user. What makes the framework easy to use is the tensor implementations support commands of the PyTorch API and are combined with secure multi-party computation and differential privacy functionalities within the same framework.

In 2018, [MR18] provided a general framework for privacy-preserving machine learning - called ABY^3 - which is used to create solutions for training linear regression, logistic regression, and neural network models. This framework is based on a three-server model where data owners divide their data among the three servers before models are trained and evaluated on the joint data using three-party computation. At the same time, [Ria+18] presented Chameleon, a framework with mixed protocols for secure function evaluation which enables two parties to jointly compute a function without disclosing their private inputs. Their work was both scalable and significantly more efficient than the ABY framework from [DSZ15] on which it is based.

Two works emerged the next year, the first one [WGC19] provided three-party secure computation protocols for a variety of neural network building blocks such as matrix multiplication, convolutions, rectified linear units, maxpool, normalization, and more. With this, they could construct three-party secure protocols for training and inference of multiple neural network architectures while ensuring that no single party learns any information about the data. The second one is QUOTIENT [Agr+19], an approach to training deep neural networks with tailored secure two-party protocols, their work was more efficient in terms of speed and accuracy compared to the existing frameworks.

Falcon [Wag+21] introduced a three-party protocol that enables efficient private training and inference of large machine learning models. It offered many key benefits compared to the existing art for private inference. On average, it is 8 times faster than SecureNN [WGC19] and ABY^3 [MR18]. Being more communication efficient, Falcon is also 6 times faster than SecureNN for private training, 4.4 times faster than ABY^3. They implement their protocols in both semi-honest and malicious settings.

Lastly, in 2021 CrypTen [Kno+21] was introduced, it is a software framework that makes the secure multi-party computation primitives accessible through abstractions which are regularly used in modern machine-learning frameworks, like tensor computations, automatic differentiation, and modular neural networks.

Finally, Zama is a homomorphic-encryption-based platform that allows users to perform computations on private data, and they have a library [Zam21] that is specific to machine learning tasks.

1.3 Personal Contributions

The contributions of this thesis are specifically on the use of secure multi-party computation in machine learning and data mining algorithms in order to be able to run these algorithms on combined databases owned by multiple players without having any of the players reveal their data to anyone else.

They mainly include the development of an ϵ-differentially private logistic regression algorithm trained on a vertically split dataset. We empirically evaluate the performance of the proposed mechanism and comparatively argue its originality and relevance in light of related work. This contribution is distinct from prior related works as it combines for the first time three privacy

specifications: first, passive security [CDN15] as per definition 3, which we achieve by using exclusively secret-sharing based multi-party computation for the training process; second, the model output satisfies ϵ-differential privacy instead of ϵ, δ-differential privacy, and third, the functional mechanism [Zha+12], an objective perturbation-based mechanism, is used which yields an output model with good accuracy. In addition, we present protocols and algorithms based on secure multi-party computation that could be applied in other machine learning and data mining algorithms. In particular, we present a pseudo-equality algorithm that can be used in a type of data analysis algorithm called truth-finding – which could be seen as a generalization of voting algorithms. Our contribution would allow multiple players to securely compute the truth of a given statement without revealing any confidential data or compromising the privacy of any player's model. Previous works in this area were based on homomorphic encryption.

All the implementations and experiments that we run serve as proof of concepts. Further cryptographic engineering and refined implementation techniques could dramatically improve their performances.

The contributions are highlighted in this section and then explained in detail in the subsequent chapters.

1.3.1 Differentially Private Logistic Regression

Use-case: Logistic regression is a type of supervised machine learning algorithm that is used for predicting the probability of a categorical dependent variable (binary outcomes). When training a logistic regression on a divided dataset, where each section is owned by a distinct entity called "player", it is critical to ensure that each player's "secret partial dataset input" is kept private. The two players' goal is to obtain the "public model output" trained on the "secret combined dataset" as input – that none of the players fully knows – while keeping the patients' information private.

Privacy specifications: Our double criterion for privacy is:

1. The computation of the public model output should have passive security as per definition 3. Very roughly, a player A should not learn information on the partial dataset input of another player B, except what A could deduce from his own partial dataset input and the public model output.

2. The public model output should satisfy ϵ-differential privacy with respect to the combined dataset input.

On one side, cryptography tools like secret-sharing-based secure multi-party computation and homomorphic encryption are used in many prior works allowing the players to train a logistic regression on their secret combined dataset without revealing their secret partial dataset.
On another side, the mechanisms of ϵ, δ-differential privacy and the better privacy guarantee mechanisms of ϵ-differential consist of noise-adding techniques that ensure that the model cannot be used to infer information about an individual record present in the secret combined dataset.

Limitations of existing solutions:

A lot of secure multi-party computation frameworks have applied their protocols to a logistic regression like [MR18; WGC19; MZ17] and other aforementioned libraries. However, at this point differential privacy is not satisfied. Some of the works that consider differential privacy satisfy only ϵ, δ-differential privacy, like [Taj+20; Wan+20]. In [Jay+18], the authors present two algorithms, the first one satisfies ϵ, δ-differential privacy and the second one satisfies ϵ-differential privacy.

However their results are limited to horizontal dataset separations. In [BP20], the authors also present an algorithm that satisfies ϵ-differential privacy for horizontally separated data.

The contribution [Pen+22] satisfies our privacy definition by using output perturbation as a differentially private mechanism, whereas we use objective perturbation from [Zha+12]. The difference between the two approaches has been previously explained in section 1.2.2.

Multi-key fully homomorphic encryption (MKFHE) [LTV12] is an alternative method to do secure multi-party computation. The state of the art [Che+19] reports speed performance 100 to 1000 times slower than secret-sharing-based secure multi-party computation. The computational complexity of their multi-key fully homomorphic encryption is so far quadratic in the number of input providers. However, a very recent preprint [KÖA23] announces to have lifted this restriction. Independently, fully homomorphic encryption is an efficient way to pre-process the "correlated randomness" used in secret-sharing-based secure multi-party computation [Che+20]. Consequently, selecting the most advantageous approach requires weighing the environment – including the players executing the algorithm – a clearly defined privacy assumption, the desired outcome, and the allocated time for the algorithm.

This thesis' contributions in regard to preserving privacy in logistic regressions are demonstrated in the comparative table 1.1.

We highlight that the table 1.1 is insufficient to ascertain which proposed technique or strategy is the preeminent one, as the circumstances vary. It is evident that employing differential privacy reduces the accuracy of the model, while homomorphic encryption lengthens the computation time.

Contribution: This contribution – published in [SKB22] – stands out from prior related works as it combines for the first time three privacy guarantees:

- We satisfy passive security as per definition 3 – achieved by using exclusively secret-sharing-based multi-party computation.
- The public model output has ϵ-differential privacy.
- The public model output has good accuracy due to the usage of functional mechanism [Zha+12; CMS11].

Note that while the computations are done in a secret-sharing-based manner, only the public model output is to be revealed. However, it is not necessary to reveal it, it could be kept secret-shared between the two parties and be used to make predictions using protocols, in this case, the players should both agree on making the inference.

Technical details: Our extensive experiments have demonstrated that the combination of secure multi-party computation and differential privacy provides comparable results to those of differential privacy alone. As previously mentioned, despite being principally intended for datasets that have been vertically split, our techniques can also be applied to those that have been horizontally divided by simply reconfiguring the method of concatenation. This is made possible by the underlying architecture which involves secret-sharing each player's secret partial dataset input, combining them into a secret combined dataset through concatenation, and then training the model – although this process is quite communication-intensive, it remains compatible with any form of dataset division. In conclusion, this mechanism has been demonstrated to be a powerful, efficient, and secure privacy-protecting solution. We elected to carry out a logistic regression as it serves as an essential basis for more intricate machine-learning models. Indeed, the results of this contribution can be utilized to develop privacy-preserving multi-layer perceptrons – i.e. binary classifiers with multiple layers –

Table 1.1: A comparative table of related works on privacy-preserving logistic regressions. The table shows the dataset separation, and the kind of noise that was injected into the algorithm in case differential privacy (DP) was used. Functional mechanism [Zha+12] is denoted by FM, MPC stands for secret-sharing-based secure multi-party computation, HE for homomorphic encryption, and FE for functional encryption. The last column indicated whether or not our defined privacy specifications are satisfied or not. The first criterion of sorting the papers is the separation, then the differential privacy level.

Reference	**Separation**	**DP**	**Noise**	**Other tools**	**Specifications**
[Sri+14]					
[Kan+20]		ϵ, δ	Input		
[Zha+17]		ϵ, δ	Output		
[Zha+12]		ϵ	Objective		
[Nas+20]	Horizontal				
[GTJ22]	Horizontal			MPC	
[Jay+18]	Horizontal	ϵ, δ	Gradient	MPC	
[Jay+18]	Horizontal	ϵ	Output	MPC	Satisfied
[BP20]	Horizontal	ϵ	Output	MPC	Satisfied
[San+04]	Vertical			MPC	
[Har+17]	Vertical			HE	
[Xu+21]	Vertical			FE	
[JZG22]	Vertical			MPC	
[Taj+20]	Vertical	ϵ, δ	Gradient	MPC	
[Wan+20]	Vertical	ϵ, δ	Gradient	MPC	
[XYW21]	Vertical	ϵ	FM	MPC, HE	
[SKB22]	Vertical	ϵ	FM	MPC	Satisfied
[Zha+22]	Any				
[SNT07]	Any			MPC	
[MZ17]	Any			MPC	
[MR18]	Any			MPC	
[Dam+19]	Any			MPC	
[Esc+20]	Any			MPC	
[Gao+19]	Any			MPC, HE	
[Pen+22]	Any	ϵ	Output	MPC	Satisfied
This thesis	Any	ϵ	FM	MPC	Satisfied

and neural networks. In chapter 4 we elaborate upon our contribution, explaining the architecture we propose.

1.3.2 Confidential Truth Finding with Secure Multi-Party Computation

Use-case: Truth-finding algorithms can be used to corroborate data from disagreeing sources. These algorithms help to find the truth value of a query by comparing and analyzing multiple sources' answers on a set of queries. As this process involves the sharing of confidential information, there is a need for protocols that address the issues of confidentiality and privacy. Majority voting can be considered a type of truth-finding algorithm. For example, a creditor may wish to determine whether loan applicants are creditworthy. The creditor would want to base their decision on the different, possibly disagreeing, evaluations of the applicants by several financial institutions. However, financial institutions only agree to contribute their respective evaluations to the decision provided it is neither revealed to the creditor, to the other financial institutions, nor to third parties. For such a purpose, we turn to secure multi-party computation [CDN15].

Privacy specifications: Our criterion for privacy is the same as the first criterion from section 1.3.1, which means that the computation of the output should have passive security [CDN15] as per definition 3.

Limitations of existing solutions: A related work would be privacy-preserving majority voting [Chi+16; NBK15] as majority voting can be considered a truth-finding algorithm because multiple sources would give their opinion on the veracity of a statement. More complex truth-finding algorithms found in [YHY08; Gal+10], demonstrate that majority voting is not the best way to corroborate data when different sources provide conflicting information about it. Privacy-preserving tools for these truth-finding algorithms were not common until 2015 with [Mia+15] and afterward with [Zhe+20; ZDW18] and other similar works; all these works consider the same specific problem, and they all use homomorphic encryption and one specific truth-finding algorithm: CRH [Li+16]. The work presented in this thesis is more general, it proposes secure multi-party computation protocols and implementation techniques that can be applied to various truth-finding algorithms. To our knowledge, secure multi-party computation has not been used to securely evaluate truth-finding algorithms.

Contribution: In our suggested method [SSB23], the sources secret-share their votes on two servers that execute the truth-finding algorithm using secret-sharing-based two-party computation protocols. The number of servers could be higher, but we stick to two players. We present a pseudo-equality test that we use to compute additions depending on a condition, a recurring operation in the truth-finding algorithm. Our pseudo-equality test is fast as it corresponds to a degree-two polynomial evaluation. To evaluate the performance of the outlined scheme, two state-of-the-art truth-finding algorithms, Cosine, and 3-Estimates, are computed in a secure multi-party computation manner, and then the results are compared with those of plain algorithms (without cryptography).

Technical details: The main key algorithm behind this contribution is an algorithm for secure equality tests. Instead of comparing two elements using general multi-party equality tests, we make use of the fact that the elements that are compared are in a set of three elements. We, therefore, define a polynomial of degree two that is equivalent to an equality test in our use case. The

approach also uses existing secure multi-party computation-friendly approximations of operations like real division and square root based on numerical methods. However, our extensive experiments have demonstrated that using secure multi-party computation provides results comparable to plain algorithms. In chapter 5 we explain further our contribution, and the the approach we have proposed.

1.3.3 Secret-sharing-based Primitives for Machine Learning and Data Mining

Use-case: Secret-sharing-based computing primitives for machine learning are a set of protocols and algorithms that can be used to securely share data and train models with different players. This contribution is an analysis that allows an understanding of how different parameters and algorithms -constituting the secure multi-party computation framework, especially for machine learning tasks- affect the computation time and results.

Privacy specifications: We use the same privacy criterion as the one specified in section 1.3.1, where the computation of the output must have passive security as per definition 3.

Limitations of existing solutions: Many works present primitives for secure multi-party computation in machine learning applications,
like [CS10; Esc+20; MZ17; WGC19]. In addition, there are many frameworks that are compatible with machine learning frameworks as TF-encrypted [Dah+18] based on tensorflow [Dil+17], and PySyft [Ryf+18] based on PyTorch [Pas+19]. The aim of this contribution is not to rival these frameworks, but rather to offer a way to experiment and compare different secret-sharing-based computation algorithms in a local and easily configurable platform.

Contributions: This contribution consists of secret-sharing-based secure two-party computation primitives for machine learning applications. We explain how secure multi-party computation affects the precision of the results, and we give a protocol for pseudo-equality tests when the input is in a publicly defined finite set. The pseudo-equality test makes computations faster machine learning algorithms like restricted Boltzmann machines for example as explained in 6.2.3.2. We implement the protocols that we propose in a socket network where one could test different protocols, and execute algorithms in a secure multi-party computation manner. First, we build a low-level framework using a local socket network. The framework allows fine-grained control over the parameters such as the ring size, the number of bit precision, and the algorithms that one would like to use. Second, we present comparative experimental results to see how these parameters affect the computation process in terms of the time of execution and the correctness of the output.

Technical details: Since secure multi-party computation protocols are implemented in a finite ring or field, whereas machine learning and data mining algorithms are evaluated on real numbers, the real numbers are approximated with a fixed-point precision and then mapped to a finite ring as it is explained in section 3.1.1. The map between the real subset and the finite ring is not a ring homomorphism, meaning that the multiplication of the images of two elements does not correspond to the image of the multiplication of these elements. This results in not having a trivial algorithm to go back and forth between the two structures. To this end, the first contribution of this thesis is a study on the impact i.e. the error resulting from using pseudo-additions and pseudo-multiplications with fixed precision.

Machine learning and data mining applications use comparison. In plaintext, comparisons are easy to compute, it suffices to check the most significant bit. In secret-sharing-based secure

multi-party computation checking the most significant bit of each player does not achieve this. Hence the need for comparison protocols, and there are many such protocols. More information about comparisons is given in section 3.2. An existing and well-performing comparison protocol is RABBIT from [Mak+21]. We give a new algorithm for a fixed-point-based numerical approximation of the sign function. The algorithm takes a secret shared element corresponding to a real number in the interval $[-1, 1]$ and returns the positivity value with a certain error. This comparison can be applied in scenarios where the real number is in this interval for example when generating a Bernoulli random variable given a uniformly random element. We also compare the existing secure multi-party algorithms that implement common "activation functions" which are used to introduce non-linearity into the machine learning algorithms, allowing them to learn complex mappings between inputs and outputs.. We also compare algorithms for activation functions based on existing tools that we mix and match in order to evaluate the performance of each combination. We evaluate the performance of each function by measuring its time of execution and by using it to train machine learning algorithms. In chapter 6 we elaborate upon our contribution, elucidating the architecture we have proposed.

1.4 Thesis Outline

This thesis begins with an overview of the relevant background information, followed by an exploration of the contributions made.

More precisely chapter 2 reviews the information to understand this thesis. It contains everything that is required to know about secure multi-party computation and secret sharing and cryptographic techniques that allow computing on secrets as this is a key concept to preserve the privacy of data. We explain the basic secure multi-party computation protocols, such as additions and multiplications and we give the security definitions that will be used in this thesis.

Subsequently, we review machine learning and data mining algorithms that we use in this thesis, like the logistic regression algorithm and two truth-finding algorithms.

In chapter 3 we explain how secure multi-party computation and privacy mechanisms are applied in the context of machine learning. More specifically, we explain the concept of using fixed-point precision and mapping real elements to a finite ring. In addition, we formally define differential privacy and homomorphic encryption.

Chapter 4 depicts a differentially-private logistic regression based on secure multi-party computation. We begin by explicitly defining the problem and recalling the related existing works. Then we explain our approach and construction in detail. Indeed we define how we evaluate the sigmoid function -which is an activation function used for logistic regression- in a secure multi-party-friendly manner, and how we generate random elements for the differential privacy mechanism. We conclude by comparatively testing our approach on two datasets.

In chapter 5 we devise and present an algorithm, and variants thereof, for privacy-preserving truth-finding algorithms, using secure multi-party computation. After explicitly defining the problem, we explain recall the preliminary definitions of truth-finding algorithms evaluated without secure multi-party computation and the prior related contributions. We then propose our approach, and more particularly an algorithm that replaces equality tests by a degree-two polynomial. We also present modifications that could be done to the baseline algorithm in order to render it less multiplication-costly which should make the algorithm more efficient when evaluated using multi-party protocols. We conclude by testing our approach and the variants thereof and comparing the results to a baseline.

Finally, in chapter 6 we develop the third contribution of this thesis, a family of secure multi-

party computation protocols and algorithms that can be used in machine learning. We first write a proof for two-party circuit evaluations with passive security using additive secret-sharing. We do so because the subsequent results will be tested in a two-party setting using additive secret-sharing. Secondly, we present an analysis of the error induced by secure multi-party computation with fixed-point precision and the two-party truncation protocol that we will be using. Thirdly, we present an algorithm that implements the sign function using numerical methods and a generalization of the equality test with a polynomial used in truth-finding. We also explain how this equality test alternative on finite sets could be used in restricted Boltzmann machines. We conclude by comparing the time of execution and the correctness of activation functions and the other algorithms presented in this chapter.

Chapter 2

Background

In this section, first will be explained preliminaries about secret-sharing-based secure multi-party computation in section 2.1, then the basic protocols to do MPC, in section 2.2. In section 2.3, we will introduce the notions of machine learning and some algorithms that we will use to explain how privacy-preserving techniques could be applied. At the end of this chapter, in section 2.4, we will explain a differential privacy technique called functional mechanism, that we use in this thesis.

2.1 Computing on Secrets

Since the beginning secure multi-party computation aimed to be the solution for computing sensitive data. In this section, we explain how it was defined and then how it evolved, we also explain the difference between secure two multi-party computation methods, one based on garbled circuits, and the other on secret-sharing schemes.

2.1.1 Problem Definition

We consider a finite field or ring $\mathbb{F} = \mathbb{Z}/q\mathbb{Z}$ of size $q \in \mathbb{N}^*$, $n \in \mathbb{N}$ parties called players, denoted by $P_1, P_2, ...P_n$, and a function $f(x^1, ..., x^n)$ where each $x^i \in \mathbb{F}$ is a confidential element held by the player P_i for $1 \leq i \leq n$. In this thesis, however, we only consider the case where $\mathbb{F}$ is a finite ring, and more precisely in the form $\mathbb{Z}/2^l\mathbb{Z}$, with l being a positive integer.

A traditional, non-cryptographic approach could be trusting a third party. That is having each player send their input to the third party who computes the function and sends back the output. Even with adding noise to the input, this solution would still leak information to the trusted party. The goal of secure multi-party computation or other cryptographic approaches, in general, is to decentralize this model and let the players learn the output of the function f, $f(x^1, .., x^n)$, without any player P_i revealing their input x^i to another player P_j.

2.1.2 Secure Multi-party Computation and Secret Sharing

Secure multi-party computation (MPC) allows the set of players to compute the value of $f(x^1, .., x^n)$ while keeping their input private without a trusted third party. This is done by evaluating the arithmetic or binary circuit of the function collaboratively, node by node.

In the history of secure multi-party computation, many approaches were considered to solve the problem. In this thesis, we only consider secret-sharing-based secure multi-party computation (which will be explained later), but we also explain garbled circuits, the original version of secure

multi-party computation. Explaining the idea behind the garbled circuit illustrates how differently secure computations could be performed.

2.1.2.1 Garbled Circuits

Garbled circuits [Yao86] are the first form of secure multi-party computation, and more specifically two-party computation. They rely on two other cryptographic primitives; symmetric encryption and oblivious transfers. Their goal is to evaluate a boolean circuit with XOR and AND gate operations. Suppose the two players are P_1 and P_2.

The first step is to let player P_1, called the garbler or generator, garble the circuit of the function the players wish to evaluate. The circuit is public because the function to be evaluated is public. To garble the circuit, for each gate P_1 has to do the following:

1. Generate two random-bit strings, these strings correspond to the possible input of the following gate (0 and 1).
2. Encrypt the possible outcomes of the four possible inputs using the random-bit strings as keys.
3. Shuffle the garbled table.

Player P_1 then sends the garbled tables to P_2 along with keys that correspond to P_1's inputs. In order to calculate the circuit, P_2 needs to know the keys that correspond to their own input as well. To this end, P_1 needs P_2 need to collaborate because only the garbler knows which keys correspond to which input. The keys are obtained through oblivious transfers, which will allow P_2 to evaluate the circuit and obtains the correct encrypted outputs. Finally, P_1 and P_2 communicate to learn the output of the boolean circuit they wished to evaluate.

2.1.2.2 Secret Sharing Based Protocols

With [BGW88], the secret-sharing-based secure multi-party computation was developed. The main secret-sharing schemes used are Shamir's and the additive scheme. A secret sharing scheme allows a player to distribute a fragment of their secret, called share, to every other player including themselves. Each share, alone, reveals no information about the secret. However, together, the players $P_1, ..., P_n$ could reconstruct the secret by revealing their respective share of the secret. A formal definition of secret sharing will be given in section 2.2.1, but first, we illustrate how to evaluate a function's arithmetic circuit using a secret sharing scheme:

1. Each player secret-shares their input.
2. Given an addition gate of the arithmetic circuit and shares of the inputs of the gates, the players use a secure multi-party computation protocol to compute shares of the addition of the inputs.
3. Given a multiplication gate of the arithmetic circuit and shares of the inputs of the gates, the players use a secure multi-party computation protocol to compute shares of the multiplication of the inputs.
4. After evaluating all the gates, the players end up with shares corresponding to the final gate. These shares are used to reconstruct the output of the final gate.

In this thesis, we will only consider secure multi-party computation protocols based on additive secret sharing. However, we explain briefly Shamir's secret sharing scheme and protocols to give an idea of how differently secure multi-party computation could be done.

Going forward in section 2.2 we assume that secrets and shares are in $\mathbb{Z}/2^l\mathbb{Z}$.

2.2 Basic Secret Sharing Based MPC Protocols

2.2.1 Definitions

Let $l \in \mathbb{N}^*$, and $\mathbb{Z}/2^l\mathbb{Z}$ a finite ring.

Definition 1 (Secret Sharing Scheme [CDN15]). *Let $t \in \mathbb{N}^*$ and $t < n$, a (t, n)-secret sharing scheme, is a mechanism in which an element $s \in \mathbb{Z}/2^l\mathbb{Z}$ is "distributed" in a sequence $s_1, ..., s_n \in \mathbb{Z}/2^l\mathbb{Z}$ - each s_i is called "share" of s - such that $m \leq t$ shares alone do not reveal any information about the secret (t-privacy), whereas $m \geq t+1$ shares allow reconstructing s ($(t+1)$-reconstruction).*

Property 2 (Linearity). *We say that a secret sharing scheme is linear if for any two secrets $s, s' \in (\mathbb{Z}/2^l\mathbb{Z}, +, \cdot)$ and a constant $\alpha \in \mathbb{Z}/2^l\mathbb{Z}$ we have:*

1. *The sum $s_i + s'_i$ is a share of $s + s'$.*
2. *The scalar multiplication αs_i is a share of αs.*

Secret sharing and secure multi-party computation, in general, require generating random fields or ring elements. To this end, we denote by $\mathcal{G}_{\mathbb{Z}/2^l\mathbb{Z}}(n)$ a function that generates n uniform pseudo-random elements in $\mathbb{Z}/2^l\mathbb{Z}$.

We say that an adversary is semi-honest (or passive) if the adversary has access to the messages seen by a certain number of players that strictly follow the secure multi-party computation protocol to the letter and do not deviate from the protocol.

Definition 3 ($n-1$ Passive security [CDN15]). *Let Π be a protocol, f an arithmetic function, and $P_1, ..., P_n$ be the players wishing to respectively learn $y^1, ..., y^n$ where $(y^1, ..., y^n) = f(x^1, ..., x^n)$, for any secret input x^i of P_i.*

Let C be the set of passively corrupted players.

Denote by $view_j = \{m; m$ is a message seen by P_j during $\Pi\}$, these messages could be messages received by other players, or random elements to which the player has access.

We say that Π evaluates f with $n-1$ passive security if there exists an efficient probabilistic algorithm S, also called simulator such that:

$$S(\{x^j, y^j\}_{P_j \in C}) \equiv \{view_j\}_{P_j \in C}, \tag{2.1}$$

where $\equiv$ means that the two probability distributions are indistinguishable for any polynomial machine.

Remark 4. *For $n = 2$, we say "passive security" instead of $n-1$ passive security as per definition 3.*

Note that indistinguishability between the probability distribution of $S(\{x^j, y^j\}_{P_j \in C})$ and $\{\text{view}_j\}_{P_j \in C}$ for a computationally unbounded machine is not dealt with in this thesis.

Algorithm 1 Additive secret sharing protocol Π_{share}

Private input: P_i's secret x^i
Output: Each player holds a share of P_i's secret
1: P_i computes $x_2^i, ..., x_n^i \leftarrow \mathcal{G}_{\mathbb{Z}/2^l\mathbb{Z}}(n-1)$
2: $x_1^i \leftarrow x^i - \sum_{j=2}^{n} x_j^i = x^i - x_2^i - x_3^i - ... - x_n^i$
3: P_i sends x_j^i to P_j for $j = 1...n$

2.2.2 Additive Secret Sharing Scheme

Additive sharing consists of choosing shares that add up to the secret. Suppose player P_i wants to share his secret $x^i \in \mathbb{Z}/2^l\mathbb{Z}$ with the rest of the players using the additive scheme, algorithm 1 denoted as Π_{share} allows that.

Lemma 5. *The additive scheme in algorithm 1 is $(n-1)$-private as per definition 1 and linear as per property 2.*

Proof. **Linearity:** Let s, s' be two additively secret-shared elements. Then each party P_i owns the shares s_i and s'_i. Set $z_i = s_i + s'_i$ then upon reconstruction of z the players obtain:

$$z = \sum z_i = \sum s_i + s'_i = \sum s_i + \sum s'_i = s + s'.$$

For scalar multiplication, the proof is similarly done.

$(n-1)$**-privacy:** Take a set of $n-1$ shares distributed by P_i to the other players.

If x_1^i is not in the set then the set does not reveal any information about x^i because the shares were generated independently of x^i in line 1 of the algorithm.

If x_1^i is in the set then the sum of the elements in the set is equal to $x^i - x_j^i$ with $j \neq 1$. However x_j^i was randomly generated in $\mathbb{Z}/2^l\mathbb{Z}$ which means that the set reveals no information about x^i. □

Note that once P_i generates shares of x^i, P_i sends each share x_j^i to every player P_j including himself.

Definition 6. *We define $\left[x^i\right]$, where $x^i \in \mathbb{Z}/2^l\mathbb{Z}$:*

$$\left[x^i\right] = (x_1^i, ..., x_n^i),$$

where $x_1^i, ..., x_n^i$ are the additive shares of the x^i that can be computed as described in algorithm 1.

When the players finish evaluating the arithmetic circuit of the function they wish to compute, they have to reconstruct the element on the final node. To do this the players just have to send their share of this element to each other, and each player adds up the received shares. This way all the players learn the output of the function.

This final step is not mandatory, it depends on the players' objective. For example, they might wish to send the result to only one of the players, or to a third party who would be the only one learning the output. In more advanced applications, the players might want to keep the final result secret-shared and use it for other purposes. In addition, a reconstruction protocol might be used during the evaluation of the arithmetic circuit. We denote the reconstruction protocol by Π_{open} which takes a secret shared element as input and returns the value of the secret to all the players, i.e. for a secret $[s]$, $\Pi_{\text{open}}([s]) = s$.

Definition 7 (Round [CDN15])**.** *In secure multi-party computation, the term round is used. Each round consists of an "inward clocking" which allows a player to send messages and an "outward clocking" which allows the player to receive messages.*

Algorithm 2 Reveal protocol Π_{open}

Private input: Shares $y_1, ..., y_n$ of the secret $[y]$; P_i holds y_i
Output: y
1: Every player P_i sends y_i to every other player
2: Every player computes $y \leftarrow \sum y_i$

Addition in Secure Multi-party Computation

Consider that before an addition gate the players have shares of two secrets $[x]$ and $[y]$. This means that each player P_i has x_i and y_i and that

$$\sum_{i=1}^{n} x_i = x \text{ and } \sum_{i=1}^{n} y_i = y.$$

The goal of the addition protocol is to obtain a sharing of the sum of $x + y$. More specifically, each player P_i should end up with a ring element z_i such that

$$\sum_{i=1}^{n} z_i = x + y.$$

Algorithm 3 Addition protocol Π_{add}

Private input: P_i holds x_i,y_i for i in $\{1, ..., n\}$
Output: P_i holds z_i for i in $\{1, ..., n\}$
1: P_i computes $z_i \leftarrow x_i + y_i$ for i in $\{1, ..., n\}$

Lemma 8. *Algorithm 3 outputs additive shares of $x + y$ without any communications between the players.*

The proof of lemma 8 is very straightforward, it is due to the linearity of the additive secret-sharing scheme. In addition, the protocol cannot reveal information about the secrets because there is no communication between the players.

Because an addition requires no communication between the player, we say that additions are free, or local. Other operations can be obtained for free in secure multi-party computation using an additive secret-sharing scheme. The first operation is multiplication by a constant. Given a secret-shared element $[x]$ and a public constant α, suppose the players would like to obtain shares of $[\alpha x]$. Then all the players can locally multiply their share of x by the constant and obtain shares of the desired result. The second free operation is the addition of a public constant to a secret, in this case only one player adds this constant to their share of the secret. The protocol for public constant multiplication can be denoted by Π_{cmul}, and the one for an addition with a public constant by Π_{cadd}.

Finally, a protocol for subtraction denoted by $\Pi_{\text{sub}}(x, y)$ which subtracts $[y]$ from $[x]$ can be defined as:

$$\Pi_{\text{sub}}([x], [y]) = \Pi_{\text{add}}([x], \Pi_{\text{cmul}}([y], -1)).$$

Note that this protocol is also local.

Multiplication in Secure Multi-party Computation Dubbed as $[x], [y] \to [x] \cdot [y]$

Consider that before a multiplication gate the players have shares of two values $[x]$ and $[y]$. This means that each player P_i has x_i and y_i and that

$$\sum_{i=1}^{n} x_i = x \text{ and } \sum_{i=1}^{n} y_i = y.$$

The goal of a multiplication protocol is to obtain a sharing of the product xy. More specifically, each player P_i should end up with a ring element z_i such that

$$\sum_{i=1}^{n} z_i = xy.$$

The linear property of the additive secret-sharing scheme does not allow us to have a free (local) multiplication as it did with the additions.

Remark 9. *Suppose that the players have the shares of* $[x], [y]$ *and they would like to obtain shares of* $[xy]$*. If each player* P_i *locally multiplies their shares:* $z_i \leftarrow x_i y_i$*, then the players will obtain shares of* z *such that:*

$$z = \sum z_i = \sum x_i y_i \neq xy.$$

In order to build a multiplication protocol for secure multi-party computation we will assume that the players can have access to shares of three elements $[a], [b], [c]$ that are secret-shared amongst the players but are unknown to the players. In other words, each player P_i has access to a_i, b_i, c_i, but does not know the values of a, b or c. Furthermore, we consider that $(a, b) = \mathcal{G}_{\mathbb{Z}/2^l\mathbb{Z}}(2)$ and that $c = ab$.

The correlated random triple (a, b, c) is called Beaver's triple [Bea91].

For the moment we denote by $\mathcal{F}_{\text{triple}}(\mathbb{Z}/2^l\mathbb{Z})$ a function that privately generates a Beavers triple as defined above, and secret-shares it to the players using the additive scheme. This means each player P_i will get a_i, b_i, c_i such that

$$\sum_{i=1}^{n} a_i = a, \sum_{i=1}^{n} b_i = b, \sum_{i=1}^{n} c_i = c.$$

Algorithm 4 Multiplication protocol Π_{mul} dubbed as $[x], [y] \to [x] \cdot [y]$

Private input: P_i holds x_i,y_i for i in $\{1, ..., n\}$
Output: P_i holds z_i for i in $\{1, ..., n\}$

1: $(a_i, b_i, c_i) \leftarrow \mathcal{F}_{\text{triple}}(\mathbb{Z}/2^l\mathbb{Z})$
2: $d_i \leftarrow x_i - a_i$
3: $e_i \leftarrow y_i - b_i$
4: $d, e \leftarrow \Pi_{\text{open}}([d]), \Pi_{\text{open}}([e])$
5: $z_i \leftarrow c_i + a_i \cdot e + b_i \cdot d$
6: $z_1 \leftarrow z_1 + e \cdot d$ ▷ Only one player adds the public constant ed to their share of $[z]$.

The last line in the multiplication protocol as defined in algorithm 4 corresponds to adding a public constant to the secret, so only one player needs to add the constant, and here we chose P_1

without loss of generalization. All the lines of the algorithm could have been written using the protocol notations defined in this section. This means

$$d_i \leftarrow x_i - a_i, e_i \leftarrow y_i - b_i, \text{ and } z_i \leftarrow c_i + a_i \cdot e + b_i \cdot d + e \cdot d,$$

respectively correspond to

$$[d] \leftarrow \Pi_{\text{sub}}([x], [a]), [e] \leftarrow \Pi_{\text{sub}}([y], [b]), \text{ and}$$

$$[z] \leftarrow \Pi_{\text{cadd}}\left(\Pi_{\text{add}}\left(\Pi_{\text{add}}\left([c], \Pi_{\text{cmul}}\left([a], e\right)\right), \Pi_{\text{cmul}}\left([b], d\right)\right), ed\right),$$

where ed is a public constant.

Lemma 10. *Algorithm 4 outputs additive shares of xy.*

Proof. For the protocol to be correct, the sum of the shares z_i should be equal to the product xy.

$$\begin{aligned} z = \sum z_i &= ed + \sum c_i + a_i \cdot e + b_i \cdot d \\ &= ed + \sum c_i + e \sum a_i + d \sum b_i \\ &= ed + c + ea + db \\ &= (y-b)(x-a) + ab + (y-b)a + (x-a)b \\ &= yx - ya - bx + ba + ab + ya - ba + xb - ab \\ &= xy. \end{aligned}$$

□

Definition 11. *For secrets $[x], [y]$ and public element α all in $\mathbb{Z}_{2^l}$, the following three notations are equivalent:*

$$\begin{array}{lll} [x] + [y] & = \Pi_{\text{add}}([x], [y]) & = [x + y] \\ [x] - [y] & = \Pi_{\text{sub}}([x], [y]) & = [x - y] \\ [x] \cdot [y] & = \Pi_{\text{mul}}([x], [y]) & = [xy] \\ [x] + \alpha & = \Pi_{\text{cadd}}([x], \alpha) & = [x + \alpha] \\ \alpha \cdot [x] & = \Pi_{\text{cmul}}([x], \alpha) & = [\alpha x] \end{array}$$

Algorithm 5 Example of the circuit evaluation protocol from figure 2.1

Private input: P_1, P_2, P_3 own respectively x^1, x^2, x^3;
Output:

1: P_i additively secret-shares $\left[x^i\right]$: $\left[x^i\right] \leftarrow \Pi_{\text{share}}(x^i)$
2: Addition of the secrets: $\left[x^2 + x^3\right] \leftarrow \Pi_{\text{add}}(\left[x^2\right], \left[x^3\right])$
3: Multiplication of secrets: $\left[x^1 x^2\right] \leftarrow \Pi_{\text{mul}}(\left[x^1\right], \left[x^2\right])$
4: Addition of secrets: $\left[x^1 x^2 + x^2 + x^3\right] \leftarrow \Pi_{\text{add}}(\left[x^1 x^2\right], \left[x^2 + x^3\right])$
5: The players reconstruct the final value together: $x^1 x^2 + x^2 + x^3 \leftarrow \Pi_{\text{open}}(\left[x^1 x^2 + x^2 + x^3\right])$

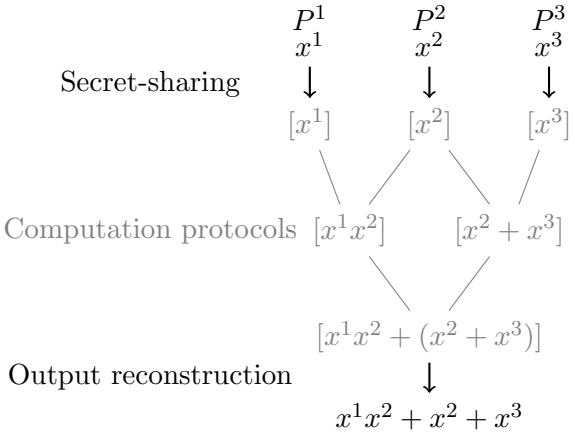

Figure 2.1: Example of a circuit evaluation using additive secret-sharing where each player P_i holds a secret x^i and their goal is to learn the function $f(x^1, x^2, x^3) = x^1x^2 + x^2 + x^3$. The secure multi-party computation protocol used to evaluate this circuit is presented in algorithm 5.

2.2.3 Shamir's Secret Sharing Scheme

Shamir's secret sharing scheme is another secret sharing scheme that can only be applied in a finite field, it was introduced before secure multi-party computation emerged. It consists of sharing a secret s in a finite field with n players with threshold $t \in \mathbb{N}; t < n$ by choosing a polynomial P of degree t, such that

$$P(0) = s \text{ and } \forall i \in \{1, ..., n\}, s_i = P(i).$$

This scheme is t-private and requires $t + 1$ players to reconstruct the secret s. The secret can be reconstructed using Lagrange's polynomial interpolation.

Shamir's scheme is linear like the additive secret sharing scheme, so it allows the players to compute addition gates, and multiplication-by-a-constant gates locally. One way to see it is that the sum of two polynomials P, P' of degree t, such that $P(0) = s$ and $P'(0) = s'$ is also a polynomial P'' of degree t with $P''(0) = s + s'$. The addition by a constant is also free, but it changes slightly; instead of having one player add the constant to their share, all the players add the constant to their share.

The reader can find a detailed explanation of the multiplication protocol, and more information about Shamir's scheme in [CDN15].

2.3 Machine Learning and Data Mining

Note that in this section there are no secure multi-party computation protocols, the functions, and variables defined here are not related to the players' secrets and the function they wish to evaluate. This means that the algorithms defined in this section are locally executed.

Machine learning [Mit97] is a subset of artificial intelligence that focuses on giving computers the ability to learn from data and to identify patterns from large datasets without being explicitly programmed. It uses algorithms that are able to detect patterns in data and use them to make predictions or decisions. In our work, we consider examples of supervised and unsupervised learning. Supervised learning is a type of machine learning algorithm that uses a known dataset (labeled data) to predict outcomes. It requires human intervention to properly label the data in order

to train the model and adjust the model's parameters to achieve desired outcomes. Supervised learning algorithms are used in a variety of applications, such as image recognition, natural language processing, and fraud detection. On the other hand, unsupervised learning is a type of machine learning algorithm used to find patterns in data without any prior knowledge or labels. This type of learning is used mainly for clustering and dimensionality reduction. It does not require human intervention and can be used to discover relationships between data points which can lead to more meaningful insights.

Data mining [Han07] is the process of finding patterns and correlations within large datasets to predict outcomes gain insight and make informed decisions.

2.3.1 Optimization of Multi-dimensional Functions

Let $n \in \mathbb{N}^*$, we consider a function f as following:

$$\begin{array}{rccc} f: & \mathbb{R}^n & \to & \mathbb{R} \\ & x = (x^1, \ldots, x^n) & \to & y = f(x^1, \ldots, x^n). \end{array} \tag{2.2}$$

The goal is to determine the points where f reaches its optimal values. We define the gradient of f at the point x, denoted by $\nabla f(x)$, as the following:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x^1}, \ldots, \frac{\partial f}{\partial x^n} \right). \tag{2.3}$$

In order to convert to a point where f reaches a local minimum, a sequence $\{x^{(0)}, x^{(1)}, x^{(2)}, \ldots\}$ where each point $x^{(i)} \in \mathbb{R}^n$ that converges to a point $x^* \in \mathbb{R}^n$ verifying $\nabla f(x^*) = 0$ amongst other conditions.

A descent method is an iterative algorithm in which a sequence of the following form is defined:

$$x^{(k+1)} = x^{(k)} + s^k d^k; k \in \mathbb{N}, s^k \in \mathbb{R}_+, d^k \in \mathbb{R}^n \text{ and } (d^k) \cdot \nabla f(x^{(k)} < 0. \tag{2.4}$$

The choice of s^k and d^k has to ensure that $f(x^{(k+1)}) < f(x^{(k)})$.

2.3.2 Examples of Machine Learning and Data mining Algorithms

In this section, we present examples machine learning and data mining algorithms. These algorithms will be used to apply secure multi-party computation protocols. We describe the linear regression model, which is the most basic of models, the logistic regression model which will be used a lot in the thesis, restricted Boltzmann machines, and data mining algorithms called truth-finding algorithms.

2.3.2.1 Linear Regression

A linear regression model aims at describing a somewhat linear relationship between a real dependent variable and other real independent variables. Given a data set D of $r \in \mathbb{N}$ rows and $d+1 \in \mathbb{N}^*$ columns:

$$D = \{x^{i,1} = 1, x^{i,2}, \ldots, x^{i,d}, y^i\}_{i=1\ldots r}, \tag{2.5}$$

a linear regression model assumes that the relationship is close to linear between $y \in \mathbb{R}^r$ (representing the y^i's) and $x \in \mathbb{R}^{n \times d}$ (representing the x^i's) where each row of x is denoted as x^i. In other words, there exist real coefficients $W^* = (W^{1*}, \ldots, W^{d*})$ called weights that best fit the data set D using a least-squares approach. Consequently, these weights should be able to correctly predict y^i for a given data point x^i.

Mathematically, the weights W^* we are searching for are defined as the following:

$$W^* = \text{argmin}_{W \in \mathbb{R}^d} \sum_{i=1}^{r} \left((W^1 x^{i,1} + \cdots + W^d x^{i,d}) - y^i \right)^2. \tag{2.6}$$

The weights W^* are computed in a way that the error between the prediction and y^i is at its minimum.

If we define the error function:

$$\mathcal{E}(W) = \sum_{i=1}^{r} ((W^1 x^{i,1} + \cdots + W^d x^{i,d}) - y^i)^2 = \sum_{i=1}^{r} (\hat{y^i} - y^i)^2, \tag{2.7}$$

where,

$$\hat{y^i} = W^1 x^{i,1} + \cdots + W^d x^{i,d}, \tag{2.8}$$

is the prediction, then we search for the weight W^* that minimize the error function $\mathcal{E}$. The minimum is computed iteratively via a gradient descent algorithm as seen in section 2.3.1, by choosing the descent direction d^k to be opposite to the gradient. The quantity $\mathcal{E}(W)$ can be written as

$$\mathcal{E}(W) = \sum_{i=1}^{r} (W \cdot x^i - y^i)^2 = \sum_{i=1}^{r} (W \cdot x^i)^2 - 2y^i (W \cdot x^i) + (y^i)^2, \tag{2.9}$$

which makes the gradient computed as the following:

$$\begin{aligned} \nabla \mathcal{E}(W) &= \sum_{i=1}^{r} 2x^i (W \cdot x^i) - 2y^i x^i \\ &= \sum_{i=1}^{r} 2x^i (W \cdot x^i - y^i) \\ &= \sum_{i=1}^{r} 2x^i (\hat{y^i} - y^i). \end{aligned} \tag{2.10}$$

Finally, let $\alpha \in \mathbb{R}_+$ be a parameter for the gradient descent, in order to compute the minimum of $\mathcal{E}$ the following sequence can be used:

$$\begin{cases} W^{(0)} & = \text{ random weights} \\ W^{(k+1)} & = W^{(k)} - \alpha \nabla \mathcal{E}(W^{(k)}). \end{cases} \tag{2.11}$$

This simplified description of linear regression is enough to understand the thesis. For more information about linear regression analysis, the reader is referred to [MPV01].

2.3.2.2 Logistic Regression

A logistic regression [Ber44] model describes a relationship between a binary dependent variable and other real independent variables. Given a data set D of $r \in \mathbb{N}$ rows and $d+1 \in \mathbb{N}^*$ columns:

$$D = \{x^{i,1} = 1, x^{i,2}, \ldots, x^{i,d}, y^i\}_{i=1 \ldots r}, \tag{2.12}$$

a logistic regression model assumes that there is a correlation between the binary variable $y \in \{0,1\}^r$ (called label) and the vector $x \in \mathbb{R}^{n \times d}$ (called features, representing the x^i's) where each x^i is a row

of x. In other words, there exist real coefficients $W^* = (W^{1*}, \dots, W^{d*})$ called weights. Precisely, the model is that the y^i's should be close to:

$$\hat{y^i} = \text{sigmoid}((x^i) \cdot W^*) = \text{sigmoid}(x^{i,1}W^1 + \cdots + x^{i,d}W^d), \tag{2.13}$$

where sigmoid function is an activation function, which takes a real and outputs a value between zero and one:

$$\begin{array}{rccc} \text{sigmoid}: & \mathbb{R} & \to & \mathbb{R} \\ & x & \to & \text{sigmoid}(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{1+\exp(-x)}. \end{array} \tag{2.14}$$

We aim at finding the weights W^* which are the maximum likelihood estimate for the y^i's, with respect to the model given in equation 2.13. In other words, we aim at minimizing the following objective function:

$$f_D(W) = -\sum_{i=1}^{r} \left(y^i \log(\hat{y^i}) + (1 - y^i) \log(1 - \hat{y^i}) \right), \tag{2.15}$$

and it can also be written as:

$$\begin{aligned} f_D(W) &= -\sum_{i=1}^{r} \left(y^i \log(\hat{y^i}) + (1 - y^i) \log(1 - \hat{y^i}) \right) \\ &= -\sum_{i=1}^{r} \left(y^i \log(\text{sigmoid}((x^i)W)) + (1 - y^i) \log(1 - \text{sigmoid}((x^i)W)) \right) \\ &= -\sum_{i=1}^{r} \left(y^i \log\left(\frac{\exp((x^i)W)}{1 + \exp((x^i)W)} \right) + (1 - y^i) \log\left(\frac{1 + \exp((x^i)W) - \exp((x^i)W)}{1 + \exp((x^i)W)} \right) \right) \\ &= -\sum_{i=1}^{r} \left(y^i \left((x^i)W - \log(1 + \exp((x^i)W)) \right) + (1 - y^i) \left(-\log(1 + \exp((x^i)W)) \right) \right) \\ &= -\sum_{i=1}^{r} \left(y^i (x^i)W - \log(1 + \exp((x^i)W)) \right) \\ &= \sum_{i=1}^{r} \left(\log\left(1 + \exp((x^i) \cdot W)\right) - y^i (x^i) \cdot W \right). \end{aligned} \tag{2.16}$$

Thus our goal is to find the weight W^* where the function f_D reaches its minimum:

$$\begin{aligned} W^* &= \text{argmin}_{W \in \mathbb{R}^d} f_D(W) \\ &= \text{argmin}_{W \in \mathbb{R}^d} \sum_{i=1}^{r} \left(\log\left(1 + \exp((x^i) \cdot W)\right) - y^i (x^i) \cdot W \right). \end{aligned} \tag{2.17}$$

The obtained minimum W^* of f_D is the model that will be used to classify the data points of D. The minimum is computed iteratively via a gradient descent algorithm which uses the gradient of f_D denoted as ∇f_D. The gradient ∇f_D can be computed by multiplying the gradient of f_D with respect to y^i by the gradient of y^i with respect to W. It is classical that from this computation we obtain the following expression:

$$\nabla f_D(W) = \sum_{i=1}^{r} x^i (\hat{y^i} - y^i). \tag{2.18}$$

Finally, let $\alpha_k \in \mathbb{R}_+$, in order to compute the minimum of $\mathcal{E}$ the following sequence can be used:

$$\begin{cases} W^{(0)} & = \text{ random weights} \\ W^{(k+1)} & = W^{(k)} - \alpha \nabla f_D(W^{(k)}). \end{cases} \tag{2.19}$$

Note that $\nabla f_D(W)$ is denoted by ∇W, and that the positive constant $\alpha \in \mathbb{R}_+$ is called learning rate. The positive integer called *epochs* represents the number of times a dataset passes through the algorithm.

In this thesis, we provide a simplified description of logistic regression, which is sufficient to understand the key concepts. For more detailed information about logistic regression analysis, the reader can can check [KK02]. We illustrate the baseline algorithm that we use for training a logistic regression in algorithm 6

Algorithm 6 Baseline algorithm for training a logistic regression

Input: D the dataset
Public parameters: α the learning rate,
epochs the number of epochs,
r the number of rows in the training dataset
Output: The players hold the optimal weights W^*

Normalize the dataset
Split the dataset horizontally into training and testing data
$x \leftarrow$ feature columns of the training data from D
$y \leftarrow$ the label column of the training data from D
Initialze weights W with a normal distribution,
for every *epoch* **do**
    $\hat{y} \leftarrow \text{sigmoid}(x \cdot W)$
    $\nabla W \leftarrow (\hat{y} - y) \cdot x$
    $W \leftarrow W - \alpha \cdot \frac{\nabla W}{r}$
end for
$W^* \leftarrow W$

2.3.2.3 Restricted Boltzmann Machines

A restricted Boltzmann machine [FI12; Hin12] is a type of "generative neural network" that can learn the probability distribution of a given set of data. Restricted Boltzmann machines can be used to generate new data that has a similar probability distribution to the dataset on which it was trained, this is why it is called "generative". They are used extensively in unsupervised learning and can be applied to a variety of problems such as image recognition, natural language processing, and recommendation systems.

Restricted Boltzmann machines are trained using a process called contrastive divergence, which is a type of algorithm that uses an iterative process to adjust the weights of the machine to minimize the difference between the output that the machine predicts and label of the data on which it is trained. The process begins by randomly initializing the weights of the machine and then adjusting the weights in each iteration based on the difference between the expected output and the actual output. The weights are then adjusted according to a learning rate, and the process is repeated until the weights are optimal. More precisely, they consist of visible and hidden neurons, arranged in bipartite layers. The visible units correspond to the components of an observation like an image,

and the hidden units correspond to the dependencies between the components of the observation, like the pixels of the image.

The training of a restricted Boltzmann machine is based on the optimization of a cost function, which measures the difference between the actual output and the predicted output. The two main equations used to train a restricted Boltzmann machine are the weight update rule and the energy function. The weight update rule states that the weights that link the visible and hidden units are adjusted based on the difference between the actual output and the predicted output. The energy function is an equation that determines the probability of a particular state of the visible and hidden units.

Set $v \in \{0,1\}^r$ and $h \in \{0,1\}^d$ the states of the visible units and hidden units respectively at iteration 0, and $W^{ij} \in \mathbb{R}^{r \times d}$ to be the weight between the i-th visible unit and j-th hidden unit, α the learning rate . Set $a \in \mathbb{R}^r$ to be the bias – which is also a vector of weights – of the visible units, and $b \in \mathbb{R}^d$ the bias of the hidden units. The energy function is expressed as:

$$E(v,h) = -\sum_{i=1}^{r}\sum_{j=1}^{d} W^{ij} v^i h^j - \sum_{i=1} a^i v^i - \sum_{j=1}^{d} b^j h^j. \tag{2.20}$$

Set $v' \in [0,1]^r$ and $h' \in [0,1]^d$ the states of the visible and hidden units respectively after an iteration. The energy function defines how v' and h' are computed, which is as the following:

$$v' = \text{sigmoid}(a + W \cdot h) \text{ and } h' = \text{sigmoid}(b + W^T \cdot v). \tag{2.21}$$

The weight update rule is expressed for a single iteration as:

$$W^{ij} \leftarrow W^{ij} + \alpha(v^i h^j - v'^i h'^j). \tag{2.22}$$

2.3.2.4 Truth-finding Algorithms

Truth finding [YHY08; Gal+10] is an effective data mining algorithm used to handle uncertain data. More specifically, when some information in a dataset is missing and the dataset owner, also called client, does not have access to this information, they can ask sources questions – also called query or fact – in order to complete the dataset. Yet again, the client may not be completely sure of the answer that the sources are delivering as their answers could be contradictory. Truth-finding algorithms rely on the correlation between the answers of all the sources. Furthermore, the client does not have any information about how the sources get their information, i.e., how they construct their model, and how they take their decisions.

Consider set of queries $\{f^1, \ldots, f^d\}$ and a set of r sources. Each source maps the query f^i to $\{-1, 0, 1\}$, and the image of the query computed by a source represents the source's view of the query, it is the source's answer to the query. A negative answer represents a false fact, a positive answer represents a true fact, and a null answer is an undetermined fact. We set $v^1, \ldots, v^r$ to be the sources' views of the facts, more precisely, $v^i \in \{-1, 0, 1\}^d$ is the ith source's view of the queries $\{f^1, \ldots, f^d\}$.

A trivial truth-finding algorithm is majority voting. In this thesis, we consider two other existing truth-finding algorithms, Cosine, and 3-Estimates from [Gal+10].

The idea of Cosine, as detailed in algorithm 7 from [Gal+10, Algorithm 1], is to successively calculate a "truth value" for every fact taking into account the perspectives of all sources, that is the truth value of a fact is the probability that the fact is correct. During each iteration, the algorithm also adjusts a "trust factor" for each source, that is an element that shows how much a source is trustworthy. Ultimately, the algorithm yields one truth value for each fact and one trust

factor for each source. The truth value and trust factor are initialized and then modified at each iteration in the following manner: the truth value is determined as the sum of answer weighted with the trust factor of each source, and then the trust factor is calculated by normalizing the number of answers that each source got right.

Algorithm 7 Cosine algorithm [Gal+10]

Input: The answers $(v^{ij})_{i=1\dots r}^{j=1\dots d}$ from the sources, where $v \in \{-1, 0, 1\}^{r\times d}$,
$\eta \in (0, 1]$ is the learning rate
Public parameter: N of iterations k
Outputs: Client computes $y \in [-1, 1]^d$, a truth value for each of the d queries
and $\theta \in [-1, 1]^r$, a trust factor for each source

for $i = 1 \dots r$ **do** ▷ Initialization of the trustworthiness of each source
  $\theta^i \leftarrow \frac{(\sum_{j:v^{ij}=1} v^{ij})+(\sum_{j:v^{ij}=-1} v^{ij})}{\sum_{j:v^{ij}\neq 0} v^{ij}}$
end for
for $j = 1 \dots d$ **do** ▷ Initialization of the truth value of each query
  $y^j \leftarrow 1$
end for
for k iterations **do**
  for $i = 1 \dots r$ **do** ▷ Update the trustworthiness of each source
    $posFacts \leftarrow \sum_{j:v^{ij}=1} y^j$
    $negFacts \leftarrow \sum_{j:v^{ij}=-1} y^j$
    $norm \leftarrow \sqrt{(\sum_{j:v^{ij}\neq 0} v^{ij})(\sum_{j:v^{ij}\neq 0}(y^j)^2)}$
    $\theta^i \leftarrow (1-\eta)\cdot\theta^i + \eta\cdot\frac{posFacts-negFacts}{norm}$
  end for
  for $j = 1 \dots d$ **do** ▷ Update the truth value of each query
    $posViews \leftarrow \sum_{i:v^{ij}=1}(\theta^i)^3$
    $negViews \leftarrow \sum_{i:v^{ij}=-1}(\theta^i)^3$
    $norm \leftarrow \sum_{i:v^{ij}\neq 0}(\theta^i)^3$
    $y^j \leftarrow \frac{posViews-negViews}{norm}$
  end for
end for
$y \leftarrow (y^j)_{j=1\dots d}$
$\theta \leftarrow (\theta^i)_{i=1\dots r}$

The algorithm 3-Estimates described in algorithm 8 from [Gal+10, Algorithm 3] takes a third variable into account: the complexity of the query. The algorithm yields a truth value and trust factor comparable to Cosine, but additionally outputs a "difficulty factor" for each fact, this factor is computed based on the number of incorrect views of the fact. More formally, for a query f^j if we set δ^j to be the likelihood of the query f^j being difficult and θ^i the probability of the ith source (who didn't answer 0) being untrustworthy, then the equation that describes how the algorithm assesses the truth value is:

$$\begin{cases} \Pr(\text{source } i \text{ is wrong on fact } j) \coloneqq \delta^j\theta^i \\ \Pr(\text{source } i \text{ is right on fact } j) \coloneqq 1-\delta^j\theta^i. \end{cases} \tag{2.23}$$

If v^{ij} the answer of the ith source to f^j, and λ^j is the number of sources that gave a non-null

answer query f^j:

$$\lambda^j = \sum_{i=1}^{r} |v^{ij}|, \tag{2.24}$$

then the probability of f^j to be true is given by:

$$\begin{aligned}\lambda^j \Pr(f^j \text{ is true}) &= \textstyle\sum_{i,v^{ij}=1} \Pr(i \text{ is right on } j) + \sum_{i,v^{ij}=-1} \Pr(i \text{ is wrong on } j) \\ &= \textstyle\sum_{i,v^{ij}=1} 1 - \delta^j\theta^i + \sum_{i,v^{ij}=-1} \delta^j\theta^i.\end{aligned}$$

The algorithm 3-Estimates described in algorithm 8 uses a function called "normalize", for a vector $x \in \mathbb{R}^n$ it is defined as the following:

$$x^i \leftarrow \frac{x^i - \min_{j=1\ldots n} x^j}{\max_{j=1\ldots n} x^j - \min_{j=1\ldots n} x^j}, \text{ for } i \in \{1, \ldots, n\}. \tag{2.25}$$

We also initialize all the trustworthiness of the players to be 0.4 and the difficulty of the query to be 0.1 as in the original paper[Gal+10].

2.4 Differential Privacy and Functional Mechanism

Note that in this section as well there are no secure multi-party computation protocols, the algorithms are executed in clear without any communications.

2.4.1 Differential Privacy

Differential privacy [Dwo+06] is a technique used to protect individual privacy during the collection and processing of data. This technique works by introducing a certain amount of randomness or "noise" into the dataset, which makes it challenging to identify a single data point in a larger group of data points. This randomness helps to mask and obscure the exact data of any individual, while still allowing the data to be used for statistical analysis in a meaningful way.

Differential privacy [Dwo+06] is typically divided into two notions, the first one is ϵ-differential privacy and ϵ, δ-differential privacy, the second one is ϵ-differential privacy. They both guarantee that an individual's data is not revealed. The notion of ϵ, δ-differential privacy is different from the latter as it contains an additional parameter which yields a weaker privacy guarantee.

In order to define ϵ-differential privacy, we consider the same definition of neighboring datasets as in [DR14], i.e. two datasets that differ in only one row.

Definition 12 (ϵ-Differential Privacy)**.** *Let $\epsilon \in \mathbb{R}_+$, a randomized algorithm $\mathcal{A}$ provides ϵ-differential privacy if for any two neighboring datasets D^1 and D^2 that differ in only one row, and for any subset of outputs S, the following holds:*

$$\Pr[\mathcal{A}(D^1) \in S] \leq e^\epsilon \cdot \Pr[\mathcal{A}(D^2) \in S]. \tag{2.26}$$

Definition 13 (ϵ, δ-Differential Privacy)**.** *Let $\epsilon, \delta \in \mathbb{R}_+$, a randomized algorithm $\mathcal{A}$ provides ϵ, δ-differential privacy if for any two neighboring datasets D^1 and D^2 that differ in only one row, and for any subset of outputs S, the following holds:*

$$\Pr[\mathcal{A}(D^1) \in S] \leq e^\epsilon \cdot \Pr[\mathcal{A}(D^2) \in S] + \delta. \tag{2.27}$$

Algorithm 8 3-Estimates algorithm [Gal+10]

Input: The answers $(v^{ij})_{i=1\dots r}^{j=1\dots d}$ from the sources
Public parameter: N of iterations k
Outputs: Client computes $y \in [0,1]^d$, a truth value for each of the d queries,
$\theta \in [0,1]^r$, a trust factor for each source
and $\delta \in [0,1]^d$, a difficulty factor for each of the d queries

for $i = 1 \dots r$ **do** ▷ Initialization of the untrustworthiness of each source
  $\theta^i \leftarrow 0.4$
end for
for $j = 1 \dots d$ **do** ▷ Initialization of the difficulty of each query
  $\delta^j \leftarrow 0.1$
end for
for k iterations **do**
  for $j = 1 \dots d$ **do** ▷ Update the truth value of each query
    $posViews \leftarrow \sum_{i:v^{ij}=1} 1 - \theta^i \delta^j$
    $negViews \leftarrow \sum_{i:v^{ij}=-1} 1 - \theta^i \delta^j$
    $nbViews \leftarrow \sum_{i:v^{ij}\neq 0} v^{ij}$
    $y^j \leftarrow \frac{posViews + negViews}{nbViews}$
  end for
  Normalize y
  for $j = 1 \dots d$ **do** ▷ Update the difficulty score of each query
    $posViews \leftarrow \sum_{i:v^{ij}=1} \frac{1-y^j}{\theta^i}$
    $negViews \leftarrow \sum_{i:v^{ij}=-1} \frac{y^j}{\theta^i}$
    $nbViews \leftarrow \sum_{i:v^{ij}\neq 0} v^{ij}$
    $\delta^j \leftarrow \frac{posViews + negViews}{nbViews}$
  end for
  Normalize δ
  for $i = 1 \dots r$ **do** ▷ Update the untrustworthiness of each source
    $posViews \leftarrow \sum_{j:v^{ij}=1} \frac{1-y^j}{\delta^j}$
    $negViews \leftarrow \sum_{j:v^{ij}=-1} \frac{1-y^j}{\delta^j}$
    $nbViews \leftarrow \sum_{j:v^{ij}\neq 0} v^{ij}$
    $\theta^i \leftarrow \frac{posViews + negViews}{nbViews}$
  end for
  Normalize θ
end for
$y \leftarrow (y^j)_{j=1\dots d}$
$\delta \leftarrow (\delta^j)_{j=1\dots d}$
$\theta \leftarrow (\theta^i)_{i=1\dots r}$

The ϵ-differential privacy can be achieved in machine learning by a variety of methods, the most common being output perturbation, and objective perturbation. Output perturbation adds noise to the output of a machine-learning model, thus obfuscating individual data points. Objective perturbation adds noise to the objective function of a machine learning model, making it difficult to identify individual data points.

2.4.2 Functional Mechanism on a Logistic Regression

We recall that ϵ is a privacy parameter where a smaller value of ϵ guarantees more privacy. From equation 2.26, if an algorithm A satisfies ϵ-differential privacy, then the probability distribution of A's output S is roughly the same for any two databases that differ in one tuple.

In [Zha+12], the authors present functional mechanism, a method – described in [Zha+12, algorithm 1] – that adds noise to any objective function in order to achieve an ϵ-differentially private optimization of that function.

We will use functional mechanism to train an *epsilon* differentially private logistic regression, by applying [Zha+12]'s method to the objection function f_D defined equation 2.16.

As per [Zha+12], first step is to approximate the objective function f_D with a second-order Taylor polynomial approximation, the following equation is not explicitly given in [Zha+12]:

$$\hat{f}_D(W) = \sum_{i=1}^{r} \sum_{k=0}^{2} \frac{f_1^{(k)}(0)}{k!} \left((x^i) \cdot W\right)^k - \left(\sum_{i=1}^{r} y^i (x^i) \right) \cdot W, \tag{2.28}$$

where $f_1(z) = \log(1 + \exp(z))$. Expanding the above equation gives:

$$\hat{f}_D(W) = \sum_{i=1}^{r} \log(2) + \sum_{i=1}^{r} \frac{(x^i) \cdot W}{2} + \sum_{i=1}^{r} \frac{((x^i) \cdot W)^2}{8} - \sum_{i=1}^{r} \left(y^i (x^i)\right) \cdot W. \tag{2.29}$$

Functional mechanism achieves privacy by optimizing a function $\tilde{f}_D$ which is defined by perturbing $\hat{f}_D$ by adding to its coefficients Laplacian noise η^0, η^1, η^2 sampled with the distribution denoted by $\mathcal{L}$. In [Zha+12, section 5.3], the authors explain how to compute the parameters a logistic regression, they are defined as follows:

$$\begin{aligned} \epsilon &\in (0, 1], \\ \Delta &= 3d + \frac{d^2}{4}, \\ \beta &= \frac{\Delta}{\epsilon}, \\ \eta^0, \eta^1, \eta^2 &\sim \mathcal{L}(0, \beta). \end{aligned} \tag{2.30}$$

Consequently, the perturbed function $\tilde{f}_D$ is not explicitly given in [Zha+12] and it is defined as follows:

$$\tilde{f}_D(W) = \sum_{i=1}^{r} (\log(2) + \eta^0) + \left(\frac{(x^i)}{2} - (x^i) y^i + \eta^1 \right) W + W^T \left(\frac{(x^i)^T \cdot (x^i)}{8} + \eta^2 \right) W. \tag{2.31}$$

The derivative of the perturbed objective function of logistic regression $\tilde{f}_D$ with respect to the weights, denoted as ∇W, will be used as a perturbed gradient to update the weights during the training process of the algorithm of the logistic regression. It is computed as follows:

$$\nabla W = \nabla \tilde{f}_D(W) = \sum_{i=1}^{r} \left((x^i) \left(1/2 - (y^i) \right) + \eta^1 \right) + 2 \left(\frac{(x^i)^T (x^i)}{8} + \eta^2 \right) W. \tag{2.32}$$

The algorithm for training a logistic regression whose output model satisfies ϵ-differential privacy with the described functional mechanism is given is algorithm 9. In simple words, in algorithm 9, instead of optimizing the objective function f_D, the noisy function $\tilde{f}_D$ is optimized through gradient descent.

Algorithm 9 Baseline algorithm for training a logistic regression with the functional mechanism

Input: D the dataset

Public parameters: ϵ the privacy parameter,
$\Delta = 3d + 0.25d^2$ from equation 2.30, where $d + 1$ is the number of columns in D,
α the learning rate,
epochs the number of epochs,
r the number of rows in the training dataset

Output: W^*

1: Normalize the dataset
2: Split data into training and testing data
3: $x \leftarrow$ feature columns of the training data from D
4: $y \leftarrow$ the label column of the training data from D
5: Initialize W with a normal distribution $\beta \leftarrow \frac{\Delta}{\epsilon} \cdot epochs$
6: **for** every *epoch* **do**
7: $\quad \eta^1 \leftarrow \mathcal{L}(0, \beta)$
8: $\quad \eta^2 \leftarrow \mathcal{L}(0, \beta)$
9: $\quad \nabla W \leftarrow \left((1/2 - y)^T x + \eta^1 \right) + 2 \left(\frac{x^T x}{8} + \eta^2 \right) W$
10: $\quad W \leftarrow W - \alpha \cdot \frac{\nabla W}{r}$
11: **end for**
12: $W^* \leftarrow W$

Theorem 14. *Algorithm 9 satisfies ϵ-differential privacy.*

Proof. The function $\tilde{f}_D$ is an objective function in a polynomial form with Laplacian noise of a well-defined scale $\frac{\Delta}{\epsilon}$ added to its coefficients. Then by By [Zha+12, theorem 1] the algorithm that computes the argument of its minimum, W^*, is ϵ-differentially private.

Moreover, for the objective function of a logistic regression f_D, the parameter $\Delta = 0.25d^2 + 3d$, as per [Zha+12, section 5.3] □

Chapter 3

Methods for Numerical Computations on Shared Secrets

In this chapter, we explore the main building block of secure multi-party computation protocols for machine learning, which is fixed-point precision. We then expose the common protocols and algorithms used in machine learning tasks, as well as the fundamentals of differential privacy, homomorphic encryption, and garbled circuits in a machine learning setting.

3.1 Fixed-Point Precision for Secure Multi-party Computation

As explained in section 2.2, multi-party computation protocols are applied to secrets defined in a finite ring or field. However, machine learning tasks are applied to real elements, so some transformations are required before applying multi-party computation protocols to private datasets.

3.1.1 Mapping Real Elements to a Finite Ring

Secure multi-party computation requires having the secrets in a finite field or ring, in our case, we will use a finite ring $\mathbb{Z}/2^l\mathbb{Z}$ with $l \in \mathbb{N}^*$. Therefore, we start the secure computation of machine learning models on private data by defining a map between the real elements and a finite ring.

We consider that the real elements are in a bounded interval $[-2^{l-p-1}, 2^{l-p-1})$ of $\mathbb{R}$ which we approximate so they can belong to the real subset

$$D_{p,l} = \{k \cdot 2^{-p}; k \in [-2^{l-1}, 2^{l-1})\}. \tag{3.1}$$

Here p is the bit fixed precision. We represent the elements of $\mathbb{Z}_{2^l} = \mathbb{Z}/2^l\mathbb{Z}$ by the integers in $[0, 2^l) \cap \mathbb{N}$.

In this section, we explore the foundation of secure multi-party computation protocols for machine learning. We expose the common protocols and algorithms used in machine learning tasks, as well as the fundamentals of differential privacy, homomorphic encryption, and garbled circuits in a machine learning setting.

$$\begin{array}{rccc} \phi: & D_{p,l} & \xrightarrow{\phi} & \mathbb{Z}_{2^l} \\ & x & \to & \bar{x} = \phi(x) = 2^p x \mod 2^l, \end{array} \tag{3.2}$$

where $\mod 2^l$ means that we choose the representative of the class in $[0, 2^l) \cap \mathbb{N}$.

This map allows the players to have secrets in the finite ring $\mathbb{Z}_{2^l}$, and perform secret-sharing schemes over their secrets. This map's domain is a given set of decimals, so, to secret-share real

numbers, the players need to approximate their input to an element that is part of the set $D_{p,l}$. This approximation induces errors which we will study and analyze in chapter 6 and more specifically in section 6.1.

3.1.2 Truncation Protocols

Suppose players need to evaluate a multiplication gate on real secrets that have been mapped and secret-shared in the finite ring. In other words, each player holds a share of $[\bar{x}]$ and $[\bar{y}]$, where

$$\bar{x} = \phi(x) = 2^p x \mod 2^l \text{ and } \bar{y} = \phi(y) = 2^p y \mod 2^l, \tag{3.3}$$

using the map ϕ defined in section 3.1.1. The players would like to obtain shares of $[\bar{z}]$, where

$$\bar{z} = \phi(z) = 2^p(xy) \mod 2^l. \tag{3.4}$$

If they use the multiplication protocol as it is defined in algorithm 4 then the players would obtain the following output:

$$\begin{aligned} \bar{x}\bar{y} = \phi(x)\phi(y) &= (2^p x \mod 2^l)(2^p y \mod 2^l) \\ &= (2^p x 2^p y) \mod 2^l \\ &= 2^{2p}(xy) \mod 2^l \\ &\neq 2^p(xy) \mod 2^l. \end{aligned} \tag{3.5}$$

In order to obtain shares of $[\bar{z}]$ the players need to "divide" the product $\bar{x}\bar{y}$ by 2^p; this is called a truncation of p bits.

In simple words, we need to apply a truncation protocol after each multiplication protocol. A two-party truncation protocol that does not use communications was described in [MZ17], it gives the correct output with a certain probability. Another truncation protocol that can be applied with any number of players is found in [CS10]. There are also other truncation protocols for multi-party computation; examples of such protocols are in [CS10] and [Esc+20]. Some of these protocols always return the correct output, others return the correct output with a certain probability.

In chapter 6 and more specifically in section 6.1 we introduce definitions, and lemmas to formalize the problem and the solution related to approximation and truncations.

3.2 Secure Multi-party Computation Protocols in Machine Learning

Since secure multi-party computation is used to protect sensitive data by allowing multiple parties to securely compute on it without exposing it, one can see that it could be especially useful when dealing with machine learning tasks. Indeed, it could be used for training models while protecting data that contain private or sensitive information, such as financial records, health records, or user profiles. Secure multi-party computation could be used in more than one way, the first one being for federated learning tasks, such as training models on large datasets that are distributed across multiple parties. Another way is through secure inference which preserves the privacy of the model.

In this section, we present the secure multi-party computation protocols and algorithms that are commonly used for machine learning tasks, like some activation functions from [MZ17], a comparison protocol from [Mak+21] as well as other algorithms and tools regularly employed for machine learning.

3.2.1 Numerical Functions

The exponentiation algorithm is detailed in algorithm 10 and it is from [Ryf19]. For $x \in \mathbb{R}$ and $n = 2^k \in \mathbb{N}^*$, the approximation y of $\exp(x)$ is computed as follows:

$$y = \left(1 + \frac{x}{n}\right)^n. \tag{3.6}$$

This exponentiation algorithm is implemented in the privacy-preserving machine learning framework PySyft [Ryf+18].

Algorithm 10 Exponentiation approximation algorithm Π_{exp}

Private input: Shares of the secret $[x]$
Public parameter: Number of iterations k
Output: Shares $[y]$ with y as in equation 3.6
1: $[y] \leftarrow 1 + [x] \cdot 2^{-k}$
2: **for** $i = 1..k$ **do**
3: $\quad [y] \leftarrow [y] \cdot [y]$
4: **end for**

Remark 15. *In algorithm 10, the operation $[x] \cdot 2^{-k}$ means that the real constant 2^{-k} is mapped to $\mathbb{Z}_{2^l}$ using the map ϕ from equation 3.2, then the public constant multiplication algorithm Π_{cmul} is used, then a truncation protocol. In practice, we choose $k = 8$.*

Another algorithm that is also from [Ryf19], approximates the logarithm using 6th-order modified Householder iterations [NNM07], we describe it in algorithm 11. Recall that the Householder method is a numerical analysis algorithm used to solve equations. In this case, the equation is:

$$1 - x \cdot \exp(-y) = 0, \tag{3.7}$$

and the goal is to find y which is equal to $\log(x)$. The algorithm uses the starting point $\frac{x}{31} + 1.59 - 20 \cdot \exp(-2x)$ for convergence.

Algorithm 11 Natural logarithm approximation algorithm $\Pi_{\log}$

Private input: Shares of the secret $[x]$
Public parameter: N of iterations k
Output: Shares $[y]$ such that $y = \log(x)$
1: $[y] \leftarrow \frac{x}{31} + 1.59 - 20 \cdot \Pi_{\exp}(-2\,[x] - 1.47)$ ▷ initial constant for convergence
2: **for** $i = 1..k$ **do**
3: $\quad [h] \leftarrow 1 - [x] \cdot \Pi_{\exp}([-y])$
4: $\quad [y] \leftarrow y - [h] \cdot (1 + \frac{[h]}{2} + \frac{[h]^2}{3} + \frac{[h]^3}{4} + \frac{[h]^4}{5} + \frac{[h]^5}{6})$
5: **end for**

For the real inverse and the real square root, the Newton-Raphson [Ypm95] algorithm is used. The corresponding algorithms are presented in algorithms 12 and 13 respectively, and they are used in many frameworks for secret-sharing-based multi-party computation like [Kno+21]. The inverse algorithm solves the equation:

$$x - \frac{1}{y} = 0. \tag{3.8}$$

The starting point of the Newton-Raphson algorithm for the inverse and the square root has to be chosen in a way that the sequence converges to $\frac{1}{x}$ and $\sqrt{x}$ respectively. We use the same starting points defined in [Kno+21]. More precisely, for the inverse, the starting point should be positive and and smaller than $\frac{2}{x}$.

Algorithm 12 Real inverse approximation algorithm Π_{inv}

Private input: Shares of the secret $[x]$
Public parameter: N of iterations k
Output: Shares $[y]$ such that $y = \frac{1}{x}$
1: $[y] \leftarrow 3 \cdot \Pi_{\exp}(\left[\frac{1}{2} - x\right]) + 0.003$ ▷ initial constant for convergence
2: **for** $i = 1..k$ **do**
3: $\quad [y] \leftarrow [y] \cdot [[-x] \cdot [y] + 2]$
4: **end for**

The square root algorithm converges toward the solution of the following equation:

$$x - \frac{1}{y^2} = 0, \tag{3.9}$$

which has a solution $y = \frac{1}{\sqrt{x}}$ and this is why the final result should be multiplied by x for the algorithm to return the desired value $\sqrt{x}$.

Algorithm 13 Real square root approximation algorithm Π_{sqrt}

Private input: Shares of the secret $[x]$
Public parameter: N of iterations k
Output: Shares $[y]$ such that $y = \sqrt{x}$
1: $[y] \leftarrow 5 \cdot \Pi_{\exp}([-0.7x - 0.6]) + 0.003$ ▷ initial constant for convergence
2: **for** $i = 1..k$ **do**
3: $\quad [y] \leftarrow 0.5\,[y] \cdot \left[[-x] \cdot [y]^2 + 3\right]$
4: **end for**
5: $[y] \leftarrow [y] \cdot [x]$

3.2.2 Comparison Protocols

Rabbit [Mak+21] is a relatively new secure comparison protocol that uses the commutative nature of addition over rings and fields. This protocol uses doubly secret-shared bits called daBits [RW19] which are bits secret-shared in both $\mathbb{Z}/2^l\mathbb{Z}$ and $\mathbb{Z}/2\mathbb{Z}$, as well as extended doubly secret-shared ring elements called edaBits [Esc+20], which refer to secret-shared integers in the arithmetic domain – the finite ring in our case – with its bit decomposition shared in the binary domain. More specifically, for a secret $[x]$ such that:

$$x = \sum_{i=1}^{l} x^i 2^i, x^i \in \{0, 1\}, \tag{3.10}$$

each bit x^i in also secret shared in $\mathbb{Z}_2$, and the secret is denoted as $\left[x^i\right]_2$. The Rabbit comparison is more efficient than prior secure comparison protocols.

Rabbit's protocol can be used to compare a secret shared ring element with a public constant. The protocol is called "less than constant", it is denoted by Π_{LTC}, and it is described in algorithm

26. The reader can find it in [Mak+21, Figure 4]. As input, the protocol takes the secret-shared element and the public constant that the players wish to compare, and the protocol returns shares of 1 if the secret is lower than the constant, and shares of 0 otherwise. We explain this algorithm in detail in section 6.2.1.

Another comparison protocol that we use in this thesis is based on functional secret sharing from [Ryf+22] is a two-party comparison protocol that requires the generation and the secret-sharing of correlated randomness, i.e. a set containing random elements and combinations thereof. It takes a secret shared element as input and returns shares of 1 if the secret is negative and shares of 0 otherwise, we illustrate it in algorithm 16. It uses two functions called KeyGen and Eval from [Ryf+22, algorithm 3 and 4] respectively, we only define their respective inputs, outputs in algorithms 14 and 15. The former generates the correlated randomness, while the former returns the sign of the secret using the random elements that were generated. They are defined as follows:

Algorithm 14 Key generation for the comparison function denoted KeyGen from [Ryf+22, algorithm 3]

Private input:

Public parameter: λ a security parameter set usually equals 128

Pseudo-random generator: $G(s)$ is a pseudo-random bit-string generator with seed s indistinguishable from random

Output: A pair of keys (k_0, k_1) such that the first l bits of the keys is α_0, α_1 respectively

Sample random $\alpha \in \mathbb{Z}_{2^l}$
Sample random $s_j^{(1)} \in \{0,1\}^\lambda$ and $t_j^{(1)} \leftarrow j$, for $j = 0, 1$
for $i = 1 \dots l$ **do**
  for $j = 0, 1$ **do**
    $(s_j^L || t_j^L, s_j^R || t_j^R), (\sigma_j^L || \tau_j^L, \sigma_j^R || \tau_j^R) \leftarrow G(s_j^{(i)}) \in \{0,1\}^{\lambda+1} \times \{0,1\}^{\lambda+1} \times \{0,1\}^{l+1} \times \{0,1\}^{l+1}$
  end for
  if $\alpha[i]$ **then**
    $cw^{(i)} \leftarrow \big((0^\lambda || 0, s_0^L \oplus s_1^L || 1), (\sigma_0^R \oplus \sigma_1^R || 1, 0^\lambda || 0)\big)$
  else
    $cw^{(i)} \leftarrow \big((s_0^R \oplus s_1^R || 1, 0^\lambda || 0), (0^\lambda || 0, \sigma_0^L \oplus \sigma_1^L || 1)\big)$
  end if
  $CW^{(i)} \leftarrow cw^{(i)} \oplus G(s_0^{(i)}) \oplus G(s_1^{(i)})$
  for $j = 0, 1$ **do**
    $state_j \leftarrow G(s_j^{(i)}) \oplus (t_j^{(i)} \cdot CW^{(i)}) = ((state_{j,0}, state_{j,1}), (state'_{j,0}, state'_{j,1}))$
    Parse $s_j^{(i+1)} || t_j^{(i+1)} = state_{j,\alpha[i]}$ and $\sigma_j^{(i+1)} || \tau_j^{(i+1)} = state'_{j,1-\alpha[i]}$
  end for
  $CW_{leaf}^{(i)} \leftarrow (-1)^{\tau_1^{(i+1)}} \cdot \left(\alpha[i] - \sigma_0^{(i+1)} + \sigma_1^{(i+1)}\right) \mod 2^l$
end for
$CW_{leaf}^{(l+1)} \leftarrow (-1)^{t_1^{(n+1)}} \cdot \left(1 - s_0^{(n+1)} + s_1^{(n+1)}\right) \mod 2^l$
$(\alpha_0, \alpha_1) \leftarrow \Pi_{\text{share}}(\alpha)$
$k_j \leftarrow \alpha_j || s_j^{(1)} || (CW^{(i)})_{i=1\dots n} || (CW_{leaf}^{(i)})_{i=1\dots n+1}$, for $j = 0, 1$

Lemma 16. *[Ryf+22, section 3.2.2] Algorithm 14 outputs a pair of keys (k_0, k_1) which are correlated random elements, and the first l bits of the keys k_0, k_1 are respectively α_0, α_1, shares of a random ring element $\alpha \in_R \mathbb{Z}_{2^l}$.*

Algorithm 15 Evaluation of the function key for comparison denoted Eval

Private input: $j \in \{0,1\}$ where j refers to the player P_j,
k_j is the corresponding key for the comparison, the keys are generated with algorithm 14 then distributed to the players by a trusted party,
x is a public element in $\mathbb{Z}_{2^l}$
Pseudo-random generator: $G(s)$ is a pseudo-random bit-string generator with seed s indistinguishable from random

Output: A share $z_j = \text{Eval}(j, k_j, x)$, where Eval is the function defined in [Ryf+22, algorithm 4]

1: Parse k_j as $\alpha_j || s_j^{(1)} || (CW^{(i)})_{i=1\ldots n} || (CW_{leaf}^{(i)})_{i=1\ldots n+1}$
2: Let $t^{(1)} \leftarrow j$
3: **for** $i = 1 \ldots l$ **do**
4: $\quad state \leftarrow G(s^{(i)} \oplus (t^{(i)} \cdot CW^{(i)}) = ((state_0, state_1), (state'_0, state'_1))$
5: $\quad$ Parse $s^{(i+1)} || t^{(i+1)} = state_{x[i]}$ and $\sigma^{(i+1)} || \tau^{(i+1)} = state'_{x[i]}$
6: $\quad out_i \leftarrow (-1)^j \cdot \left(\tau^{(i+1)} \cdot CW_{leaf}^{(i)} + \sigma^{(i+1)}\right) \mod 2^l$
7: **end for**
8: $out_{n+1} \leftarrow (-1)^j \cdot \left(t^{(n+1)} \cdot CW_{leaf}^{(n+1)} + \sigma^{(n+1)}\right) \mod 2^l$
9: $z_j \leftarrow \sum_i out_i \mod 2^l$

Lemma 17. *[Ryf+22, section 3.2.2] Algorithm 15 outputs shares of $z = (x \leq \alpha)$, i.e:*

$$z_0 + z_1 = \text{Eval}(0, k_0, x) + \text{Eval}(1, k_1, x) = (x \leq \alpha) \mod 2^l, \tag{3.11}$$

where $(x \leq \alpha) \in \{0,1\}$. The algorithm 15 is deterministic and non-interactive i.e. the players do not communicate with each other.

Algorithm 16 Functional secret sharing comparison protocol Π_{comp}

Private input: k_j is the random key of player P_j, the keys are generated with algorithm 14 then distributed to the players by a trusted party,
$[y]$ is a secret in $\mathbb{Z}_{2^l}$

Output: Shares of $[z]$

1: P_j computes α_j the first l bits of k_j
2: $x = \Pi_{\text{open}}([\alpha + y])$
3: P_j computes $z_j \leftarrow \text{Eval}(j, k_j, x)$ ▷ Eval function in defined in algorithm 15

Lemma 18. *[Ryf+22, section 3.2.2] Given shares of a secret $[u]$, algorithm 16 returns shares of $[z]$ such that $z = (y \leq 0)$.*

Chapter 4

Differentially-private and Fully Secure Logistic Regression

This work was done with Kumari Vaibhavi[1] *and Stéphane Bressan*[2] *and published in [SKB22].*

Modern organizations train and use machine learning models to support business decisions and optimize processes. Collaborating organizations, for instance, multiple government agencies, may see the need and the benefits of federatively building such models and, possibly, sharing them with third parties. In this process, the privacy of the data used to train the models needs to be protected against attacks not only from members of the collaborating organizations among themselves but also from third-party users of the models. Indeed, membership inference attacks [Sho+17] show that machine learning models may unwantedly reveal information about the training data.

Secure multi-party computation and ϵ-differential privacy devise solutions to address the issues of protection from collaborating organizations and from users of the models separately. Can these be combined to form a unified solution? We hereby propose, present, and evaluate a two-party ϵ-differentially private and fully secure logistic regression on a vertically split dataset. In this process, the players need to jointly train a model, rather than independently train models from their local data as horizontal federated learning players do.

4.1 Model Definition

We consider two players, P_1 and P_2, representing two collaborating organizations, who jointly own a vertically split tabular dataset $D \in \mathbb{R}^{r \times d+1}$ containing private data with r entries represented in rows, d feature columns, and a label column for the sake of the classification task. P_1 owns D^1 which are d^1 columns of the dataset D, and P_2 owns D^2, d^2 columns of D - including the label column - with $d^1 + d^2 = d + 1$ and $D^1||D^2 = D$. We also consider a user, U, consuming the model. The reader notices that the players, P_1 and P_2, are also potential users of the model. Their common goal is to compute a logistic regression on the whole dataset such that P_i doesn't reveal D^i to the other player, and that the result is ϵ-differentially private. More precisely, their goal is to obtain the model $W^* \in \mathbb{R}^d$ (or weights) produced by a logistic regression performed on the concatenated dataset with respect to the classes defined by the label column. As explained in section 1.3.1 we should satisfy these privacy specifications:

[1] kvaibhavi@nus.edu.sg
[2] https://www.comp.nus.edu.sg/~steph/

- Passive security as per definition 3 – achieved by using exclusively secret-sharing- based multi-party computation.
- The public model output has proven ϵ-differential privacy.
- The public model output has experimentally good accuracy due to the usage of functional mechanism [Zha+12], [CMS11].

To do so we consider the secure multi-party computation model where P_1 and P_2 secret-share their data with each other and evaluate together the required protocols to compute the differentially private logistic regression described in algorithm 9.

In some protocols, we consider that players are initialized, in a setup phase, with correlated randomness. This data is independent of their inputs and outputs. Thus it can be for example generated between them using what is known as a "pre-processing" protocol [Dam+19]. For efficiency, we suppose that the correlated randomness is already generated and distributed amongst the players.

Note that the solution we present in this chapter can function with any partitioning of the training dataset.

For prerequisites on logistic regression, the reader is referred to section 2.3.2.2 and for training an ϵ-diferentially private logistic regression using functional mechanism [Zha+12], the reader is referred to sections 2.4.1 and 2.4.2. A comparative list of the related works is given in table 1.1.

4.2 Proposed Approach

The steps for adapting the logistic regression algorithm to secure multi-party computation are the following. First, the data normalization is done locally by P_1 and P_2. As the data is split vertically, the normalization part does not need information from the columns of the other party. Second, real decimal values are mapped to $\mathbb{Z}/2^l\mathbb{Z}$. This is done automatically in PySyft, which is a Python library for secure and private deep learning that we use to implement the proposed approach in this chapter. Real numbers are represented in the ring $\mathbb{Z}/2^l\mathbb{Z}$ with $l \in \{32, 64\}$. Third, secret sharing, addition, and multiplication protocols are already implemented in PySyft, and they are used to compute the regression. Finally, the accuracy is computed using the appropriate algorithm and the players open the weights W^* by adding their shares together. The operations in the logistic regression are computed by evaluating consecutively addition, multiplication, and comparison protocols.

4.2.1 Normalization of the Dataset

First, each player, P_1 and P_2, normalize the columns of their respective dataset in clear locally, so that the values of their column are real elements between 0 and 1. They do so by computing for each column $i \in \{1, ..., d\}$ the minimum m^i and maximum M^i elements of the column i, then they subtract m^i from each element of the column i and they divide the result by $M^i - m^i$.

The second step of normalization is required for the functional mechanism, in their paper [Zha+12], the authors state that the sum of the squares of the elements of each row should be less than 1. To do this normalization locally, the players divide each element of their dataset by $\sqrt{d}$. As the elements were less than 1 after the first normalization, dividing them by $\sqrt{d}$ makes them all smaller than $\frac{1}{\sqrt{d}}$, therefore, adding the squares of the elements of a row will be less than 1.

Remark 19. *Since the solution that we propose relies on secret sharing, it could be applied not only to vertically split datasets but also to datasets with any partitioning. Normalizing the columns of the dataset would then be done in a secret-shared manner using secure operations like comparisons to compute the maximum and minimum elements of the column. Since normalizing the rows of the dataset is done locally, it would not be affected by the separation of the dataset.*

4.2.2 Secure Computation of the Sigmoid Function

Computing a logistic regression requires at some point computing the function sigmoid:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{4.1}$$

This function contains an exponentiation and a division which are expensive to compute using secure multi-party computation. The idea is to approximate the sigmoid function with additions, multiplications, and comparisons. We use a secure multi-party computation-friendly approximation of the sigmoid function, which we denote S and define below, and is linear in three pieces. A linear by-parts approximation was introduced in [MZ17]. For $x \in \mathbb{R}$:

$$S(x) = \begin{cases} 0 & \text{if } x < -2 \\ \frac{x}{4} + \frac{1}{2} & \text{if } -2 \leq x \leq 2 \\ 1 & \text{if } x > 2. \end{cases} \tag{4.2}$$

Our reason for using this approximation is for increased performance. Indeed, there exists a secure multi-party computation protocol for comparisons from [Ryf+22] that we illustrated in algorithm 16 and that we denote by Π_{comp} and that is based on "function secret sharing" (FSS). This protocol is available in the PySyft 0.2.9 library, it performs a secure comparison in only one round in the online phase. Therefore we use the algorithm Π_{comp} to construct an algorithm Π_{sigmoid} that outputs the result of the function S defined in 4.2 as the following. We recall that real decimals are mapped to the finite ring, but for the sake of simplicity, we refer to both a real element and its image in $\mathbb{Z}_{2^l}$ using the same notation. So given a secret $[x]$, Π_{sigmoid} securely outputs $[S(\tilde{x})]$ by linearizing the function S:

$$S(x) = (1 - (x < -2)) \left(\left(\frac{x}{4} - \frac{1}{2} \right) (x < 2) + 1 \right). \tag{4.3}$$

Algorithm 17 Sigmoid protocol Π_{sig}

Private input: The players hold shares of $[x]$
Output: The players hold shares of $[y]$

1: $[u] \leftarrow \Pi_{\text{comp}}([x], -2)$ ▷ Π_{comp} from algorithm 16
2: $[v] \leftarrow \Pi_{\text{comp}}([x], 2)$
3: $[y] \leftarrow \left[\frac{x-2}{4}\right]$
4: $[y] \leftarrow [1 - u] \cdot [[y] \cdot [v] + 1]$

Lemma 20. *The secure multi-party approximation of the sigmoid protocol described in algorithm 17 returns shares of* $[S(x)]$.

Proof. If $x < -2$, then $u = 1$ and Π_{sig} would return 0. If $-2 < x < 2$, then $u = 0$ and $v = 1$, which results in Π_{sig} returning $\frac{x-2}{4} + 1$ which is equal to $\frac{x}{4} + \frac{1}{2}$. If $x > 2$, then $u = 0$ and $v = 0$ which returns 1. Therefore, Π_{sig} implements then approximation function S. □

As a result of using Π_{sig}, the overall secure multi-party computation of S, shown in algorithm 17 and denoted as Π_{sig}, requires a small number of interactions in the online phase – where the evaluation of the function takes place, as opposed to the offline phase.

4.2.3 Generation of Random Noise

The works [Thi+19; Wu+16] present many algorithms for noise generation. In order to achieve differential privacy, the Laplace noise used in the gradient descent 2.31 should be unknown to the user of the model U, but also to P_1, P_2 as they are potential users of the model.

Therefore, we present an algorithm that generates shares of Laplace noise for P_1 and P_2, still without them knowing the value of this noise. We use a standard technique in which we proceed in two steps. The first step is an algorithm that generates shares that correspond to a uniformly random element in $[0, 1]$ unknown to both parties. In the second step, the players obtain a shared random element that follows a Laplace distribution using the secret uniformly random element that they had generated.

More precisely, in the first step the players compute $[u] : (u_1, u_2)$ where $u \sim \mathcal{U}(0, 1)$ is done with the algorithm denoted as Π_{unif} in which P_1 and P_2 generate two elements $a, b \sim \mathcal{U}(0, 1)$ and secret-share them. Then the players compute shares of u using Π_{comp} and equation 4.4 as shown in algorithm 18.

$$u = \begin{cases} a + b & \text{if } a + b < 1 \\ a + b - 1 & \text{if not.} \end{cases} \tag{4.4}$$

Algorithm 18 Generation of uniformly random elements protocol Π_{unif}

Private input:

Output: The players hold shares of $[u]$

1: P_1 generates $a \sim \mathcal{U}(0, 1)$ and secret-shares it
2: P_2 generates $b \sim \mathcal{U}(0, 1)$ and secret-shares it
3: $[u] \leftarrow [a + b]$
4: $[u] \leftarrow [u] + \Pi_{\text{comp}}([u], 1) - 1$

Our implementation of the algorithm Π_{unif} takes one round of communication since we use protocol Π_{comp} from [Ryf+22] illustrated in algorithm 16 (available in the PySyft library) for comparisons.

Subsequently, in the second step, given $[u]$ the next step is to apply Π_{Lap} described in algorithm 19 to compute shares of the Laplacian noise $\left[F^{-1}(u)\right]$ where F is the cumulative distribution function of Laplace.

$$l = F^{-1}(u) = -\beta \operatorname{sign}(u - \frac{1}{2}) \log(1 - 2|u - \frac{1}{2}|). \tag{4.5}$$

This requires computing the natural logarithm, and in order to do so we use the sub-algorithm $\Pi_{\log}$ illustrated in algorithm 11 that is already implemented in the PySyft library.

Obtaining $l \sim \mathcal{L}(0, \beta)$ using $u \sim \mathcal{U}(0, 1)$ and the function F^{-1} is a known technique [Thi+19] used to compute random elements following a Laplace distribution in a secure multi-party computation manner. Notably, in [Thi+19, equation 6.1], the author computes an equivalent form of F^{-1} in which the sign function is replaced by a generation of a random bit.

For convenience, we directly used the implementation of logarithm in PySyft. Notice that an alternative implementation of the natural logarithm is presented in [Thi+19] also for the purpose of generating Laplace noise in secure multi-party computation.

Algorithm 19 Generation of random Laplace elements protocol Π_{Lap}

Private input: The standard deviation $\beta \in \mathbb{R}$
Output: The players hold shares of $[l]$

1: The players generate shares of a uniformly random element $[u]$ using Π_{unif}
2: $[s] \leftarrow 2 \cdot \Pi_{\text{comp}}(\left[u - \frac{1}{2}\right], 0) - 1$
3: $[t] \leftarrow [s] \cdot \left[(u - \frac{1}{2})\right]$
4: $[l] \leftarrow -\beta \cdot [s] \cdot \Pi_{\log}([-2t + 1])$

4.2.4 Training Algorithm in a Two-party Setting

Using the algorithms that were defined in this chapter, a training algorithm that uses the functional mechanism and private data can be defined. We present it in algorithm 20. The final result - the optimized weights W^* - could be left in a shared form and used later for multi-party predictions.

Algorithm 20 Differentially-private logistic regression with secure MPC Π_{train}

Private input: Players P_1, P_2 respectively hold their secret dataset inputs D^1, D^2
Public parameters: ϵ the privacy budget,
$\Delta = 3d + 0.25d^2$ from equation 2.30, where $d + 1$ is the number of columns in $D = D^1||D^2$,
α the learning rate,
epochs the number of epochs,
r the number of rows in the training dataset
Output: The public model output W^*

1: Normalize data locally ▷ as described in section 4.2.1
2: Split data into training and testing data
3: $[D] \leftarrow \Pi_{\text{share}}(D^1)||\Pi_{\text{share}}(D^2)$
4: $\beta \leftarrow \frac{\Delta}{\epsilon} \cdot epochs$
5: $[x] \leftarrow$ concatenated feature columns of the training data from $[D]$
6: $[y] \leftarrow$ the label column of the training data from $[D]$
7: Initialize secret-shared weights $[W]$ with normal distribution
8: **for** every *epoch* **do**
9: $\quad [\eta_1] \leftarrow \Pi_{\text{Lap}}(\beta)$
10: $\quad [\eta_2] \leftarrow \Pi_{\text{Lap}}(\beta)$
11: $\quad [\nabla W] \leftarrow \left[\left[x^T\right] \cdot [1/2 - y] + \eta_1 + 2\left[\frac{\left[x^T\right] \cdot [x]}{8} + \eta_2\right] \cdot [W]\right]$
12: $\quad [W] \leftarrow [W] - \alpha \cdot \left[\frac{\nabla W}{r}\right]$
13: **end for**
14: $[W^*] \leftarrow [W]$
15: $W^* \leftarrow \Pi_{\text{open}}([W^*])$

Theorem 21. *Algorithm 20 computes a logistic regression model that satisfies the following two privacy specifications:*

1. *Passive security as per definition 3 and in the preprocessing model of [CDN15, chapter 8.5] and [Ryf+22].*
2. *ϵ-differential privacy as per definition 12.*

Proof. The secure multi-party computation protocols used in algorithm 20 guarantee the security and correctness of the result, furthermore ϵ-differential privacy is guaranteed by the use of the functional mechanism like in theorem 14. □

Remark 22. *When real elements are mapped to a finite ring, fixed-bit precision is used, meaning a fixed number of decimals are kept and the rest is truncated. Note that this does not affect the guarantees from ϵ-differential privacy.*

The accuracy of the model output is computed by testing the model on a number of unseen data points, more precisely the accuracy corresponds to the number of well-predicted labels divided by the total number of data points that it was given. To compute the accuracy of the model $[W^*]$ without revealing their testing data, the players can use secure multi-party computation to know the accuracy γ of the model using the testing data $[x_{test}], [y_{test}]$, and the optimized weights $[W^*]$ as follows:

$$\gamma = 1 - \text{mean}(\text{round}(|\,\text{sigmoid}(x_{test} \cdot W^*) - y_{test}|)). \tag{4.6}$$

The algorithm that could be used to test the accuracy of the model uses the algorithm Π_{sigmoid} to compute the sigmoid function and the comparison protocol Π_{comp} to perform the absolute value and rounding operations. These operations are computed using the following equations, given $x \in \mathbb{R}$, and $y \in [0, 1]$:

$$\text{abs}(x) = 2x(x > 0) - x \text{ and } \text{round}(y) = (0.5 > y). \tag{4.7}$$

4.3 Experimental Results

We implement our work in PySyft (version 0.2.9). The PySyft library includes secure multi-party computation protocols that we will be using. We use the PySyft framework with two players, P_1 and P_2.

We perform two kinds of evaluation to check the effectiveness of our approach. First, we compare the accuracy of the following four logistic regression models: the original logistic regression, referred to as "Standard", the secure multi-party logistic regression, referred to as the "MPC", the implementation of a functional mechanism-based ϵ-differential private logistic regression is referred to as "DP", and the proposed secure multi-party functional mechanism-based ϵ-differential private logistic regression is referred to as "DP+MPC". "Standard" and "MPC" are independent from the parameter ϵ.

4.3.1 Brazil Dataset

We use the Brazil census dataset from IPUMS [IPU] for our first comparative experiment. We normalize each feature line to ensure that it lies within the range of [-1, 1]. After the cleaning and preprocessing – i.e. remove rows and columns –, we chose a subset of the Brazil dataset with 14,000 rows and five columns. We also normalize the samples in each dataset such that $||x^i||_2 \leq 1$ for $i \in 1, ..., r$.

The accuracy γ is measured as the average number of times the model is able to assign correct values to the target column for all new (unseen) data points. The regression is run 100 times on the same data, for 10 epochs each time, with all the data in one batch.

The figures compare the accuracy achieved by the four models mentioned above. It shows the accuracy of the models on varying the value of ϵ from 0.01 to 1. Naturally, the accuracy of the original and the secure multi-party computation model does not depend on the perturbation introduced by the differential privacy mechanism as seen in figure 4.1. The ϵ-differential private mechanism

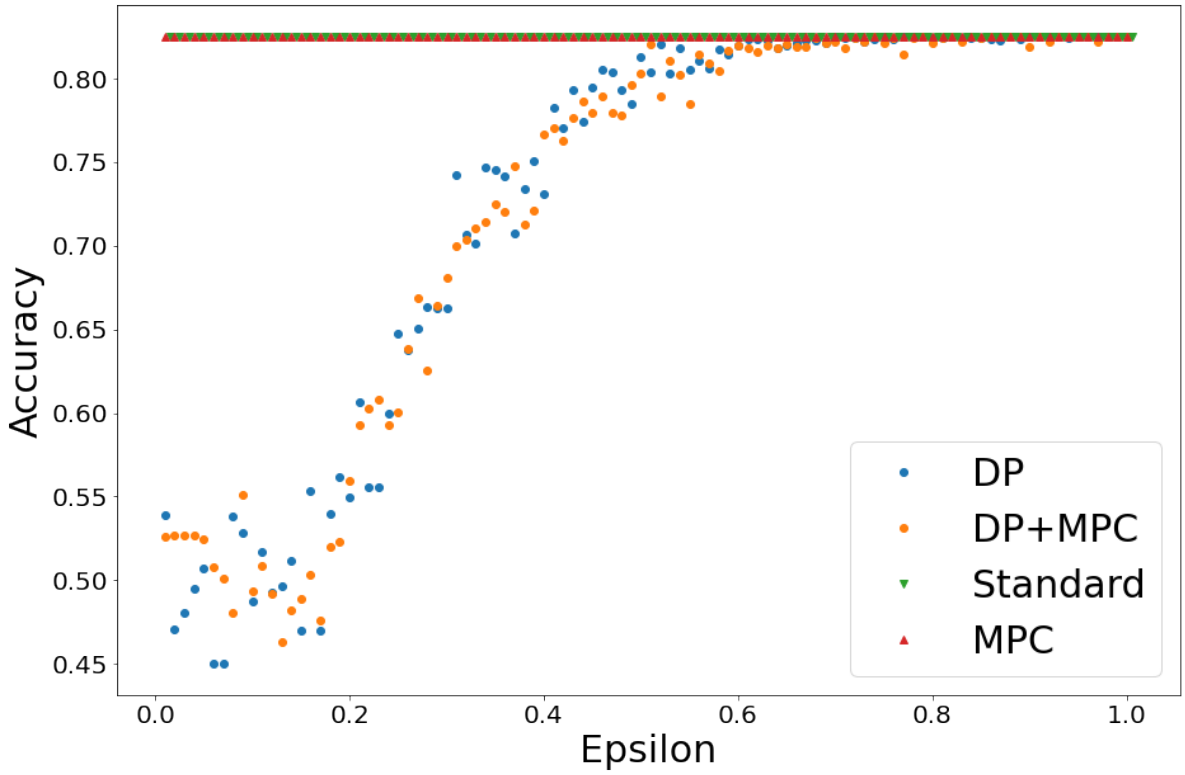

Figure 4.1: The average accuracy of models trained on the Brazil dataset 100 times. Standard and DP models were trained in clear on real numbers, whereas MPC and DP+MPC use $\mathbb{Z}_{2^l}$ for the multi-party computation protocols. The MPC and standard models do not depend on ϵ, and the average accuracy of the models trained without differential privacy is the same because no randomness was introduced in the algorithm. The DP and DP+MPC points correspond to the average accuracy of the models trained with differential privacy while varying the parameter ϵ. The accuracy of these models does not coincide as these algorithms depend on random noise. Indeed, for every value of ϵ the model is trained and tested 100 times, but the accuracy of each trained model depends on the Laplacian noise which was injected into the algorithm, and it is different for each of the 100 regressions.

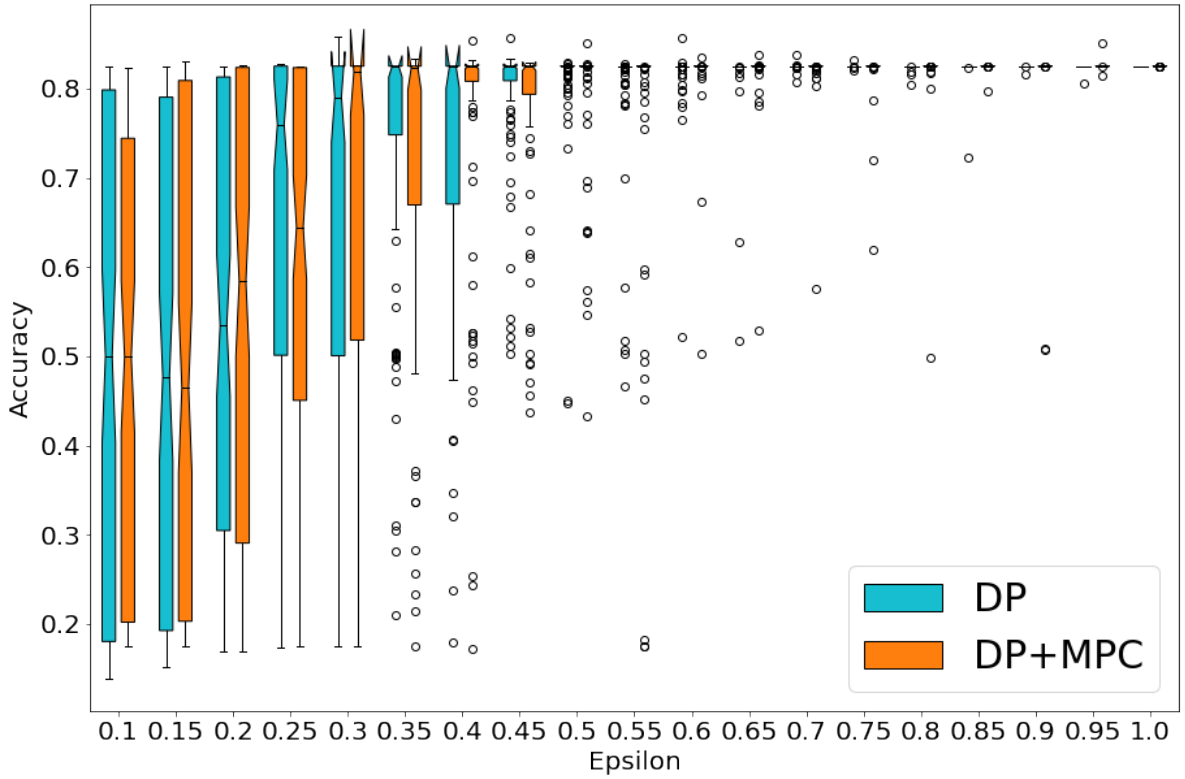

Figure 4.2: Distribution of the accuracy of each of the 100 models trained on the Brazil dataset in two settings, with and without MPC. The training of the DP models was done in clear on real numbers. The circle points represent outliers, and the lower and upper bars represent the limit of the outliers on the accuracy of the 100 models. The other values of the boxplot represent the first quartile, the median, and the third quartile, which respectively are where $25\%, 50\%$ and 75% of the accuracy of the 100 trained models falls. We notice that when ϵ gets closer to 1 the accuracy of the trained models converges to the accuracy of the model trained in clear because the noise injected in the algorithm becomes less important; ϵ is inversely proportionate to the noise.

introduces a trade-off between the privacy parameter ϵ and the accuracy for ϵ-differentially private models both without and with secure multi-party computation. The accuracy of both these models improves as the value of ϵ increases. This is in accordance with the theory that a stronger privacy guarantee results from extra noise that deteriorates the utility. Figure 4.2 gives a boxplot that indicates the minimum, maximum, quartiles, and the lower and upper fences that represent the cut-off values for upper and lower outliers on the accuracy of the model after each of the 100 regressions. This accuracy was achieved by the DP model and the DP+MPC model for 100 runs. Small values of ϵ show significant variance in the accuracy of the models. As the privacy parameter is relaxed, the variance in the accuracy becomes less, and the median accuracy becomes closer to the actual accuracy of the model.

The goal was to add secure multi-party computation to the differential privacy model without impacting the accuracy, and it is achieved. Combining the secure multi-party computation model and ϵ-differential privacy into the mechanism we propose in this paper, we see that we can achieve a fully secure and privacy-preserving mechanism at a good economy of accuracy.

4.3.2 Adult Dataset

We use the Adult census dataset for our second comparative experiment. We normalize each feature line to ensure that it lies within the range of [-1, 1]. After normalizing the numerical features and encoding all the relevant categorical features of the Adult dataset, we have a dataset with 18,000 rows and eight columns. We also normalize the samples in each dataset such that $||x^i||_2 \leq 1$ for $i \in 1, ..., r$.

The figures compare the accuracy achieved by the four models mentioned above. It shows the accuracy of the models on varying the value of ϵ from 0.01 to 1, which are standard measures of ϵ in the state-of-the-art. Naturally, the accuracy of the original and MPC model does not depend on the perturbation introduced by the differential privacy mechanism as seen in figure 4.3. Figure 4.4 gives a boxplot of the accuracy achieved by the DP and the DP+MPC models for 100 regression runs. Same as before, when the privacy parameter is low, the variance in accuracy is high, but as it is increased, the variance decreases and the median accuracy gets closer to the true accuracy of the model.

4.4 Further Discussion and Conclusion

We have proposed, presented, and evaluated an original ϵ-differentially private and fully secure logistic regression on a vertically split dataset. The results can also be applied to any dataset separation because the players secret-share their entire dataset and then concatenate it as they wish. The concatenation does not affect in any way to rest of the algorithm and the security and privacy guarantees.

As explained, secure multi-party computation ensures security when performing computations. However, it does not guarantee user data privacy if the output of the computation is made public. Adding a layer of differential privacy over secure multi-party computation guarantees additional data privacy.

The solution proposed utilizes the functional ϵ-differential privacy mechanism and a fully secure original multi-party gradient descent computation for the logistic regression.

The functional mechanism for differential privacy consists of a trade-off between privacy and accuracy. We have used secret-sharing-based secure multi-party computation without homomorphic encryption for the advantages that secure multi-party computation offers in terms of efficiency. While the functional mechanism guarantees the model's privacy by adding noise to the objective

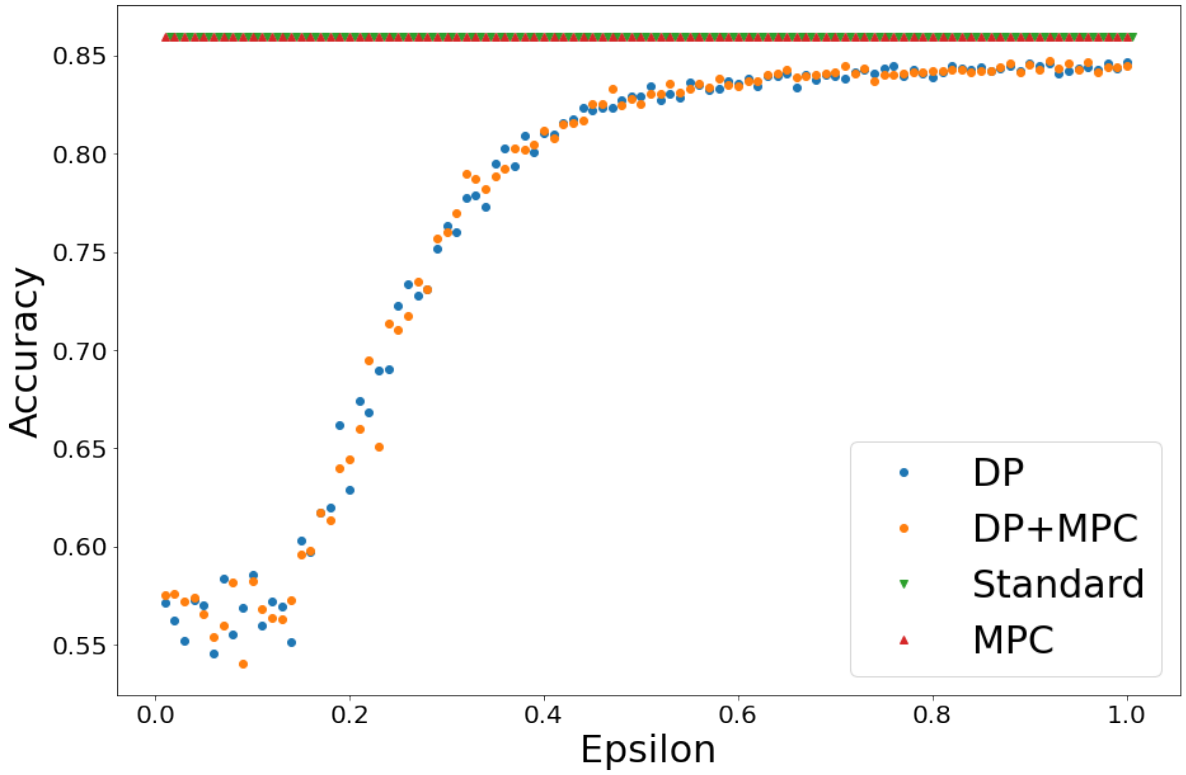

Figure 4.3: The average accuracy of 100 models trained on the Adult dataset. Standard and DP models were trained in clear on real numbers, whereas MPC and DP+MPC use $\mathbb{Z}_{2^l}$ for the multi-party computation protocols. The MPC and standard models do not depend on ϵ, and the average accuracy of the models trained without differential privacy is the same because no randomness was introduced in the algorithm. The DP and DP+MPC points correspond to the average accuracy of the models trained with differential privacy while varying the parameter ϵ. The accuracy of these models does not coincide as these algorithms contain random noise. Indeed, for every value of ϵ the model is trained and tested 100 times, but the accuracy of each trained model depends on the Laplacian noise which was injected into the algorithm, and it is different for each of the 100 regressions.

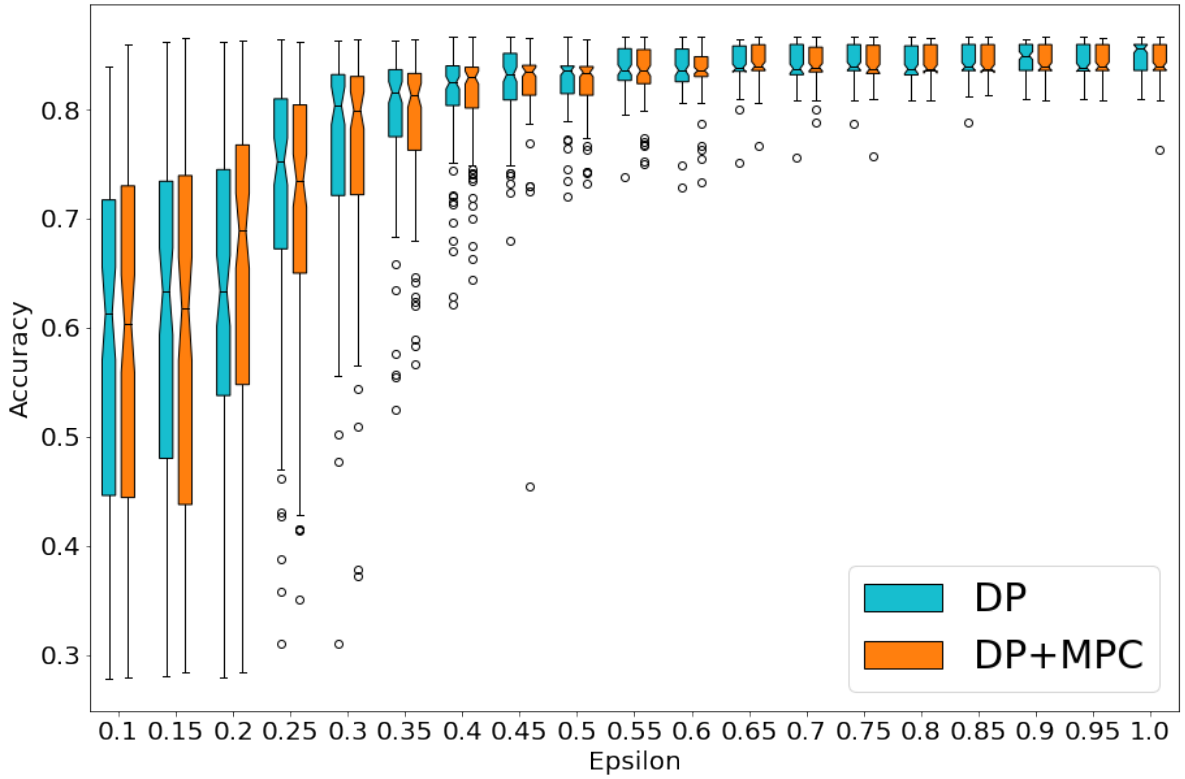

Figure 4.4: Distribution of the accuracy of each of the 100 models trained on the Brazil dataset in two settings, with and without MPC. The training of the DP models was done in clear on real numbers. The circle points represent outliers, and the lower and upper bars represent the limit of the outliers on the accuracy of the 100 models. The other values of the boxplot represent the first quartile, the median, and the third quartile, which respectively are where $25\%, 50\%$ and 75% of the accuracy of the 100 trained models falls. We notice that when ϵ gets closer to 1 the accuracy of the trained models converges to the accuracy of the model trained in clear because the noise injected in the algorithm becomes less important; ϵ is inversely proportionate to the noise.

function, which affects the model's accuracy, secure multi-party computation is a tool that will only compute the desired function without seeing the input without affecting the result of the computation.

Our empirical studies on the dataset show that the impact of the addition of secure multi-party computation over differential privacy is very close to training using the concepts of differential privacy only. The evaluation of performance and the comparative discussion of the proposed mechanism show that it constitutes an original effective, efficient, secure, and privacy-preserving solution.

It is possible to modify the algorithms defined in this thesis to replace the functional mechanism with the differentially private stochastic gradient descent from [Aba+16]. This method consists of clipping the gradients over each individual data point and then adding noise to it. The added noise could be Laplacian, which is the noise that we generate in this chapter for the functional mechanism. Moreover, the model output would satisfy ϵ-differential privacy.

We chose to compute a logistic regression because it constitutes a fundamental building block for more complex machine learning models. We are now considering the transposition of the original contributions of this paper to privacy-preserving and secure multi-layer perceptrons and neural networks.

Chapter 5

Truth Finding with Secure Multi-party Computation

This work was done with Pierre Senellart[1] *and Stéphane Bressan*[2].

Federated knowledge discovery and data mining [LDZ22; Yu+22] are challenged to assess the trustworthiness of data originating from autonomous sources while protecting confidentiality. Truth-finding algorithms [Li+15] help corroborate data from disagreeing sources. For each query it receives, a truth-finding algorithm predicts a truth value of the answer, possibly updating the trustworthiness factor of each source. Few works, however, address the issues of confidentiality and privacy. We consider the design and implementation of truth-finding algorithms that protect the confidentiality of sources' data.

The secure multi-party protocols are then implemented on two non-colluding servers. We empirically evaluate the performance of the proposed protocol on two state-of-the-art truth-finding algorithms, Cosine [Gal+10, Algorithm 1] and 3-Estimates [Gal+10, Algorithm 4] (see also [Li+12; Ber15] for further experiments on these algorithms), and compare them with that of the non-secure baseline algorithms. The results confirm that the secure multi-party algorithms are as accurate as the corresponding baselines except for proposed modifications to reduce the efficiency loss incurred.

5.1 Model Definition

Set $r \in \mathbb{N}^*$, and let $\mathcal{V}$ be a set of r sources. A "client" would like to label r queries (or facts) $\{f^1, \ldots, f^d\}$. A truth-finding algorithm outputs a truth value for a query when the sources provide disagreeing answers to it. Concretely, the truth-finding algorithm takes $v^1, \ldots, v^r$ as input with $v^i \in \{-1, 0, 1\}^d$, and outputs estimated truth values in $[-1, 1]^d \subset \mathbb{R}^r$ or $[0, 1]^d \subset \mathbb{R}^d$ depending on the truth-finding algorithm.

Truth-finding (or truth discovery) algorithms [Li+15] are usually run by the client in order to know the truth value of a given query when the sources give disagreeing answers. That is, for each of the client's queries, each source in $\mathcal{V}$ delivers an answer v^i such that an output of 1 corresponds to a positive answer, -1 to a negative one, and 0 if the source does not wish to classify the data point.

The goal of this work is to execute the two truth-finding algorithms Cosine and 3-Estimates – described in algorithms 7 and 8 respectively – while using secure multi-party computation so that

[1] `https://pierre.senellart.com/`
[2] `https://www.comp.nus.edu.sg/~steph/`

the sources do not disclose their answers.

The secure multi-party computation model consists of having the sources secret-share their answers on two servers that execute the multi-party computation of the truth-finding algorithm. The two algorithms Cosine and 3-Estimates will be executed in the same secure multi-party computation model.

For details about these algorithms, the reader is referred to the background chapter, section 2.3.2.4, and works related to truth-finding can be found in section 1.2.2.

5.2 Proposed Approach

The first task we wish to achieve is private voting, i.e. the client sends queries to each source, and the source classifies the query. In the case where the query is a vector of features and the models are logistic regressions, existing secure multi-party computation works [MR18] can keep the query private. We suppose that the answers are already computed and secret-shared by the sources on two servers P_1 and P_2 using a two-party additive secret sharing. In other words, P_1 holds v_1^{ij} and P_2 holds v_2^{ij} such that $v^{ij} = v_1^{ij} + v_2^{ij} \mod 2^l$ (in the ring $\mathbb{Z}_{2^l}$) is the ith source's answer for the query f^j and is equal to -1, 0, or 1.

The second step which is the aggregation of the data (the answers) is computed on the two servers P_1 and P_2. The problem is now constructing a secure two-party computation algorithm with additively shared data that implements the truth-finding algorithms using their arithmetic circuits. Once the circuits are evaluated, the two servers (P_1 and P_2) send their share of the output to the client who reconstructs it by adding the received shares together.

Furthermore, we use additive secret-sharing-based secure multi-party computation, and the protocols are computed in a finite ring. However, just like for machine learning applications the inputs and the operations in truth-finding algorithms are in $\mathbb{R}$. We, therefore, map the real data inputs to the finite ring using a fixed point precision [CS10], i.e. the number of decimals is fixed to a certain number of bits. For simplicity, we refer with the same notation to the real inputs and their ring mapping. Real addition and multiplication operations are approximated by other operations as we will explain in chapter 6, but for simplicity, we also refer to them with the same notation.

Other than additions and multiplications, the truth-finding algorithms we implement – Cosine and 3-Estimates – use three real-number operations: division, square root, and conditioned sums. We now explain how these three operations can be approximated using arithmetic circuits consisting of additions and multiplications. Furthermore, multiplications are communication-costly in secure multi-party computation, so the aim is to use a low number of multiplications – or communications in general.

5.2.1 Division and Square Root

The approximations given below are based on numerical methods. These approximations are widely used in secure multi-party computation frameworks for machine learning, like in [Ryf+18; Kno+21], where the real inverse and the square root are computed iteratively using the Newton-Raphson method. Suppose P_1 holds a_1 and P_2 holds a_2 such that $a = a_1 + a_2$. Then we denote by $\Pi_{\text{sqrt}}(a_1, a_2)$ the protocol that allows each player to hold b_1 and b_2 respectively without learning information about a_1 and a_2, and where $b_1 + b_2 = \sqrt{a}$. Similarly, we denote by $\Pi_{\text{inv}}(a_1, a_2)$ the protocol that allows each player to hold c_1 and c_2 respectively without learning information about a_1 and a_2, and where $c_1 + c_2 = \frac{1}{a}$. These algorithms Π_{sqrt}, Π_{inv} -from algorithms 13 and 12 respectively- are both defined as a succession of additions and multiplications. They can be modeled into an arithmetic circuit which could be evaluated using existing secure multi-party computation protocols

for additions and multiplications like Π_{add} and Π_{mul}. Finally, note that division is multiplication by an inverse. Consequently, the evaluation of the arithmetic circuit satisfies the same level of security as these two protocols.

5.2.2 Conditioned Additions

The truth-finding algorithms we use require conditioned additions, it is an operation where elements are added to a sum if they satisfy a condition. We can find such operations in lines 9 and 10 of algorithm 7, and in lines 9 and 10 of algorithm 8.

Given two vectors of same size $t = (t^1, \ldots, t^r) \in \mathbb{R}^r$, $z = (z^1, \ldots, z^r) \in \{-1, 0, 1\}^r$, and an element $\kappa \in \{-1, 0, 1\}$, we define the following operation:

$$S := \sum_{i:z^i=\kappa} t^i. \tag{5.1}$$

In other words, the ith element of t, t^i, is added to the sum only if the ith element of z, z^i, is equal to κ. The difficulty is that even though κ is public, z^i is private. To achieve this in secure multi-party computation we start by defining the following function, for $i \in \{1, \ldots, r\}$:

$$\mathcal{E}(z^i, \kappa) = \begin{cases} 1 \text{ if } z^i = \kappa \\ 0 \text{ if not.} \end{cases} \tag{5.2}$$

A naive way to compute the sum S is as follows:

$$S = \sum_i \mathcal{E}(z^i, \kappa) \cdot t^i. \tag{5.3}$$

This way to compute S requires an equality test which is costly in secure multi-party computation. To this end, we propose an alternative that makes good use of the fact that $z^i, \kappa \in \{-1, 0, 1\}$. The goal is to express the function $\mathcal{E}$ as a polynomial so that it can be computed using the smallest number of additions and multiplications possible. We define and use the following expressions of $\mathcal{E}(z^i, \kappa)$.

If $\kappa = -1$, we compute S as follows:

$$S = \sum_i \frac{1}{2}((z^i)^2 - z^i) \cdot t^i. \tag{5.4}$$

We have:

$$\frac{1}{2}((z^i)^2 - z^i) = \begin{cases} 1 & \text{if } z^i = -1 \\ 0 & \text{if } z^i = 0 \\ 0 & \text{if } z^i = 1, \end{cases} \tag{5.5}$$

hence by multiplying $\frac{1}{2}((z^i)^2 - z^i)$ by t^i, the only elements considered in the sum are the ones such that $z^i = -1$. The function $\frac{1}{2}((z^i)^2 - z^i)$ is equal to $\mathcal{E}(z^i, -1)$.
If $\kappa = 0$ we similarly compute S as:

$$S = \sum_i (1 - (z^i)^2) \cdot t^i. \tag{5.6}$$

It is also straightforward that the function $1-(z^i)^2$ is equal to $\mathcal{E}(z^i, 0)$ because it outputs 1 if $z^i = 0$ and 0 elsewise. If $\kappa = 1$, in the same way, S is computed as:

$$S = \sum_i \frac{1}{2}((z^i)^2 + z^i) \cdot t^i, \tag{5.7}$$

where $\frac{1}{2}((z^i)^2 + z^i) = \mathcal{E}(z^i, 1)$.

Algorithm 21 Pseudo-equality protocol $\Pi_{\mathcal{E}}$

Private input: A secret-shared element $[z]$,
a public element $\kappa \in \{-1, 0, 1\}$
Output: Shares of $[s]$

1: **if** $\kappa = -1$ **then**
2: $\quad [s] \leftarrow \left[\frac{1}{2}[z]^2 - [z]\right]$
3: **end if**
4: **if** $\kappa = 0$ **then**
5: $\quad [s] \leftarrow \left[1 - [z]^2\right]$
6: **end if**
7: **if** $\kappa = 1$ **then**
8: $\quad [s] \leftarrow \left[\frac{1}{2}[z]^2 + [z]\right]$
9: **end if**

Lemma 23. *The protocol $\Pi_{\mathcal{E}}$ from algorithm 21 returns shares of $s = \mathcal{E}(z, \kappa)$ using the three defined degree-2 polynomials. $\Pi_{\mathcal{E}}$ does not reveal information about the other's player's share.*

Proof. The three conditioned sums defined in this section do not need comparisons and they are expressed using only additions and multiplications, so their security level is the same as Π_{add} and Π_{mul}. □

5.2.3 3-Estimates and Cosine with Secure Multi-party Computation

Our secure multi-party computation implementation of 3-Estimates is given in algorithm 22.

Theorem 24. *Algorithm 22 ensures that the client learns the truth value and the difficulty of each query as well as the trust factor as given by 3-Estimates with passive security as per definition 3.*

Proof. The answers are additively secret-shared on the servers at the beginning, giving the servers no information about the sources' answers at this point. Then the entire computation takes place in a secret-shared manner by evaluating an arithmetic circuit with secure addition and multiplication protocols, making the rest of the computation secure in the sense that the servers learn no information about the secrets. □

Our secure multi-party computation implementation of Cosine is given in algorithm 23.

Theorem 25. *Algorithm 23 ensures that the client learns the truth value of each query as well as the trust factor as given by Cosine with passive security as per definition 3.*

5.2.4 Efficient Alternatives for Secure Truth-finding Algorithms

The protocols presented in this chapter allow us to implement the truth-finding algorithms with secure multi-party computation. In this section, we propose changes to Cosine and 3-Estimates to reduce communication costs, at the cost of a possibly higher number of errors. We also illustrate in this section the secure multi-party computation version of 3-Estimates, along with a sketch of a security proof.

Algorithm 22 3-Estimates algorithm with secure multi-party computation

Private input: The sources hold the answers $(v^{ij})_{i=1\ldots r}^{j=1\ldots d}$

Public parameter: A number of iterations k

Outputs: The client receives y, θ, δ

$\left[v^{ij}\right] \leftarrow \Pi_{\text{share}}(v^{ij})$ ▷ Each source secret-shares each answer $(v^{ij})_{i=1\ldots r}^{j=1\ldots d}$ on the two servers

for $i = 1 \ldots r$ **do** ▷ Initialization of the untrustworthiness of each source as in algorithm 8

  $\theta^i \leftarrow 0.4$

end for

for $j = 1 \ldots d$ **do** ▷ Initialization of the difficulty of each query as in algorithm 8

  $\delta^j \leftarrow 0.1$

end for

for $j = 1 \ldots d, i = 1 \ldots r$ **do** ▷ Compute equality tests for the conditioned sums

  $\left[\sigma^{ij}\right] \leftarrow \Pi_{\mathcal{E}}(\left[v^{ij}\right], 1)$

  $\left[\tau^{ij}\right] \leftarrow \Pi_{\mathcal{E}}(\left[v^{ij}\right], -1)$

end for

for k iterations **do**

  for $j = 1 \ldots d$ **do** ▷ Update the truth value of each query

    $[posViews] \leftarrow \left[\sum_i \left[\sigma^{ij}\right] \cdot (1 - \theta^i \delta^j)\right]$

    $[negViews] \leftarrow \left[\sum_i \left[\tau^{ij}\right] \cdot (\theta^i \delta^j)\right]$

    $[nbViews] \leftarrow \left[\sum_i \left[\sigma^{ij}\right] + \left[\tau^{ij}\right]\right]$

    $\left[y^j\right] \leftarrow [(posViews + negViews) \cdot \Pi_{\text{inv}}([nbViews])]$

  end for

  Normalize $[y]$

  for $j = 1 \ldots d$ **do** ▷ Update the difficulty score of each query

    $[posViews] \leftarrow \left[\sum_i \left[\sigma^{ij}\right] \cdot (1 - \left[y^j\right]) \cdot \Pi_{\text{inv}}(\left[\theta^i\right])\right]$

    $[negViews] \leftarrow \left[\sum_i \left[\tau^{ij}\right] \cdot \left[y^j\right] \cdot \Pi_{\text{inv}}(\left[\theta^i\right])\right]$

    $[nbViews] \leftarrow \left[\sum_i \left[\sigma^{ij}\right] + \left[\tau^{ij}\right]\right]$

    $\left[\delta^j\right] \leftarrow [(posViews + negViews) \cdot \Pi_{\text{inv}}([nbViews])]$

  end for

  Normalize $[\delta]$

  for $i = 1 \ldots r$ **do** ▷ Update the untrustworthiness of each source

    $[posFacts] \leftarrow \left[\sum_j \left[\sigma^{ij}\right] \cdot (1 - \left[y^j\right]) \cdot \Pi_{\text{inv}}(\left[\delta^j\right])\right]$

    $[negFacts] \leftarrow \left[\sum_j \left[\tau^{ij}\right] \cdot \left[y^j\right] \cdot \Pi_{\text{inv}}(\left[\delta^j\right])\right]$

    $[nbFacts] \leftarrow \left[\sum_j \left[\sigma^{ij}\right] + \left[\tau^{ij}\right]\right]$

    $\left[\theta^i\right] \leftarrow [(posFacts + negFacts) \cdot \Pi_{\text{inv}}([nbFacts])]$

  end for

  Normalize $[\theta]$

end for

$y, \theta, \delta \leftarrow \Pi_{\text{open}}([y], [\theta], [\delta])$

Algorithm 23 Cosine algorithm with secure multi-party computation

Private input: The sources hold the answers $(v^{ij})_{i=1\ldots r}^{j=1\ldots d}$,
the learning rate η
Public parameter: A number of iterations k
Outputs: Client receives y, θ

$\left[v^{ij}\right] \leftarrow \Pi_{\text{share}}(v^{ij})$ ▷ Each source secret-shares each answer $(v^{ij})_{i=1\ldots r}^{j=1\ldots d}$ on the two servers
for $j = 1 \ldots d, i = 1 \ldots r$ **do** ▷ Compute equality tests for the conditioned sums
  $\left[\sigma^{ij}\right] \leftarrow \Pi_{\mathcal{E}}(\left[v^{ij}\right], 1)$
  $\left[\tau^{ij}\right] \leftarrow \Pi_{\mathcal{E}}(\left[v^{ij}\right], -1)$
  $\left[\rho^{ij}\right] \leftarrow \left[\sigma^{ij}\right] + \left[\tau^{ij}\right]$
end for
for $i = 1 \ldots r$ **do** ▷ Initialization of the untrustworthiness of each source
  $\left[\theta^i\right] \leftarrow \left[\left(\sum_j \left[\sigma^{ij}\right] \left[v^{ij}\right] + \sum_j \left[\tau^{ij}\right] \left[v^{ij}\right]\right) \cdot \Pi_{\text{inv}}\left(\left[\sum_j \rho^{ij} \left[v^{ij}\right]\right]\right)\right]$
end for
for $j = 1 \ldots d$ **do** ▷ Initialization of the truth value of each query
  $y^j \leftarrow 1$
end for
for k iterations **do**
  for $i = 1 \ldots r$ **do** ▷ Update the untrustworthiness of each source
    $[posFacts] \leftarrow \left[\sum_j \left[\sigma^{ij}\right] \cdot \left[y^j\right]\right]$
    $[negFacts] \leftarrow \left[\sum_j \left[\tau^{ij}\right] \cdot \left[y^j\right]\right]$
    $[norm] \leftarrow \Pi_{\text{sqrt}}\left(\left[\sum_j \rho^{ij} \left[v^{ij}\right] \cdot \sum_j \rho^{ij} (\left[y^j\right])^2\right]\right)$
    $\left[\theta^i\right] \leftarrow \left[(1-\eta)\theta^i + \eta(posFacts - negFacts) \cdot \Pi_{\text{inv}}([norm])\right]$
  end for
  Normalize $[\theta]$
  for $j = 1 \ldots d$ **do** ▷ Update the truth value of each query
    $[posViews] \leftarrow \left[\sum_i \left[\sigma^{ij}\right] \cdot (\theta^i)^3\right]$
    $[negViews] \leftarrow \left[\sum_i \left[\tau^{ij}\right] \cdot (\theta^i)^3\right]$
    $[norm] \leftarrow \left[\sum_i \rho^{ij} \cdot (\theta^i)^3\right]$
    $\left[y^j\right] \leftarrow [(posViews - negViews) \cdot \Pi_{\text{inv}}([norm])]$
  end for
  Normalize $[y]$
end for
$y, \theta \leftarrow \Pi_{\text{open}}([y], [\theta])$

5.2.4.1 Efficient Normalization in 3-Estimates

In algorithm 3-Estimates, the truth value, trust factor, and difficulty score need to be normalized at each step. This could be done using a secure comparison protocol to securely compute the minimum and the maximum of each value, and then normalize them as it is done in [Gal+10]. Secure comparisons however are very costly in secure multi-party computation. To reduce the amount of communication we replace the normalization based on finding the maximum and minimum by a pre-computed linear transformation which forces the values to stay between 0 and 1. Concretely we apply the function $h(x) = 0.5x + 0.25$ to all the values after each update. We choose the function h to be linear so that its execution in secure multi-party computation is free. Furthermore, the specific coefficients were chosen as such as we noticed that on the specific dataset that we use, the values we normalize never go out of the interval $[-0.25, 1.5]$. We evaluate the impact of this change in the experiments. The chosen function, h, is not perfect. Indeed, if we have information about the distribution of the parameters, we can pre-compute a linear normalization for every iteration. Using any public pre-computed or pre-defined normalizing function improves the efficiency of the algorithm because it would translate to using multiplication and addition by public constants, which is communication-free.

5.2.4.2 Efficient Alternative for Cosine

In Cosine, the truth value and trust factor can be negative, and protocol Π_{inv} can only be applied to positive numbers. Consequently, every time there is a division by an element x, the inverse protocol is applied to $|x|$ and then the result is multiplied by the sign of x [Kno+21]. Computing the sign of x requires computing a secure comparison, which is communication-costly. With the aim to reduce the number of communications, we propose inverting x^2 and multiplying by x. This technique should give the same result with fewer communications. However, in the Cosine algorithm, the denominators are a linear combination of $(\theta^i)^3$ – trust factors of sources to the cube – and since the trust factor is between -1 and 1, $(\theta^i)^3$ could be very small, and squaring it for the sake of a faster inverse makes it even smaller. To avoid any precision issues, we implement a version of the algorithm where we replace $(\theta^i)^3$ by θ^i which will have an impact on the truth value. This impact, however, does not affect the sign of the truth value, it only affects its amplitude, leaving the rounding (i.e. the final label) unchanged. We evaluate the impact of this change in the experiments. Additionally, replacing $(\theta^i)^3$ by θ^i saves multiplications.

5.3 Experimental Results

We evaluate our protocols on two computing servers. We suppose that the sources have already answered and secret-shared their answers. We use the ring $\mathbb{Z}_{2^{60}}$ with 20 bits of fixed precision, that means corresponds to decimals in $[-2^{39}, 2^{39})$ with a step of 2^{-20}. The two servers communicate via a local socket network; for the sake of the experiment, these communications are not encrypted nor authenticated.

5.3.1 3-Estimates on Hubdub Dataset

We implement our solution using the dataset Hubdub from [Gal+10].[3] This dataset is constructed from 457 questions from a Web site where users had to bet on future events. As the questions

[3] All datasets used, as well as the source code of our implementation, is available on `https://github.com/angelos25/tf-mpc.git`

had multiple answers, they have been increased to 830 questions to obtain binary questions with answers $-1, 0$ or 1. The client sends the 830 queries to be classified by each source, and after the classification, the sources secret-share them on two servers to evaluate using secure multi-party computation the 3-Estimates truth-finding algorithm. At the end of the evaluation, the results are reconstructed by the client. The results include the truth value for each query (the label), a difficulty score for each query, and a trustworthiness factor for each of the 471 sources. In figure 5.1 we show the difference between the predictions from the base model and the predictions from the secure multi-party computation evaluation. The base model corresponds to the 3-Estimates algorithm implemented without secure multi-party computation on the plain data.

The secure multi-party computation evaluation contains errors compared to the base model, and these errors are mostly below 10^{-4}. To evaluate the impact of the errors induced by secure multi-party computation, we look at label prediction. The secure multi-party computation method labels all the questions exactly the same way as the baseline method, so both methods made the same number of errors, i.e. 269 (as shown in [Gal+10], this is less than majority voting and some other methods). On average, the execution of each of the k iterations took $52.85s$ wall-clock time or $39.58s$ CPU time. The secure multi-party computation model is 2 000 times slower than the base model, this is due to the high number of comparisons that should be made to normalize the three factors.

If we use the pre-computed linear function h presented in Sec. 5.2.4 the outputs will be very different of course because of the reasons above, but the wall-clock time of each iteration is reduced to $0.58s$ and the CPU time to $0.48s$ making it almost 100 times faster. This normalization alternative increases the number of queries labeled differently by the secure multi-party computation to 5, however, it yields 266 errors in total. For this specific dataset, the pre-computed normalization used happens to give better results than the original baseline.

5.3.2 Cosine on MNIST

We also implement our solution using the MNIST dataset [Den12] (an image classification dataset where the task is to recognize digits between 0 and 9 in the image), this time with the Cosine algorithm. We consider 15 sources, each training a logistic regression model for MNIST on a subset of the considered dataset. The client chooses 120 binary queries to be answered by each source. To apply the secure multi-party computation solution, the sources secret-share the answers on two servers and evaluate using secure multi-party computation the Cosine truth-finding algorithm. At the end of the evaluation, the results are reconstructed by the client. The results include the truth value for each query (the binary label) and a trustworthiness factor for each of the 15 sources. To evaluate secure multi-party computation's impact, we compare the results obtained to a base model. The base model corresponds to the Cosine algorithm implemented without secure multi-party computation on the same answers. Figure 5.2 shows the difference between predictions from the base model and from the secure multi-party computation evaluation.

The secure multi-party computation evaluation contains errors compared to the base model, mostly below 10^{-3}. To evaluate the impact of the errors induced by secure multi-party computation, we look at label prediction. The secure multi-party computation method labels all the questions exactly the same way as the baseline method, so both methods made the same number of errors which is 12. On average, the execution of each iteration took $0.47s$ wall-clock time or $0.36s$ CPU time. The secure multi-party computation model can be up to 4 000 times slower than the base model.

If we apply the modifications for Cosine presented in Sec. 5.2.4 the outputs will be very different of course because of the reasons above and wall-clock time of each iteration is barely reduced to $0.44s$

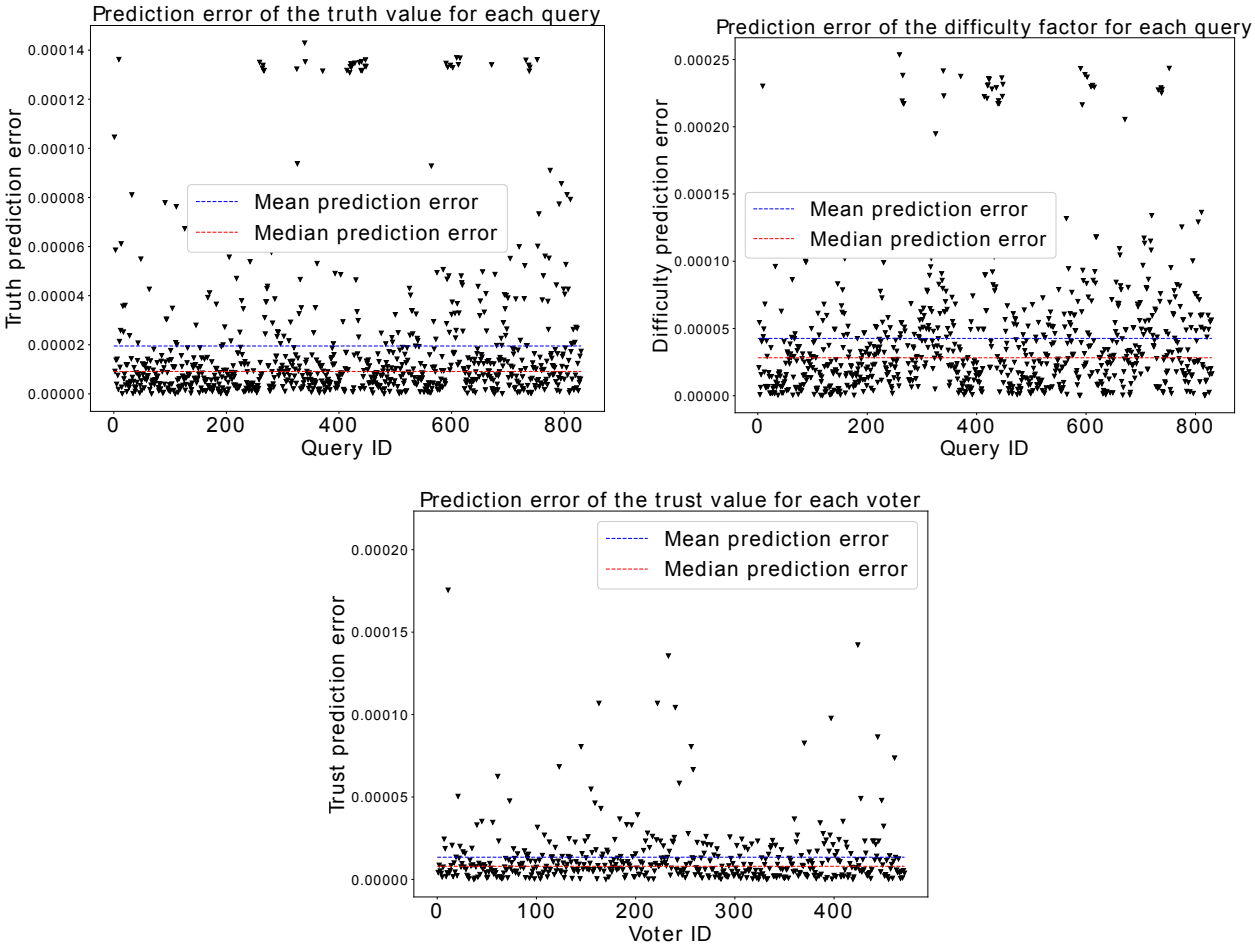

Figure 5.1: Dataset: the hubdub dataset has 830 queries and 457 sources – referred to as voters.
Query ID: the number of the query j on the x-axis.
Voter ID: the number of the source i on the x-axis.
Prediction error: the y-axis indicates the difference in absolute value between the outputs y^j, δ^j, θ^i of 3-Estimates from algorithm 22 that uses secure multi-party computation and the outputs of the algorithm 8 executed in clear on the same inputs v^{ij}. The elements y^j, δ^j, θ^i respectively correspond to the truth value of query j, the difficulty factor of query j, and the trust value of source i.
The blue and red lines respectively correspond to the mean and median difference of the outputs.

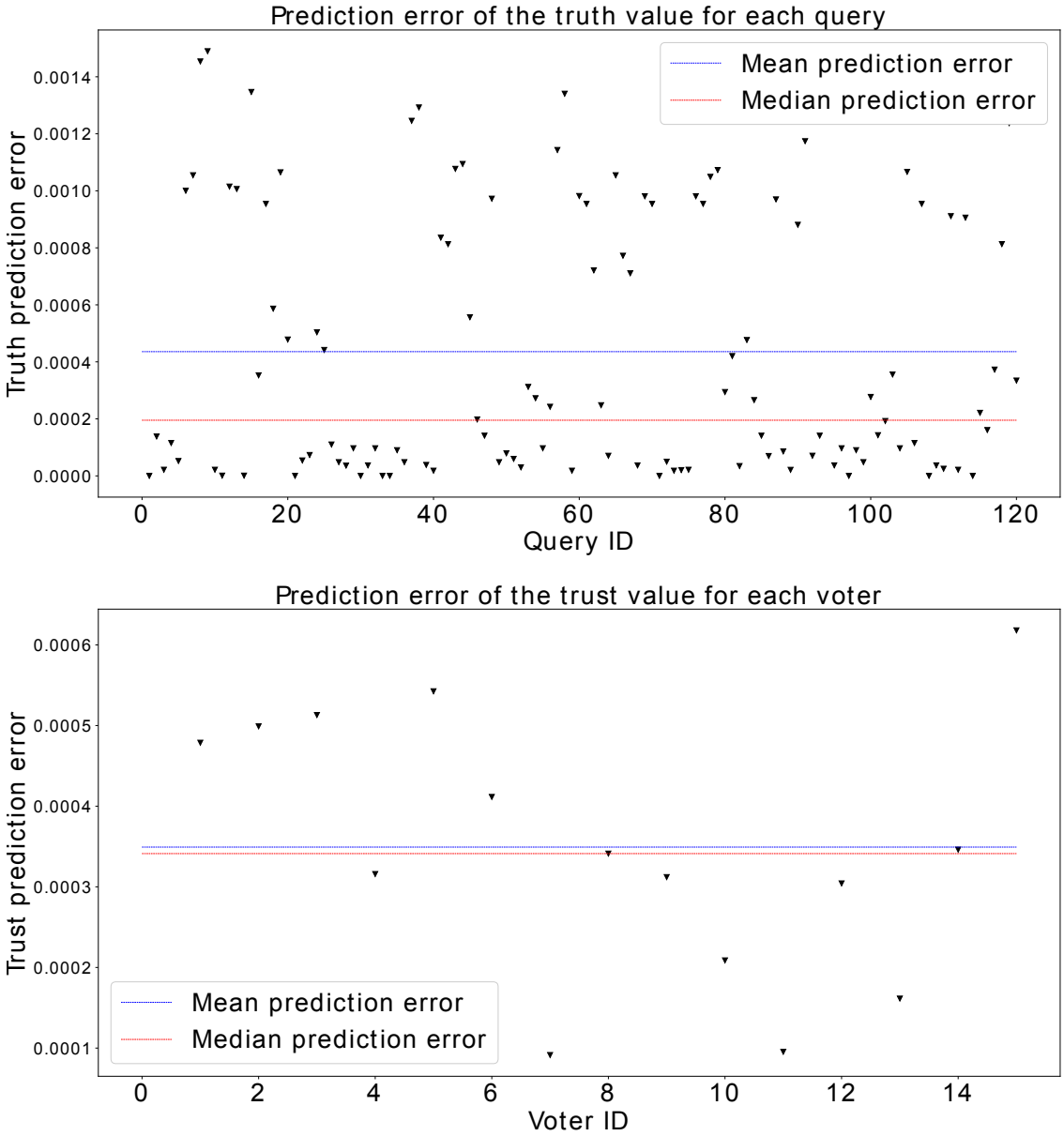

Figure 5.2: Dataset: the MNIST dataset with 120 queries and 15 sources – referred to as voters.
Query ID: the number of the query j on the x-axis.
Voter ID: the number of the source i on the x-axis.
Prediction error: the y-axis indicates the difference in absolute value between the outputs y^j, θ^i of Cosine from algorithm 23 that uses secure multi-party computation and the outputs of the algorithm 7 executed in clear on the same inputs v^{ij}. The elements y^j, θ^i respectively correspond to the truth value of query j, and the trust value of source i.
The blue and red lines respectively correspond to the mean and median difference of the outputs.

and the CPU time to $0.33s$. This alternative increases the number of queries labeled differently by the secure multi-party computation version to 2, however, the number of errors is the same: 12.

5.4 Further Discussion and Conclusion

In this chapter, we devised, presented, and evaluated the performance of secure multi-party computation protocols for truth-finding algorithms corroborating information from disagreeing views while preserving the confidentiality of the data in the sources. This solution is very helpful to complete missing, uncertain, or rare data that is confidential or sensitive, such as financial and medical data (or scientific data in general). The secure multi-party computation protocols we have proposed are versatile and can be used to securely implement other algorithms.

In the specific case where the client's queries correspond to datapoint that needs to be labeled by the sources, and where the sources use a pre-trained model to answer the query, the solution proposed can be further improved by using secure multi-party computation to protect the client's data points and by using differential privacy techniques [DR14] to train the sources' models and protect their privacy. Several works have demonstrated the possibility to combine secure multi-party computation and differential privacy [Pen+22; SKB22]. Indeed this would help further protect the models from inversion attacks. Another application of the model we propose would be combing secure multi-party computation with regular voting and distributed noise generation techniques [Thi+19] to build a version of PATE (private aggregation of teacher ensembles) [Pap+17; Pap+18] that keeps the teacher's data private. In addition, using a truth-finding algorithm like 3-Estimates instead of regular voting for PATE might yield better labeling of incomplete data. A research direction would be to evaluate the privacy, security, efficiency, and accuracy of different combinations of tools like secure multi-party computation, differential privacy, and truth-finding algorithms.

Chapter 6

Secret-sharing-based Primitives for Machine Learning and Data Mining

Since secure multi-party computation protocols are implemented in a finite ring or field, whereas machine learning and data mining algorithms are evaluated on real numbers, the real numbers are approximated with a fixed-point precision and then mapped to a finite ring as it is explained in section 3.1.1. The map between the real subset and the finite ring is not a ring homomorphism, meaning that the multiplication of the images of two elements does not correspond to the image of the multiplication of these elements. This results in not having a trivial algorithm to go back and forth between the two structures. To this end, we present a study on the impact i.e. the error resulting from using pseudo-additions and pseudo-multiplications with fixed precision.

Machine learning and data mining applications use comparison. In plaintext, comparisons are easy to compute, it suffices to check the most significant bit. In secret-sharing-based secure multi-party computation checking the most significant bit of each player does not achieve this. Hence the need for comparison protocols, and there are many such protocols. More information about comparisons is given in section 3.2. An existing and well-performing comparison protocol is Rabbit from [Mak+21]. We give a new algorithm for a fixed-point-based numerical approximation of the sign function. The algorithm takes a secret shared element corresponding to a real number in the interval $[-1, 1]$ and returns the positivity value with a certain error. This comparison can be applied in scenarios where the real number is in this interval for example when generating a Bernoulli random variable given a uniformly random element. We also compare the existing algorithms that implement common activation functions in secure multi-party computation. We also compare algorithms for activation functions based on existing tools that we mix and match in order to evaluate the performance of each combination. We evaluate the performance of each function by measuring its time of execution and by using it to train machine learning algorithms.

6.1 Truncation Errors

In the background section, we defined the map ϕ in equation 3.2, but this definition leaves question marks. We recall that we consider that the real elements are in a bounded interval $[-2^{l-p-1}, 2^{l-p-1})$ of $\mathbb{R}$ which we approximate so they can belong to the real subset

$$D_{p,l} = \{k \cdot 2^{-p}; k \in [-2^{l-1}, 2^{l-1})\},$$

and the map ϕ:

$$\begin{array}{rccc} \phi: & D_{p,l} & \xrightarrow{\phi} & \mathbb{Z}_{2^l} \\ & x & \to & \bar{x} = \phi(x) = 2^p x \mod 2^l. \end{array} \tag{6.1}$$

The problem is that even if a product is in the range delimiting $D_{p,l}$, it may not be in $D_{p,l}$ due to precision problems.

Example 26. *With $p = 1$ and $l = 2$, $D_{1,2} = \{-1, \frac{-1}{2}, 0, \frac{1}{2}\}$. If we compute:*

$$\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4} \notin D_{1,2}.$$

The purpose of the study on truncation errors is to formalize the error that would occur with secure multi-party computation protocols.

In this section, a bar-less element x represents a real element, and $\bar{x}$ represents a ring element.

6.1.1 Redefining Addition and Multiplication

Definition 27 (Truncation)**.** *Let $k \in \mathbb{R}_+, q \in \mathbb{N}, p \in \mathbb{Z}$ and $r \in \mathbb{R}, 0 \leq r < 2^p$ such that $k = q \cdot 2^p + r$. We denote by:*

$$\lfloor k \rfloor_p = q. \tag{6.2}$$

For $k \in \mathbb{R}_-$, $-k \in \mathbb{R}_+$ and we write $-k = q \cdot 2^p + r$, and we have:

$$\lfloor k \rfloor_p = -q. \tag{6.3}$$

Property 28. *For $k \in \mathbb{R}$ we have:*

$$|\lfloor k \rfloor_p - k \cdot 2^{-p}| < 1.$$

Proof. Let $q \in \mathbb{Z}$ and $r \in \mathbb{N}$ with $r < 2^p$ such that; for $k > 0$ we have $k = q \cdot 2^p + r$ and for $k < 0$ we have $-k = q \cdot 2^p + r$, we have:

$$\begin{aligned} |\lfloor k \rfloor_p - k \cdot 2^{-p}| &= \begin{cases} |q - (q + r \cdot 2^{-p})| & \text{if } k > 0 \\ |-q + (q + r \cdot 2^{-p})| & \text{if not} \end{cases} \\ &= |r \cdot 2^{-p}| \\ &< 1. \end{aligned}$$

□

We denote by $D_p = \{k \cdot 2^{-p}; k \in \mathbb{Z}\}$. For $x, y \in D_p$, there exist $k_x, k_y \in \mathbb{Z}; x = k_x \cdot 2^{-p}, y = k_y \cdot 2^{-p}$. We define the two following operations:

$$+_D : (x, y) \to x +_D y = x + y$$

$$* : (x, y) \to x * y = \lfloor k_x k_y \rfloor_p 2^{-p}.$$

Property 29. *The set D_p is closed under the binary operations $+_D$ and $*$.*

Proof. We have

$$x +_D y = x + y = k \cdot 2^{-p} + k_y \cdot 2^{-p} = (k_x + k_y) 2^{-p} \in D_p.$$

It is obvious that $x * y \in D_p$. □

Remark 30. *The set D_p equipped with the two binary operations: $(D_p, +, *)$ is not a ring in general, because $*$ is nor associative nor distributive. The operation $*$ is considered a pseudo-multiplication.*

Example 31. *Consider the set $D_2 = \{\frac{k}{4}; k \in \mathbb{Z}\}$. Let $0.25, 1.25, 1.75 \in D_2$, we have:*

$$\begin{aligned}(0.25 * 1.25) * 1.75 &= \frac{\lfloor 1 \cdot 5 \rfloor_2}{4} * 1.75 \\ &= \frac{1}{4} * 1.75 \\ &= \frac{\lfloor 1 \cdot 7 \rfloor_2}{4} \\ &= \frac{1}{4}.\end{aligned}$$

Whereas,

$$\begin{aligned}0.25 * (1.25 * 1.75) &= 0.25 * \frac{\lfloor 5 \cdot 7 \rfloor_2}{4} \\ &= 0.25 * \frac{8}{4} \\ &= \frac{\lfloor 1 \cdot 8 \rfloor_2}{4} \\ &= \frac{2}{4}.\end{aligned}$$

Therefore $*$ is not associative and $(D_2, +, *)$ is not a ring

6.1.2 Error Measuring

Let $p > 0$, we consider the following map:

$$\begin{array}{rccc} \psi: & \mathbb{R} & \to & D_p \\ & x & \to & \psi(x) = \lfloor x \rfloor_{-p} \cdot 2^{-p}. \end{array} \tag{6.4}$$

Lemma 32. *By construction, the map ψ is well-defined.*

The objective is to compute a bound for e_1, e_2, and e_3, respectively corresponding to the error of approximating a real element with its image by ψ, of approximating the real sum by the sum of the images, and the real multiplication by the binary operation $*$. Mathematically, the errors are defined as the following, for $x, y \in \mathbb{R}$ we have:

$$\begin{aligned} |\psi(x) - x| &= e_1 \\ |(\psi(x) + \psi(y)) - (x + y)| &= e_2 \\ |(\psi(x) * \psi(y)) - (x \cdot y)| &= e_3. \end{aligned} \tag{6.5}$$

Lemma 33 (Error of approximating real elements)**.** *Real elements are approximated by ψ with a bounded error of 2^{-p}, hence $e_1 < 2^{-p}$.*

Proof. For $x \in \mathbb{R}$,

$$\begin{aligned} e_1 &= |\psi(x) - x| \\ &= |\lfloor x \rfloor_{-p} \cdot 2^{-p} - x 2^p 2^{-p}| \\ &= 2^{-p} \cdot |\lfloor x \rfloor_{-p} - x 2^p| \\ &< 2^{-p} \cdot 1. \end{aligned}$$

□

Lemma 34 (Addition error)**.** *Adding the images of real numbers by ψ instead of adding the real numbers incurs an error of $e_2 \leq 2^{-p+1}$.*

Proof. For $x, y \in \mathbb{R}$, let us write:

$$\psi(x) = x + \epsilon_x \text{ and } \psi(y) = y + \epsilon_y,$$

with $|\epsilon_x| \leq e_1$ and $|\epsilon_y| \leq e_1$. Then,

$$\begin{aligned} e_2 &= |(\psi(x) + \psi(y)) - (x + y)| \\ &\leq |\epsilon_x| + |\epsilon_y| \\ &\leq 2e_1 \\ &< 2^{-p+1}. \end{aligned}$$

□

Lemma 35 (Multiplication error)**.** *For $x, y \in \mathbb{R}$, the pseudo-multiplication error e_3 is bounded by:*

$$e_3 < 2^{-p}(1 + 2^{-p} + |\psi(x)| + |\psi(y)|). \tag{6.6}$$

Proof. For $x, y \in \mathbb{R}$, let us write:

$$\psi(x) = x + \epsilon_x \text{ and } \psi(y) = y + \epsilon_y,$$

with $|\epsilon_x| \leq e_1$ and $|\epsilon_y| \leq e_1$. In addition, $\psi(x), \psi(y) \in D_p$ so there exist $k_x, k_y; \psi(x) = k_x 2^{-p}$ and $\psi(y) = k_y 2^{-p}$. We compute:

$$\begin{aligned} |\psi(x) * \psi(y) - \psi(x) \cdot \psi(y)| &= |\lfloor k_x k_y \rfloor_p 2^{-p} - k_x k_y 2^{-p} 2^{-p}| \\ &= 2^{-p} |\lfloor k_x k_y \rfloor_p - k_x k_y 2^{-p}| \\ &< 2^{-p} |1|. \end{aligned}$$

Then, we can deduce that:

$$\begin{aligned} e_3 &= |(\psi(x) * \psi(y)) - (x \cdot y)| \\ &\leq |\psi(x) * \psi(y) - ((\psi(x) - \epsilon_x) \cdot (\psi(y) - \epsilon_y)| \\ &\leq |\psi(x) * \psi(y) - \psi(x)\psi(y) - \epsilon_x \psi(y) - \epsilon_y \psi(x) + \epsilon_x \epsilon_y| \\ &\leq |\psi(x) * \psi(y) - \psi(x)\psi(y)| + |\epsilon_x \epsilon_y| + |\epsilon_x \psi(y)| + |\epsilon_y \psi(x)| \\ &< 2^{-p}(1 + 2^{-p} + |\psi(x)| + |\psi(y)|). \end{aligned}$$

□

Remark 36. *Some secure multi-party computation frameworks use floating-point like in [KW15]. However, reference [Cam18] points out that the operations would then come at a high cost as the computation requires secure access to the bits of the binary representation.*

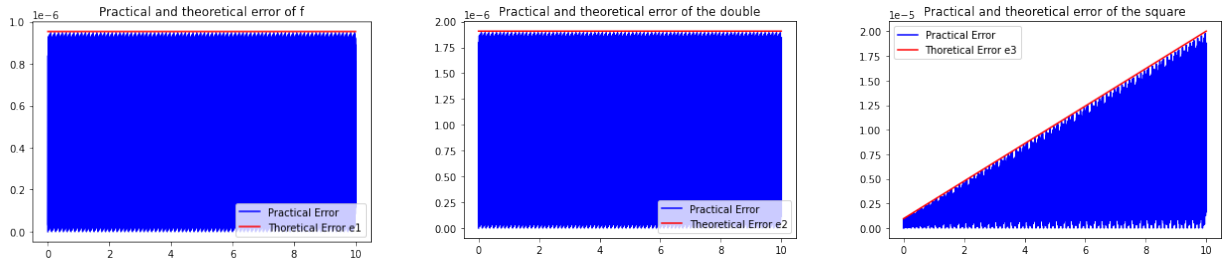

Figure 6.1: Theoretical and practical errors of approximations for three functions. Let $p = 20$, and ψ as in equation 6.4. We would like to compute the error by approximating a real number with its image by ψ, the double, and the square of $x \in \mathbb{R}$ using the set D_p. In the following graphs, we compare on the y-axis the theoretic error bound computed in the previous lemmas, with the practical errors computed for $x \in \{\frac{k}{1000}; k \in [0, 10000]\}$ on the x-axis for three functions, the approximation function, the functions double, and the function square. More precisely, we measure the error of approximating x by $\psi(x)$, i.e. $|\psi(x) - x|$ for $x \in \{\frac{k}{1000}; k \in [0, 10000]\}$ and we denote it as the practical error. We compare this practical error drawn in blue to e_1 - drawn in red - which is the theoretical error.
We also measure the error of approximating $2x$ by $2\psi(x)$, i.e. $|2\psi(x) - 2x|$ for $x \in \{\frac{k}{1000}; k \in [0, 10000]\}$ and we compare this practical error drawn in blue to e_2 - drawn in red - which is the theoretical error computed above.
We do the same for the square function, we measure the error of approximating x^2 by $\psi(x) * \psi(x)$, i.e. $|\psi(x) * \psi(x) - x^2|$ for $x \in \{\frac{k}{1000}; k \in [0, 10000]\}$ and we compare this practical error drawn in blue to e_3 - drawn in red - which is the theoretical error computed above. The graphs in blue look like an area rather than a curve because the points are very close one to another and their value varies between 0 and the error bound. The graphs confirm that the practical error is bounded by the theoretical error that was computed.

6.1.3 Consequences of Approximations on Ring Operations

Given $l \in \mathbb{N}^*$, we consider the real interval $[-2^{l-p-1}, 2^{l-p-1}[\subset \mathbb{R}$, we also consider

$$D_{p,l} = \{k \cdot 2^{-p}; k \in [-2^{l-1}, 2^{l-1}[\},$$

and $\mathbb{Z}_{2^l} = \mathbb{Z}/2^l\mathbb{Z} = [0, 2^l)$. We define the restriction of ψ on the real interval $[-2^{l-p-1}, 2^{l-p-1})$. We will also denote it by ψ:

$$\begin{array}{rccc} \psi: & [-2^{l-p-1}, 2^{l-p-1}) & \to & D_{p,l} \\ & x & \to & \psi(x) = \lfloor x \rfloor_{-p} \cdot 2^{-p}. \end{array} \tag{6.7}$$

The set $D_{p,l}$ is not closed for the operations $+$ and $*$. However, if $x + y, x * y \in D_{p,l}$, then the errors e_1, e_2, e_3 are still the same as the ones computed after equation 6.5.

We use this restriction to find the set $D_{p,l}$ in which we approximate real elements that will be mapped to the finite ring for the purpose of secure multi-party computation. We recall the map ϕ from equation 3.2:

$$\begin{array}{rccc} \phi: & D_{p,l} & \xrightarrow{\phi} & \mathbb{Z}_{2^l} \\ & x & \to & \bar{x} = \phi(x) = 2^p x \mod 2^l. \end{array} \tag{6.8}$$

Lemma 37. *The function ϕ is a one-to-one correspondence and its inverse is:*

$$\begin{array}{rccc} \phi^{-1}: & \mathbb{Z}_{2^l} & \to & D_{p,l} \\ & \bar{x} & \to & x = \begin{cases} 2^{-p}\bar{x} & \text{if } \bar{x} < 2^{l-1} \\ 2^{-p}(\bar{x} - 2^l) & \text{otherwise.} \end{cases} \end{array} \tag{6.9}$$

The proof of lemma 37 is straightforward because of the construction of ϕ.

With ϕ being bijective – a one-to-one correspondence – with a well-defined inverse, players would be able to map their real secret in $[-2^{l-p-1}, 2^{l-p-1})$ to a single ring element using $\phi \circ \psi$. In addition, each ring element, including the value on the last node of an arithmetic circuit in the finite ring will correspond to a single real element, that can be easily computed with the inverse of ϕ. The goal now is to adapt and understand how secure multi-party computation protocols are adapted.

Lemma 38. *For $x, y \in D_{p,l}$, such that $x + y \in D_{p,l}$ we have:*

$$\phi^{-1}(\bar{x} + \bar{y} \mod 2^l) = x + y.$$

Proof. For $x, y \in D_{p,l}$, there exist $k_x, k_y \in [-2^{l-1}, 2^{l-1}[; x = k_x 2^{-p}$.

$$\begin{aligned} x + y \in D_{p,l} &\implies k_x + k_y \in [-2^{l-1}, 2^{l-1}[\\ &\implies k_x + k_y \mod 2^l = \begin{cases} k_x + k_y & \text{if } k_x + k_y \geq 0 \\ 2^l + (k_x + k_y) & \text{otherwise.} \end{cases} \end{aligned}$$

The image of x by ϕ is $\phi(x) = \bar{x} = k_x \mod 2^l$.

$$\begin{aligned} \phi^{-1}(\bar{x} + \bar{y}) &= \phi^{-1}(k_x + k_y \mod 2^l) \\ &= \begin{cases} 2^{-p}(k_x + k_y) & \text{if } k_x + k_y \geq 0 \\ -2^{l-p} + 2^{-p}(2^l + (k_x + k_y)) & \text{otherwise.} \end{cases} \\ &= 2^{-p}(k_x + k_y) \\ &= x + y. \end{aligned}$$

□

Lemma 38 shows that the addition protocol in secure multi-party computation needs not to be modified. This is not the case for the multiplication as seen in the background section 3.1.2, and more precisely in equation 3.5. In the background section, we said that we need to truncate p bits after each multiplication. Here we prove that the image by ψ^{-1} of a multiplication followed by a truncation in the rings is equal to the formally defined pseudo-multiplication $*$ in $D_{p,l}$.

Lemma 39. *For $x, y \in D_{p,l}$, such that there exist $k_x, k_y; x = 2^{-p}k_x, y = 2^{-p}k_y$ such that $k_x k_y \in [-2^{l-1}, 2^{l-1})$ we define:*

$$Tr(\bar{x}, p) = \begin{cases} \lfloor \bar{x} \rfloor_p & \text{if } \bar{x} < 2^{l-1} \\ 2^l - \lfloor 2^l - \bar{x} \rfloor_p & \text{otherwise.} \end{cases} \tag{6.10}$$

and

$$\bar{*} : (\bar{x}, \bar{y}) \to \bar{x} \bar{*} \bar{y} = Tr(\bar{x}\bar{y}, p).$$

We have:

$$\phi^{-1}(\bar{x} \bar{*} \bar{y}) = x * y.$$

Proof.

$$\begin{aligned}
\phi^{-1}(\bar{x} \bar{*} \bar{y}) &= \phi^{-1}(Tr(\bar{x}\bar{y})) \\
&= \begin{cases} \phi^{-1}(\lfloor \bar{x}\bar{y} \rfloor_p) & \text{if } \bar{x}\bar{y} < 2^{l-1} \\ \phi^{-1}(2^l - \lfloor 2^l - \bar{x}\bar{y} \rfloor_p) & \text{otherwise.} \end{cases} \\
&= \begin{cases} 2^{-p} \lfloor \bar{x}\bar{y} \rfloor_p & \text{if } \bar{x}\bar{y} < 2^{l-1} \\ -2^{-p} \lfloor 2^l - \bar{x}\bar{y} \rfloor_p & \text{otherwise.} \end{cases} \\
&= \begin{cases} 2^{-p} \lfloor k_x k_y \rfloor_p & \text{if } \bar{x}\bar{y} < 2^{l-1} (\implies k_x k_y \geq 0) \\ -2^{-p} \lfloor 2^l - (2^l + k_x k_y) \rfloor_p & \text{otherwise.} \end{cases} \\
&= \begin{cases} 2^{-p} \lfloor k_x k_y \rfloor_p & \text{if } k_x k_y \geq 0 \\ -2^{-p} \lfloor -k_x k_y \rfloor_p) & \text{otherwise.} \end{cases} \\
&= 2^{-p} \lfloor k_x k_y \rfloor_p \\
&= x * y.
\end{aligned}$$

□

As mentioned before, there are protocols to perform a truncation in secure multi-party computation. We state the protocol defined in [MZ17] for two parties.

Algorithm 24 Two-party truncation protocol Π_{trunc}

Private input: Players P_0 and P_1 respectively hold the share $\bar{x_0}$ and $\bar{x_1}$ of the secret $[\bar{x}]$
Output: Players P_0 and P_1 respectively hold the share $\bar{e_0}$ and $\bar{e_1}$ of the secret $[\bar{e}]$

1: P_0 locally computes $\bar{e_0} \leftarrow \lfloor \bar{x_0} \rfloor_p$
2: P_1 locally computes $\bar{e_1} \leftarrow 2^l - \lfloor 2^l - \bar{x_1} \rfloor_p$

Lemma 40 ([MZ17])**.** *For $\bar{x} \in \mathbb{Z}_{2^l}$,*

$$Tr(\bar{x}, p) = \begin{cases} \lfloor \bar{x} \rfloor_p & \text{if } \bar{x} < 2^{l-1} \\ 2^l - \lfloor 2^l - \bar{x} \rfloor_p & \text{if not,} \end{cases}$$

The multi-party computation protocol in algorithm 24 return shares of $[Tr(\bar{x}, p) + \epsilon]\,; \epsilon \in \{0, 1, -1\}$ *with a probability* $P(\bar{x})$ *given by:*

$$\begin{array}{rccl} P: & \mathbb{Z}_{2^l} & \to & [0,1] \\ & \bar{x} & \to & P(\bar{x}) = \begin{cases} 2^{-p}\bar{x} & \text{if } \bar{x} < 2^{l-1} \\ 2^{-p}(\bar{x} - 2^l) & \text{otherwise.} \end{cases} \end{array} \tag{6.11}$$

Based on the proof given for lemma 40 in [MZ17], the correctness of the protocol depends on the choice of the shares of $[\bar{x}]$ which can be written with $\bar{r} \in_R [0, 2^l)$ as the following:

$$[\bar{x}] \to (x_0, x_1) = (\bar{x} + \bar{r} \mod 2^l, 2^l - \bar{r}).$$

For the protocol to return the correct output we should have:

$$\begin{cases} \bar{r} < 2^l - \bar{x} & \text{if } \bar{x} < 2^{l-1} \\ \bar{r} > 2^l - \bar{x} & \text{otherwise.} \end{cases} \tag{6.12}$$

This condition is satisfied with probability $P(\bar{x})$.

Example 41. *Given* $l = 60, p = 20$*, and an element* $[\bar{x}]$ *secret-shared in the ring, after the truncation in algorithm 24 if we would like to have an output equal to* $[Tr(\bar{x}, p) + \epsilon]\,; \epsilon \in \{0, 1, -1\}$ *with a probability greater than* $1 - 2^{-10}$*, we need to have:*

$$\begin{cases} \bar{x} < 2^{50} & \text{if } \bar{x} < 2^{59} \\ \bar{x} > 2^{60} - 2^{50} & \text{otherwise.} \end{cases}$$

In other words, the absolute value of the real element should be less than 2^{10}*. Because if we suppose that the real element is* 2^{10}*, it would correspond to the ring element* 2^{30} *which before the truncation is equal to* 2^{50}*.*

The formalization of the approximation errors, the definition of lemmas 32 to 39, and lemma 40 from [MZ17] allow us to state theorem 42 whose results, to the best of our knowledge, are not addressed in previous works - including [MZ17; Esc+20; CS10], and [Cam18], where the author shows that it is sometimes better to postpone some truncations and truncate once every few layers of a neural network to make it more efficient time-wise.

Theorem 42. *Let* Π_{mul} *and* Π_{trunc} *be the multiplication and truncation protocols respectively as defined in algorithms 4 and 24. Let* $p, l \in \mathbb{N}$ *be two integers with* $p < l$*, and* $x, y \in [-2^{l-p-1}, 2^{l-p-1})$ *be two real elements such that the product* $x \cdot y \in [-2^{l-p-1}, 2^{l-p-1})$ *with* $\psi(x), \psi(y) \in D_{p,l}$*. We define* $\bar{x} = \phi(\psi(x))$ *and* $\bar{y} = \phi(\psi(y))$ *both in* $\mathbb{Z}_{2^l}$*. Let* $[\bar{x}]$ *and* $[\bar{y}]$ *be two random sharings of* $\bar{x}$ *and* $\bar{y}$*.*

We define the error of the multiplication $\Pi_{\text{trunc}}(\Pi_{\text{mul}}([\bar{x}], [\bar{y}]))$ *as the following:*

$$e = \max |x \cdot y - \phi^{-1}(\Pi_{\text{open}}(\Pi_{\text{trunc}}(\Pi_{\text{mul}}([\bar{x}], [\bar{y}]))))|.$$

Set P *as defined in equation 6.11, and* e_3 *as computed in lemma 35 equation 6.6.*

With probability $P(\bar{x}\bar{y})$*,* $e = e_3 + 2^{-p}$*.*

Proof. As per lemma 40, the probability of a truncation with Π_{trunc} not having an output of $[Tr(\bar{x}\bar{y}, p) + \epsilon]\,; \epsilon \in \{0, 1, -1\}$ is $P(\bar{x}\bar{y})$. Consequently, with probability $P(\bar{x}\bar{y})$ we have:

$$
\begin{aligned}
e &= |x \cdot y - \phi^{-1}(\Pi_{\text{open}}(\Pi_{\text{trunc}}(\Pi_{\text{mul}}([\bar{x}], [\bar{y}]))))| \\
&= |x \cdot y - \phi^{-1}(\Pi_{\text{open}}(\Pi_{\text{trunc}}([\bar{x}\bar{y}])))| \\
&\leq |x \cdot y - \phi^{-1}(\Pi_{\text{open}}([Tr(\bar{x}\bar{y}, p) + \epsilon]))|, \epsilon \in \{-1, 0, 1\} \text{ by lemma 40} \\
&\leq |x \cdot y - \phi^{-1}(Tr(\bar{x}\bar{y}, p) + \epsilon)| \\
&\leq |x \cdot y - \phi^{-1}(Tr(\bar{x}\bar{y}, p)) + phi^{-1}(\epsilon)| \text{ by lemmas 37 and 38} \\
&\leq |x \cdot y - \psi(x) * \psi(y) + \phi^{-1}(\epsilon)| \text{ by lemma 39} \\
&\leq |x \cdot y - \psi(x) * \psi(y)| + |\phi^{-1}(\epsilon)| \\
&\leq e_3 + 2^{-p} \text{ by lemma 35.}
\end{aligned}
$$

Therefore, $e = e_3 + 2^{-p}$.

□

6.2 MPC Protocols for Machine Learning

6.2.1 Details on the Rabbit Comparison Algorithm

As mentioned in the introduction Rabbit is a recently developed secure comparison protocol presented in [Mak+21] that takes advantage of the commutative property of addition in rings and fields. The goal of this section is to explain the algorithm in detail.

This protocol utilizes doubly secret-shared bits referred to as daBits [RW19], which are divided between $\mathbb{Z}/2^l\mathbb{Z}$ and $\mathbb{Z}/2\mathbb{Z}$, and extended doubly secret-shared ring elements known as edaBits [Esc+20], which involve sharing integers within an arithmetic framework and its bit decomposition within a binary domain. In comparison to earlier secure comparison protocols, Rabbit is more efficient and does not require bounds or statistical parameters.

The "less than bits" algorithm Π_{LTB} presented in algorithm 25 from [Mak+21, figure 3] is the first algorithm used by Rabbit. It compares a ring element written in binary form, with each bit secret-shared in $\mathbb{Z}_2$. Concerning the notation, we recall that the sharing of a bit b that has been secret-shared in $\mathbb{Z}_2$ is represented as $[b]_2$, which is distinct from when it's shared in the ring $\mathbb{Z}/2^l\mathbb{Z}$, which does not include a subscript. This algorithm uses a doubly shared positive integer called EdaBit [Esc+20]; given a random ring element r, the secret $[r]_{2^l}$ is shared in $\mathbb{Z}_{2^l}$, and the bits $[r_0]_2, \ldots, [r_{l-1}]_2$ are shared in $\mathbb{Z}_2$ with $r = \sum_{i=0}^{l-1} r_i 2^i$.

The algorithm Π_{LTC}, a sub-protocol used to compare a secret shared ring element with a public constant referred to as the "less than constant", is presented in algorithm 26 and taken from figure 4 of [Mak+21].

A proof of the correctness of algorithm 26 was given in [Mak+21], in this thesis, we expand it.

Lemma 43. *The comparison algorithm described in 26 outputs the correct result.*

Proof. Set $M = 2^l$, a set of n players needs to obtain shares of $(x < R) \in \{0, 1\}$ shared in $\mathbb{Z}_2$, where x is a secret and R is a public constant such that $0 \leq x < M$ and $0 \leq R < M$, which corresponds to the representation of the finite ring. Let $\alpha, \beta \in \mathbb{Z}$, $0 \leq \alpha < M$, and $0 \leq \beta < M$. The comparison protocol is based on the equation below:

$$
(\alpha + \beta) \mod M = \alpha + \beta - M \cdot ((\alpha + \beta) \mod M < \alpha). \tag{6.13}
$$

We recall that the notation $[x]$ represents the value $x \in \mathbb{Z}_M$ shared in $\mathbb{Z}_M$.

Algorithm 25 Less than bits protocol Π_{LTB} [Mak+21]

Private input: $[x]$ shared bitwise; $[x_0]_2, [x_1]_2, \ldots, [x_{l-1}]_2$, where $x = \sum_{i=0}^{l-1} 2^i x_i$,
and public value R
Output: Boolean value $[c]_2 = [R \leq x]_2$

for $i = 0 \ldots l-1$ **do**
  $[y_i]_2 \leftarrow [x_i]_2 \oplus R_i$ ▷ R_i is the i-th bit of R
end for
for $i = 0 \ldots l-1$ **do**
  $[z_i]_2 \leftarrow \vee_{j=i}^{j=l-1} [y_j]_2$
end for
$[z_l]_2 \leftarrow 0$
for $i = 0 \ldots l-1$ **do**
  $[w_i]_2 \leftarrow [z_i]_2 - [z_{i+1}]_2$
end for
$[c]_2 \leftarrow 1 - \sum_{i=0}^{l-1} R_i \cdot [w_i]_2$

Algorithm 26 Less than constant protocol Π_{LTC} [Mak+21]

Private input: $[x]$ shared in the ring $\mathbb{Z}_{2^l}$,
a public value R,
and an EdaBit $([r], [r_0]_2, \ldots, [r_{l-1}]_2)$; $r = \sum_{i=0}^{l-1} 2^i r_i$
Output: Boolean value $[w]_2 = [x < R]_2$

$[a] \leftarrow [x + r]$
$[b] \leftarrow [a - R]$
Reveal a and b
$[w_1]_2 \leftarrow \Pi_{\text{LTB}}(a, [r_0]_2, \ldots, [r_{l-1}]_2)$
$[w_2]_2 \leftarrow \Pi_{\text{LTB}}(b, [r_0]_2, \ldots, [r_{l-1}]_2)$
$w_3 \leftarrow (b < 2^l - R)$
$[w]_2 \leftarrow 1 - ([w_1]_2 - [w_2]_2 + w_3)$

In [Mak+21] it is written that:

$$B = M - R, a = (x + r) \mod M, c = (x + B) \mod M,$$
$$\text{and } b = (x + r + B) \mod M,$$

where b can be written in two ways:

The first one is: $b = (a + B) \mod M$ which by using the equation 6.13 twice corresponds to:

$$\begin{aligned} b = (a + B) \mod M &= a + B - M \cdot (b < B) \\ &= x + r - M \cdot (a < r) + B - M \cdot (b < B). \end{aligned}$$

The second one is: $b = (c + r) \mod M$ which similarly is written as:

$$\begin{aligned} b = (c + r) \mod M &= c + r - M \cdot (b < r) \\ &= x + B - M \cdot (c < B) + r - M \cdot (b < r). \end{aligned}$$

By equalizing the two forms of b the authors obtain:

$$\begin{aligned} x + r - M \cdot (a < r) + B - M \cdot (b < B) &= x + B - M \cdot (x < B) + r - M \cdot (b < r) \\ (a < r) + (b < B) &= (c < B) + (b < r) \\ (c < B) &= (a < r) - (b < r) + (b < B). \end{aligned}$$

The next step is to prove that $(c < B)$ is equivalent to $1 - (x < R)$:

$$\begin{aligned} c < B = (x + B) \mod M < B &= (x + M - R) \mod M < (M - R) \\ &= (x - R) \mod M < (M - R) \\ &= \begin{cases} (x - R) < (M - R) & \text{if } (x - R) \geq 0 \\ (x + M - R) < (M - R) & \text{if } (x - R) < 0 \end{cases} \\ &= \begin{cases} x < M & \text{if } x \geq R \\ x < 0 & \text{if } x < R \end{cases} \\ &= \begin{cases} 1 & \text{if } x \geq R \\ 0 & \text{if } x < R \end{cases} \text{ because } x \in \mathbb{Z}_M \\ &= 1 - (x < R). \end{aligned}$$

This gives $(x < R) = 1 - ((a < r) - (b < r) + (b < M - R))$. Therefore, by computing $1 - ((a < r) - (b < r) + (b < M - R))$ in algorithm 26 line 7, the algorithm returns shares of $x < R$. □

As mentioned in their paper [Mak+21], in the finite ring $\mathbb{Z}/2^l\mathbb{Z}$ the Less-than-constant algorithm requires $2 + \log_2 l$ rounds which are twice less than the algorithm presented in [Esc+20].

6.2.2 Numerical Approximations

As mentioned in the previous chapters, we use a finite ring to represent the real elements and use numerical approximations to compute numerical functions in secret-sharing-based secure multi-party computation. This allows us to execute numerical approximations of the sign function, sigmoid, and other activation functions in a secure multi-party computation manner. By doing this, we reduce the communication cost of secure multi-party computation significantly. Secret-sharing-based secure multi-party computation is not the only way to compute such functions, it could be done by using garbled circuits [Bal+19; Lam+22].

6.2.2.1 Numerical Approximation of the Sign Function

In this section, we present a way to approximate the sign function on a real interval $I = [a, b]$.

$$\begin{array}{rccl} f: & I = [a,b] & \to & \{0,1\} \\ & x & \to & f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise.} \end{cases} \end{array}$$

Without loss of generalization we consider the interval $I = [-1, 1]$ because, in many of the considered real-world applications, the domain on which we evaluate the sign function is $I = [-1, 1]$.

The idea is to have an algorithm that converges to 0 if the input $x \in [-1, 1]$ is strictly negative, and to 1 if the input is strictly positive. To this end we define the following function:

Theorem 44. *Let g be the function defined as the following:*

$$\begin{array}{rccl} g: & [-1,1] & \to & [-1,1] \\ & x & \to & g(x) = -\frac{1}{2}x^3 + \frac{3}{2}x. \end{array}$$

Given $x_0 \in [-1, 1]$, let (U_n) be the sequence the defined as the following:

$$(U_n) : \begin{cases} U_0 = x_0 \\ U_{n+1} = g(U_n). \end{cases}$$

The sequence (U_n) quadratically converges to 1 if $x_0 > 0$, to -1 if $x_0 < 0$, and to 0 if $x_0 = 0$.

Proof. The function g has three fixed point $(-1, 0, 1)$.

The derivative of g is $g'(x) = \frac{-3}{2}x^2 + \frac{3}{2}$, thus $g'(0) = \frac{3}{2} > 1$, $g'(1) = 0$ and $g'(-1) = 0$. It follows that (U_n) converges to 1 if x_0 is positive and to -1 otherwise.

Now we prove that the order of convergence is quadratic, we start by proving it for the fixed point 1, and for the other point (-1) the proof is similarly done. We should prove the following:

$$\exists \mu \in \mathbb{R}, \forall k \in \mathbb{N}, |U_{k+1} - 1| \leq \mu |U_k - 1|^2.$$

We have:

$$\frac{|U_{k+1} - 1|}{|U_k - 1|^2} = \frac{|\frac{-1}{2}U_k^3 + \frac{3}{2}U_k - 1|}{|U_k - 1|^2}.$$

And since the proof is for the fixed point 1, the initial point $U_0 > 0$ (is positive) and so are all the terms $0 < U_k < 1$ of the sequence because it is strictly increasing. Therefore, we should find μ such that:

$$\frac{\frac{1}{2}U_k^3 - \frac{3}{2}U_k + 1}{(U_k - 1)^2} \leq \mu.$$

Finally,

$$\begin{aligned} \frac{\frac{1}{2}U_k^3 - \frac{3}{2}U_k + 1}{(U_k - 1)^2} &\leq \frac{(U_k - 1)(\frac{1}{2}U_k^2 + \frac{1}{2}U_k - 1)}{(U_k - 1)^2} \\ &\leq \frac{(U_k - 1)^2(\frac{1}{2}U_k + 1)}{(U_k - 1)^2} \\ &\leq \frac{1}{2}U_k + 1 \\ &\leq \frac{3}{2} \text{ because } 0 < U_k < 1, \end{aligned}$$

and we set $\mu = \frac{3}{2}$. □

Algorithm 27 Numerical approximation of the sign function protocol Π_{sign}

Private input: The players hold their shares of the secret $[x]$

Public parameter: A number of iterations k

Output: The players hold shares of $[y]$

1: $[y] \leftarrow \left[\frac{-1}{2}[x]^3 + \frac{3}{2}[x]\right]$
2: **for** k iterations **do**
3: $\quad [y] \leftarrow \frac{-1}{2}[y]^3 + \frac{3}{2}[y]$
4: **end for**
5: $[y] \leftarrow \left[\frac{1}{2}(\left([y]^2 + [y]\right)\right]$

We use sequence (U_n) and its convergence properties to define an algorithm that approximates the sign function on the interval $[-1, 1]$.

Lemma 45. *Algorithm 27 returns shares of a secret close to* 1 *if the input* x *corresponds to a strictly positive element, and shares of a secret close to* 0 *otherwise.*

Proof. The approximations of the multiplication and the addition in secure multi-party computation make the algorithm converge to 1 if the input x is positive and to -1 in case the input is negative. The algorithm should have returned 0 had the input been null as per the previous theorem 44. The last operation correspond to a polynomial that maps 0 and -1 to 0, and 1 to 1, hence obtaining the required convergence. □

The numerical approximation of the sign function is more advantageous to use in the case where the number of iterations required to have a good performance does not result in a higher communication cost than the Rabbit algorithm.

Example 46. *For* $l = 60, p = 20$*, if we would like the algorithm to output shares of exactly* 1 *if* $x > 0$*, the algorithm requires* $k = 26$ *iterations. Indeed, given the smallest possible starting point* $x_0 = 2^{-20}$*, then*

$$|x_k - 1| \leq \mu |x_{k-1} - 1|^2 \leq \cdots \leq \mu^k |x_0 - 1|^{2^k}.$$

So we need k *such that* $\mu^k |2^{-20} - 1|^{2^k} < 2^{-20}$*, which is* $k \geq 26$ *for the value* $\mu = \frac{3}{2}$*.*

Remark 47. *The lower the number of fixed-bit precision, the lower iterations the algorithm needs to converge.*

6.2.2.2 Activation Functions for Machine Learning

The purpose of this section is to present different ways of computing some activation functions in order to evaluate how they affect the time of computation and the correctness of the output.

Sigmoid:

Sigmoid is a very popular activation function mostly used for time series. We recall that it is defined for any real element $x \in \mathbb{R}$ as the following:

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}.$$

In our work on a differentially private logistic regression [SKB22] - explained in chapter 4 - we defined an approximation of the functions sigmoid as a piecewise linear function. This idea

is not new, works like [WGC19; MZ17] suggest having piecewise linear approximations. The first approximation that we consider is defined as the following:

$$S(x) = \begin{cases} 0 & \text{if } x \leq -2 \\ \frac{1}{4}x + \frac{1}{2} & \text{if } -2 < x \leq 2 \\ 1 & \text{if } 2 < x. \end{cases}$$

We also consider an approximation that is also linear by parts, defined as the following:

$$S^1(x) = \begin{cases} 0 & \text{if } x \leq -1 \\ x + \frac{1}{2} & \text{if } -1 < x \leq 1 \\ 1 & \text{if } 1 < x. \end{cases}$$

The approximation S^1 from [MZ17] is similar to S, and both functions can be computed using twice the rabbit comparison protocol denoted by Π_{LTC}.

In the paper [Kno+21], the authors compute the function sigmoid by using the approximations of the exponential and the real inverse functions denoted as Π_{exp} and Π_{inv} and recalled in algorithms 10 and 12. We denote their approximation by S^2.

Finally we present a communication-free alternative to compute the function sigmoid, we denote it by S^3 and we define it as the following:

$$S^3(x) = \frac{1}{4}x + \frac{1}{2}.$$

This approximation is communication-free. However, if x is lower than -2 or higher than 2, then S^3 returns values outside $[0, 1]$. Therefore it is to be used when the input is guaranteed to be within the interval $[-2, 2]$.

Hyperbolic tangent:

Hyperbolic tangent or tanh is a non-linear activation function commonly used in machine learning. It takes any real value and outputs a value between -1 and 1. The output of tanh is also zero-centered, meaning the mean of its output is zero. For any real element $x \in \mathbb{R}$, it is defined as the following:

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}.$$

In a similar way to the approximations S^1 of sigmoid, we can define the approximation:

$$T^1(x) = \begin{cases} 0 & \text{if } x \leq -1 \\ x & \text{if } -1 < x \leq 1 \\ 1 \text{ if } 1 < x. \end{cases}$$

We denote by T^2 the approximation of tanh computed by using the approximations of the exponential and the real inverse functions denoted as Π_{exp} and Π_{inv}, and recalled in algorithms 10 and 12.

Finally, the communication-free alternative to compute the function tanh is denoted it by T^3 and we define it as the following:

$$T^3(x) = x.$$

Like S^3, T^3 is communication-free but if x is lower than -1 or higher than 1, then T^3 returns values outside $[-1, 1]$. Therefore it is to be used when the input is guaranteed to be within the interval $[-1, 1]$.

6.2.3 Protocol for Equality Tests on Finite Sets

In chapter 5 section 5.2.2, we presented an equality test to check if a secret is equal to a public value in a finite set, and we used it for truth-finding. The equality test corresponds to a degree-two polynomial that takes advantage of the fact that the elements that we compare are in a set of size three. In this section, we present a generalization of this protocol to a finite set of arbitrary sizes.

6.2.3.1 Construction of Polynomials for Equality Tests

We defined the following function for x a secret, and $\kappa \in \{-1, 0, 1\}$ a public value:

$$\mathcal{E}(x, \kappa) = \begin{cases} 1 \text{ if } x = \kappa \\ 0 \text{ if not,} \end{cases}$$

and the three polynomials:

$$\tfrac{1}{2}((x)^2 - x) = \begin{cases} 1 & \text{if } x = -1 \\ 0 & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases} \quad 1 - (x)^2 = \begin{cases} 0 & \text{if } x = -1 \\ 1 & \text{if } x = 0 \\ 0 & \text{if } x = 1 \end{cases} \quad \tfrac{1}{2}((x)^2 + x) = \begin{cases} 0 & \text{if } x = -1 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x = 1, \end{cases}$$

such that:

$$\begin{aligned} \frac{1}{2}((x)^2 - x) &= \mathcal{E}(x, -1), \\ 1 - (x)^2 &= \mathcal{E}(x, 0), \\ \frac{1}{2}((x)^2 + x) &= \mathcal{E}(x, 1). \end{aligned}$$

To formalize and generalize the previous definitions we consider the set $\mathcal{K} = \{\kappa^1, \ldots, \kappa^n\} \subset \mathbb{R}$. We would like to define a family of polynomials $(P_{\kappa,\mathcal{K}})_{\kappa \in \mathcal{K}} \subset \mathbb{R}[X]$, such that for $x, \kappa \in \mathcal{K}$,

$$P_{\kappa,\mathcal{K}}(x) = \mathcal{E}(x, \kappa).$$

For $\kappa \in \mathcal{K}$ and $n = |\mathcal{K}|$, $P_{\kappa,\mathcal{K}}$ has at least $n - 1$ zeros which implies that $P_{\kappa,\mathcal{K}}$ is of degree at least $n - 1$, hence no polynomial of degree strictly less than $n - 1$ could satisfy the condition $P_{\kappa,\mathcal{K}}(x) = \mathcal{E}(x, \kappa)$. In addition, a polynomial of degree exactly equal to $n - 1$ could be constructed as the following:

$$P_{\kappa,\mathcal{K}}(X) = \lambda \prod_{\substack{\kappa_i \in \mathcal{K} \\ \kappa_i \neq \kappa}} (X - \kappa_i), \lambda \in \mathbb{R}.$$

The condition $P_{\kappa,\mathcal{K}}(\kappa) = 1$ does not affect the degree of the polynomial, it only affects the real constant λ which is computed such that:

$$\frac{1}{\lambda} = \prod_{\substack{\kappa_i \in \mathcal{K} \\ \kappa_i \neq \kappa}} (\kappa - \kappa_i).$$

This way $P_{\kappa,\mathcal{K}}(\kappa) = 1$, and for $\kappa_i \in \mathcal{K}; \kappa_i \neq \kappa, P_{\kappa,\mathcal{K}}(\kappa_i) = 1$. Furthermore, this polynomial is unique. Indeed, the constructed polynomial $P_{\kappa,\mathcal{K}}$ is the polynomial of lowest degree interpolating n points hence its uniqueness (Lagrange interpolation theorem).

Algorithm 28 Equality test algorithm on a finite set protocol Π_{equal}

Private input: The players hold shares of $[x]$ and public element κ with $\kappa, x \in \mathcal{K}$
Output: The players hold shares of $[y]$
1: $[y] \leftarrow [P_{\kappa,\mathcal{K}}(x)]$

Lemma 48. *The algorithm returns shares of* 1 *if* $x = \kappa$*, and* 0 *otherwise.*

Remark 49. *Note that in a secure multi-party computation algorithm implementing the equality test on a finite set of size* n *using a polynomial requires at most* $n - 2$ *multiplications.*

Remark 50. *For* $\mathcal{K} = \{\kappa_1, \ldots, \kappa_n\}$*, the polynomial that we defined to execute equality tests in* $\mathcal{K}$ *could be redefined to implement any combination of boolean expression like for example:*

$$\mathcal{E}(x, \kappa_i) \vee \mathcal{E}(x, \kappa_j), \text{ with } i, j \in \{1, \ldots, n\},$$

which is executed in one polynomial evaluation instead of two parallel exceptions of an equality test.

6.2.3.2 Application to Restricted Boltzmann Machines

As explained in chapter 2 section 2.3.2, a restricted Boltzmann machine [FI12; Hin12] is a type of generative artificial neural network. They can be used to learn the probability distribution of a given set of data and can be used to generate new data that has a similar probability distribution to the original data set.

The authors of [SMH07] use restricted Boltzmann machines – two-layer undirected graphical models– to model tabular data, such as user's ratings of movies. They present efficient learning and inference procedures for this class of models and demonstrate that restricted Boltzmann machines can be successfully applied to the Netflix data set, containing over 100 million user/movie ratings.

More technically the rating between 0 and 5 of a movie is mapped to 1 if the rating is higher than 1 and to 0 if it is lower than that. This way the binary input represents whether a user likes a movie or not. On the other hand, the unwatched movies were labeled by -1. Their goal was to use the restricted Boltzmann machine to predict whether or not a user would enjoy an unwatched movie.

In their algorithm, a vector of input of size $r \in \mathbb{N}$ denoted by $v \in \mathcal{K}^r = \{-1, 0, 1\}^r$ is used to compute $k \in \mathbb{N}$ layers of the network to obtain $y \in \mathbb{R}^r$. At some point in the computation, the i-th element of the vector y which is y^i should be used to compute and element z^i which is equal to -1 if v^i was equal to -1 and y^i otherwise. Mathematically, this translates to the following operation:

$$\forall i \in \{1, \ldots, r\}, z^i = \begin{cases} y^i & \text{if } v^i \neq -1 \\ -1 & \text{otherwise.} \end{cases}$$

While evaluating this algorithm with secret-sharing-based secure multi-party computation, this equality test could be replaced with the polynomial $P_{-1,\mathcal{K}}(x) = \frac{x^2 - x}{2}$ in the following way:

$$\forall i \in \{1, \ldots, r\}, z^i = y^i(1 - P_{-1,\mathcal{K}}(v^i)) - P_{-1,\mathcal{K}}(v^i).$$

Indeed,

$$\begin{array}{lllllll} v^i = -1 & \implies & P_{-1,\mathcal{K}}(v^i) = 1 & \implies & z^i & = y^i \cdot (1-1) - 1 & = -1, \\ v^i = 0 & \implies & P_{-1,\mathcal{K}}(v^i) = 0 & \implies & z^i & = y^i \cdot (1-0) - 0 & = y^i, \\ v^i = 1 & \implies & P_{-1,\mathcal{K}}(v^i) = 0 & \implies & z^i & = y^i \cdot (1-0) - 0 & = y^i. \end{array}$$

6.3 Experimental Results

We start by presenting the framework in which we evaluate our experiments in section 6.3.1, before showing and comparing the results that were obtained in section 6.3.2.

6.3.1 Framework

The experiments are executed in a two-player low-level local socket network built from scratch. Three files called "*alice.py*", "*bob.py*", and "*main.py*" contain respectively the first and second players' codes and the common function they wish to evaluate. Each of the two players possesses a file with shares of random elements such as Beavers triples and EdaBits. Indeed, the framework considers only the online phase and we suppose that all preliminary material was already generated.

In order to evaluate the common function written in *main*, *alice* and *bob* call the model called *mpc* which contains a class that takes parameters such as their port number, the ring size l, and the number of bit precision p. The class from *mpc* uses two modules: *network* and *protocols*. The module *network* contains the required python code in order to define and serve a non-secure and basic local socket network with IPv4 and *SOCK STREAM* which is the socket type for the protocol that will be used to transport messages in the network. The module *protocols* on the other hand contains a class with all the algorithms and protocols related to secure multi-party computation – listed in section 6.3.1.1 – that inherits from another class called *communication* which contains all the functions related to the serialization of messages with the library *pickle*.

Once the players define the network and obtain the protocols using the module *mpc*, the players use it to evaluate the main function.

We illustrate this in figure 6.2. The code could be found on GitHub by clicking the link below[1].

6.3.1.1 The Protocols Module

The framework contains many protocols and options. Indeed, the protocols class takes many parameters such as the ring size, the number of bit precision, and the shares random elements stored locally. We note that the framework is part of ongoing work and has room for improvement.

Secret-sharing and reconstructions: A secret-sharing execution consists of mapping the real elements to the finite ring i.e. $[0, 2^l) \cap \mathbb{N}$ and then using additive secret sharing to send the corresponding share to the corresponding player. Players also have the possibility to secret-share binary elements in the finite ring $\mathbb{Z}_2$. Every secret-sharing function executed by player P_1 should be paralleled with P_2 executing a function called "receive shares" so P_2 can receive the shares of P_1 secret.

Furthermore, after the reconstruction secrets, the players have the option between mapping the ring element back to its decimal form using ϕ^{-1}, or keeping it in $[0, 2^l) \cap \mathbb{N}$

Basic functions: All the basic operations such as secret additions, subtractions, and multiplications are implemented for both ring and binary elements. There are also functions to compute additions and multiplications by a public constant $\alpha \in \mathbb{R}$. Both of these functions have the option to map the α to the finite ring using $\phi \circ \psi$.

For each secret multiplication, a Beaver's triple is used. These random triples are also used for matrix-vector and matrix-matrix multiplications. For the multiplication of two matrices of size (m, n) and (n, k) respectively, the currently implemented algorithm uses $m \cdot n \cdot k$ multiplications

[1] `https://github.com/angelos25/framework`

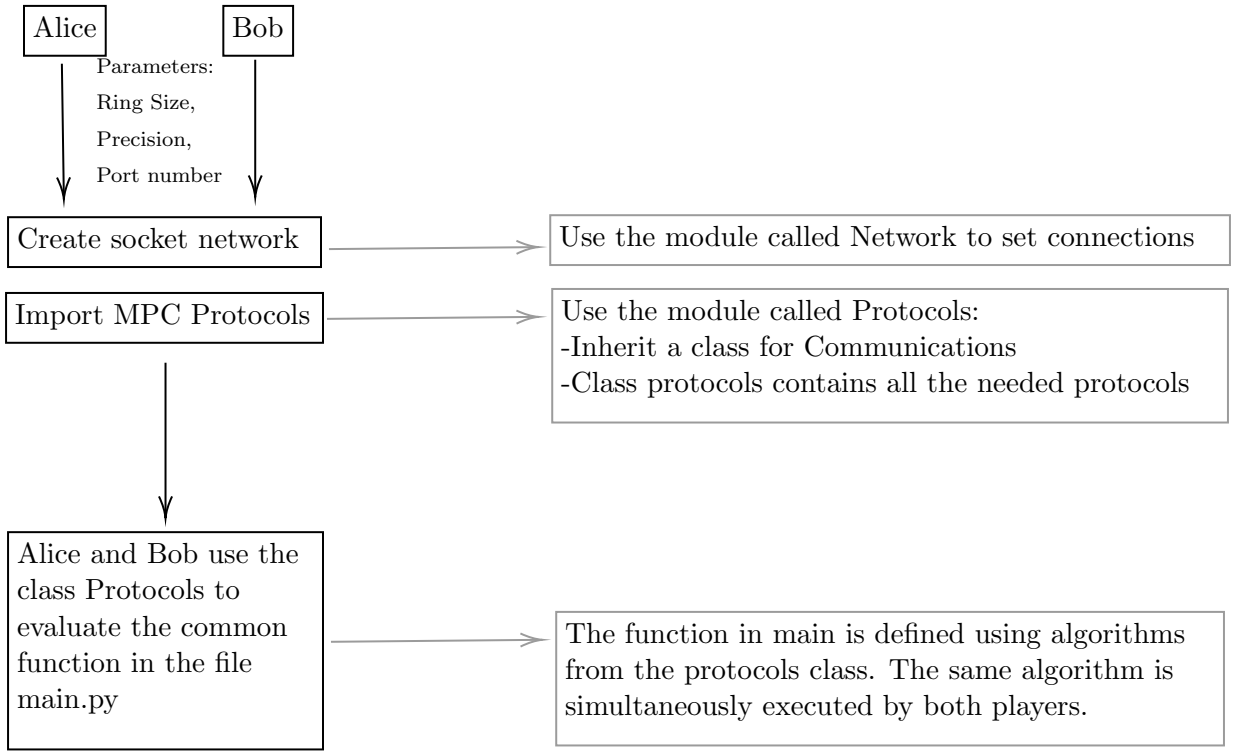

Figure 6.2: Map of the modules used to evaluate the function in *main*, starting from *alice* and *bob*.

and multiplication triples. After every multiplication, a truncation of p bits is performed using the protocol defined in [MZ17] and denoted as Π_{trunc}.

Functions for machine learning applications: There are functions like the Rabbit comparison, numerical real inverse, square root of a real element, exponentials, and natural logarithms which use is not limited to machine learning applications.

Other machine learning functions include the sigmoid activation function and all its variants. Other activation functions like ReLU, softmax, softplus and the hyperbolic tangent are also implemented.

Additionally, we have implemented the functions that allow the generation of uniformly random bits, real uniformly random elements, and random elements following Laplace distribution.

We have functions that allow us to implement a restricted Boltzmann machine such as first, the equality test to reset the output to -1 if the input is -1 as explained in section 6.2.3.2 and second, an algorithm to generate shares of a random element following Bernoulli distribution given a secret-shared probability as a parameter.

Other functions: In the protocol class, we define variables that can count the number of interactions between the players, as well as the number of multiplication, and the number of triples that were used. There is also a function that allows converting the binary shares of a bit to ring shares and functions that allow the players to compute logical operations on secret bits.

6.3.1.2 Testing the Framework

The frameworks contain several implemented examples, the most basic and important of the examples in the tutorial with *alice* and *bob*, the algorithm they should execute is defined in algorithm 29, it has to be executed by both players. As explained before, the player sets the parameters to be able to use the protocols. The players use these protocols to secret-share their data and evaluate the function defined in the main tutorial.

Algorithm 29 Tutorial code example of player P_1

Private input: Secret x^1 of the player P_1, port number, name, ring size N, number of bit precision p, module *mpc*

Output: The execution of the function defined in the main file

1: prot $\leftarrow$ mpc. Run$(port, name, N, p)$
2: $\left[x^1\right] \leftarrow$ prot. secret-share(x^i)
3: $\left[x^2\right] \leftarrow$ prot. receive-shares()
4: main(prot, $\left[x^1\right], \left[x^2\right]$)

Algorithm 30 shows what a main file looks like. In the tutorial, we use basic addition and multiplication, as well as the rabbit comparison protocol to check the sign of multiplication. The tutorial also includes an evaluation of the sigmoid function. The sigmoid function defined in the tutorial is the one approximated by S, the piecewise linear function from equation 4.2.

Algorithm 30 Tutorial code example of main function

Private input: Shares of the secrets $\left[x^1\right]$ and $\left[x^2\right]$ as well as prot

Output: The execution of the function defined in the main file

1: $[a] \leftarrow$ prot. add$(\left[x^1\right], \left[x^2\right])$
2: print("The addition is :", prot.reconstruct($[a]$))
3: $[m] \leftarrow$ prot. mul$(\left[x^1\right], \left[x^2\right])$
4: print("The multiplication is :", prot.reconstruct($[m]$))
5: $[p] \leftarrow$ prot. rabbit-compare$([m], 0)$
6: print("The positivity of the multiplication is :", prot.reconstruct($[p]$))
7: $[s] \leftarrow$ prot. sigmoid$([m])$
8: print("The sigmoid of the multiplication is :", prot.reconstruct($[s]$))

The other implementation examples in the framework are logistic regression, a restricted Boltzmann machine, the cosine algorithm paired with PATE, and the four experiments from chapter 5 containing cosine and 3-estimates with their variants.

6.3.2 Comparative Results

We first compare the output and the time of execution of the Rabbit comparison and the numerical approximation of the sign function in six different settings. We do the same with the approximations of the sigmoid function. We also test the 4 approximations of the sigmoid function on logistic regression in four different settings.

6.3.2.1 The Sign Function

We compare the output and time of execution of the Rabbit comparison and the numerical approximation of the sign function on the real interval $[-1, 1]$. To do this, we choose as input a secret

shared vector x defined as the following:

$$x = (x^i)_i; i \in [-1000, 1000] \cap \mathbb{N}, x^i = \frac{i - 1000}{1000},$$

in other words the vector $x = (-1, -0.999, 0.998, \ldots, 0.999, 1)$.

Three versions of the numerical approximation of the sign function are compared with the Rabbit protocol. Indeed the numerical approximation consists of a number of iterations, we test the function for $10, 15$ and 20 iterations. In addition we test the functions in $\mathbb{Z}_{2^{32}}$ with $8, 10$ and 12 bits of precision, and in $\mathbb{Z}_{2^{60}}$ with $15, 20$ and 25 bits of precision.

In figure 6.3 we compare the error measured as the distance between the output of the sign function in clear and each of the outputs of the Rabbit protocol and the outputs of the three numerical approximations of the sign function – with $10, 15$ and 20 iterations respectively – all computed in the ring $\mathbb{Z}_{2^{32}}$ with $8, 10$ and 12 bits of precision. The error should be between 0 and 1, the points whose image is equal to -1 correspond to points whose computation in the ring has overflowed due to errors like the ones induced by the truncation protocol we use. Overall, the best way to compute the sign function is by using the rabbit protocol or a high number of iterations using the numerical approximation. If the precision p is smaller, then there is less chance to have truncation errors, indeed, as we explained at the beginning of the chapter, the probability of having a truncation error depends on the number of bit precision.

In figure 6.4 we compared the same functions, but in the ring $\mathbb{Z}_{2^{60}}$ with also $15, 20$ and 25 bits of precision. The points whose image is equal to -1 also correspond to points whose computation in the ring has overflowed due to errors like the ones induced by the truncation protocol we use. Same to the computation in the smaller ring, the best way to compute the sign function is by using the rabbit protocol or a high number of iterations using the numerical approximation. These experiments also show that the probability of having a truncation error depends on the number of bit precision.

6.3.2.2 The Approximations of Sigmoid

We compare the output and time of execution in the sigmoid approximations in the real interval $[-5, 5]$. To do this, we choose as input a secret shared vector x defined as the following:

$$x = (x^i)_i; i \in [-50, 50] \cap \mathbb{N}, x^i = \frac{i - 50}{10},$$

in other words the vector $x = (-5, -4.9, -4.8, \ldots, 4.9, 5)$.

The four approximations of sigmoid S, S^1, S^2, and S^3 are compared. In addition we test the functions in $\mathbb{Z}_{2^{32}}$ with $8, 10$ and 12 bits of precision, and in $\mathbb{Z}_{2^{60}}$ with $15, 20$ and 25 bits of precision.

In figure 6.3 we compare the error measured as the distance between the output of the sigmoid function in clear and each of the approximations of sigmoid all computed in the ring $\mathbb{Z}_{2^{32}}$ with $8, 10$ and 12 bits of precision. The error should be between 0 and 1, the points whose image is equal to -1 correspond to a point whose computation in the ring has overflowed due to errors like the ones induced by the truncation protocol we use. The approximation with the smallest error is the one that numerically computes the exponent and the inverse defined in the sigmoid function. However using these algorithms makes it more probable for the computation to diverge from the expected results due to truncation errors, indeed, more multiplications mean more truncations that could go bad.

In figure 6.4 we compared the same functions, but in the ring $\mathbb{Z}_{2^{60}}$ with $15, 20$ and 25 bits of precision. The images equal to -1 also correspond to points whose computation in the ring has

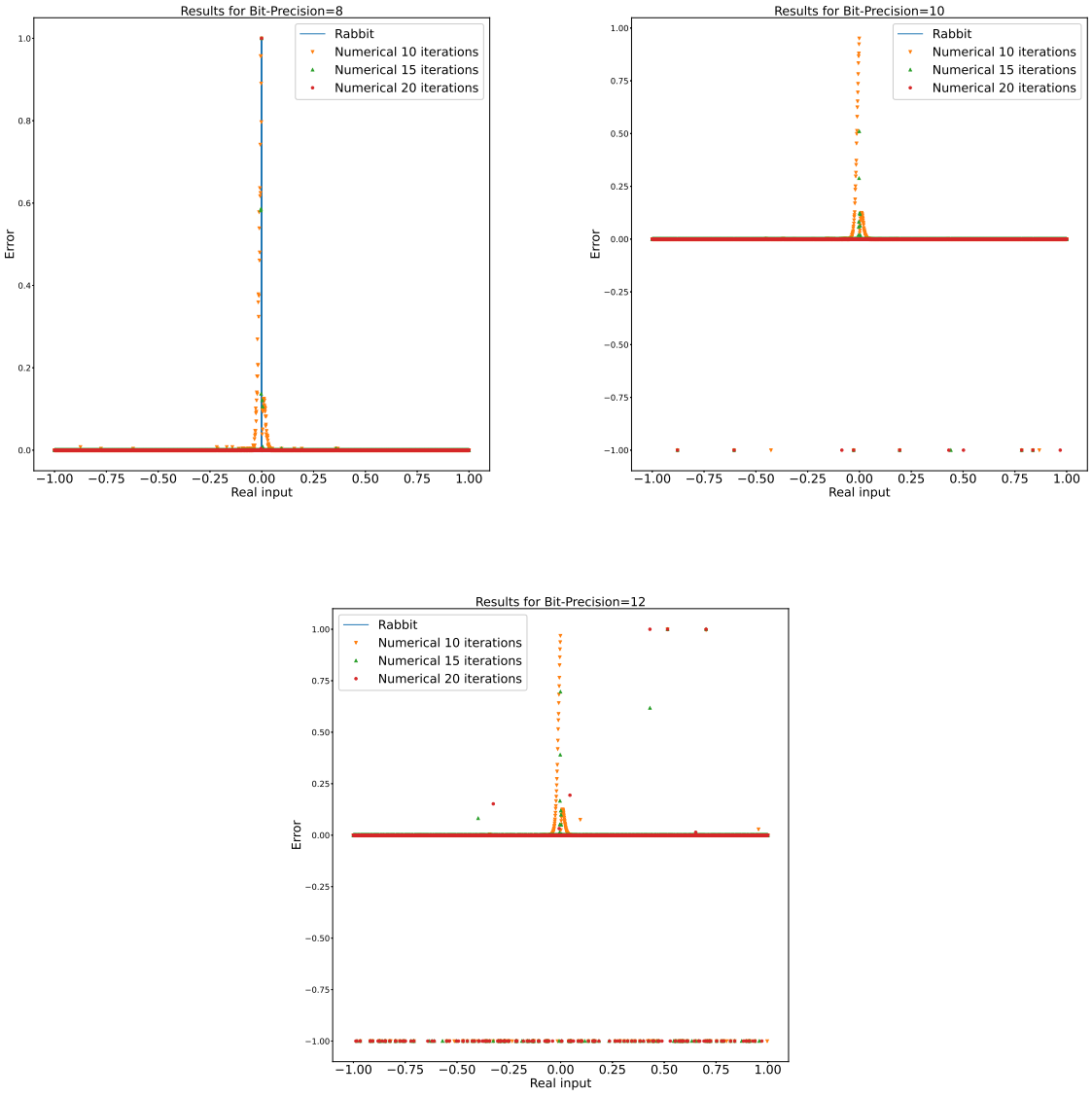

Figure 6.3: Comparison of the absolute value errors of the Rabbit protocol and the numerical approximation of the sign function with $10, 15$ and 20 iterations in the ring $\mathbb{Z}_{2^{32}}$ with $8, 10$ and 12 precision. The error is measured using the output of the sign function in clear.

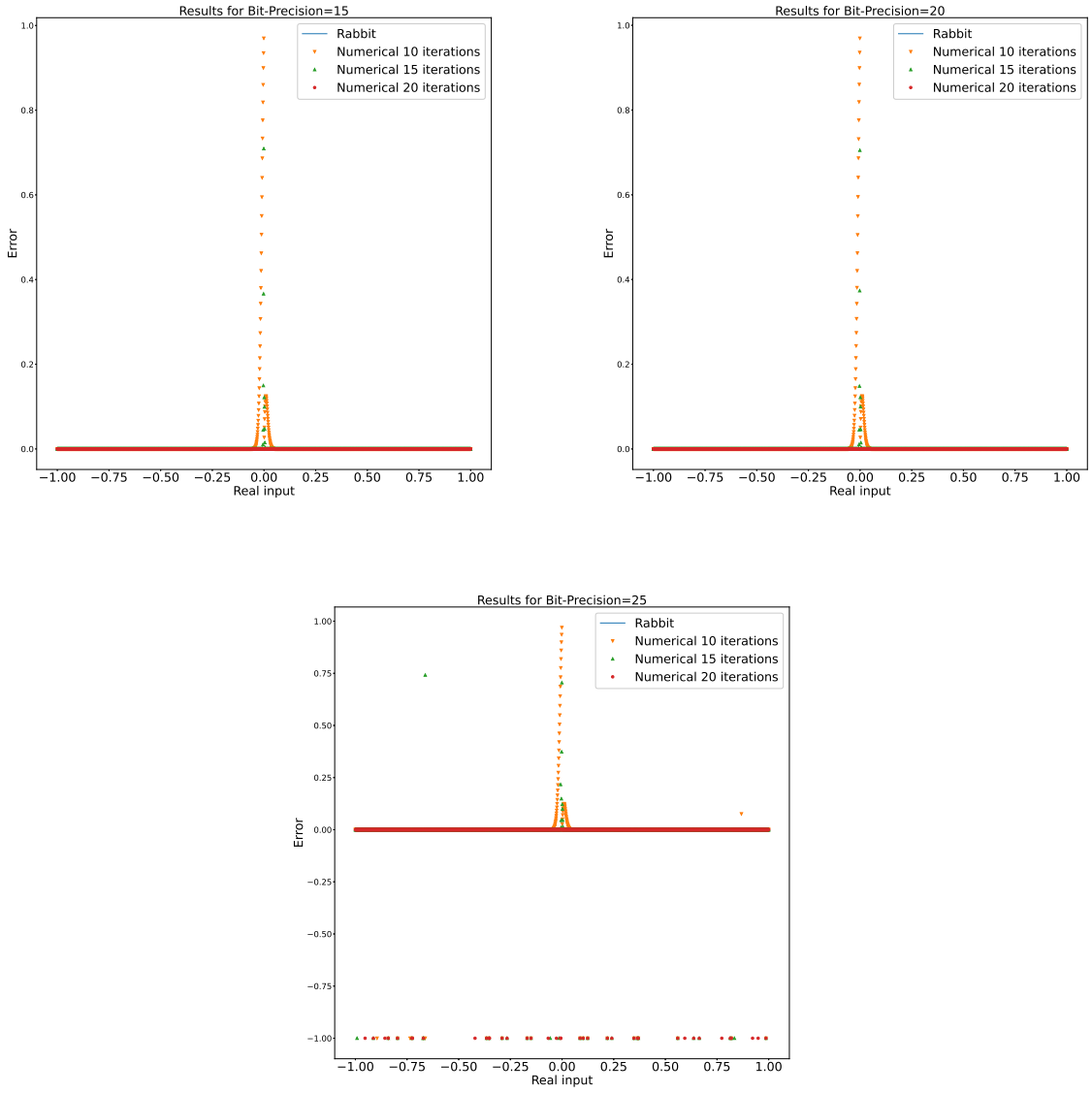

Figure 6.4: Comparison of the absolute value errors of the Rabbit protocol and the numerical approximation of the sign function with $10, 15$ and 20 iterations in the ring $\mathbb{Z}_{2^{60}}$ with $15, 20$ and 25 precision. The error is measured using the output of the sign function in clear.

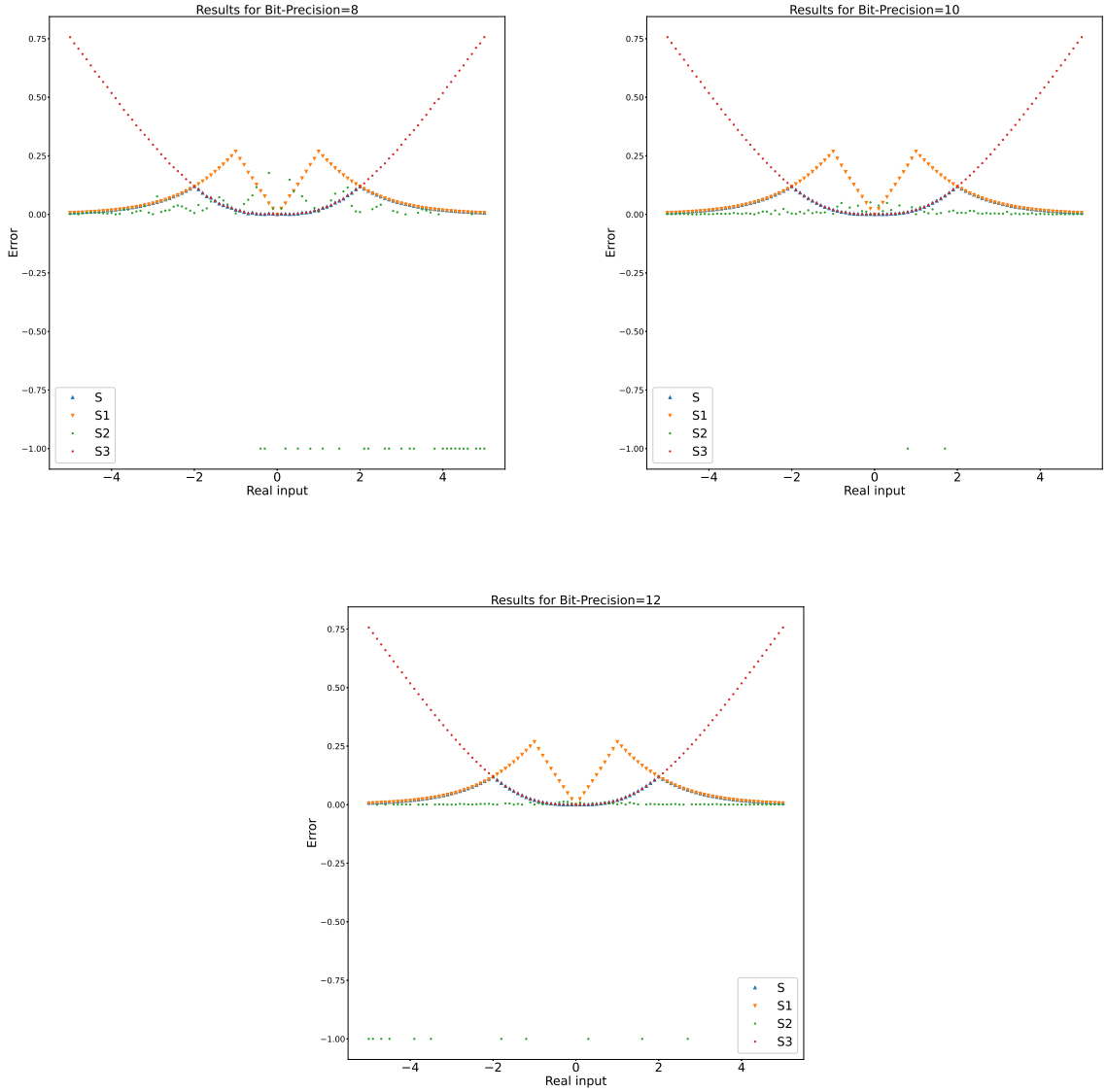

Figure 6.5: Comparison of the absolute value errors of the approximations of sigmoid in the ring $\mathbb{Z}_{2^{32}}$ with $8, 10$ and 12 precision. The error is measured using the output of the sigmoid function in clear.

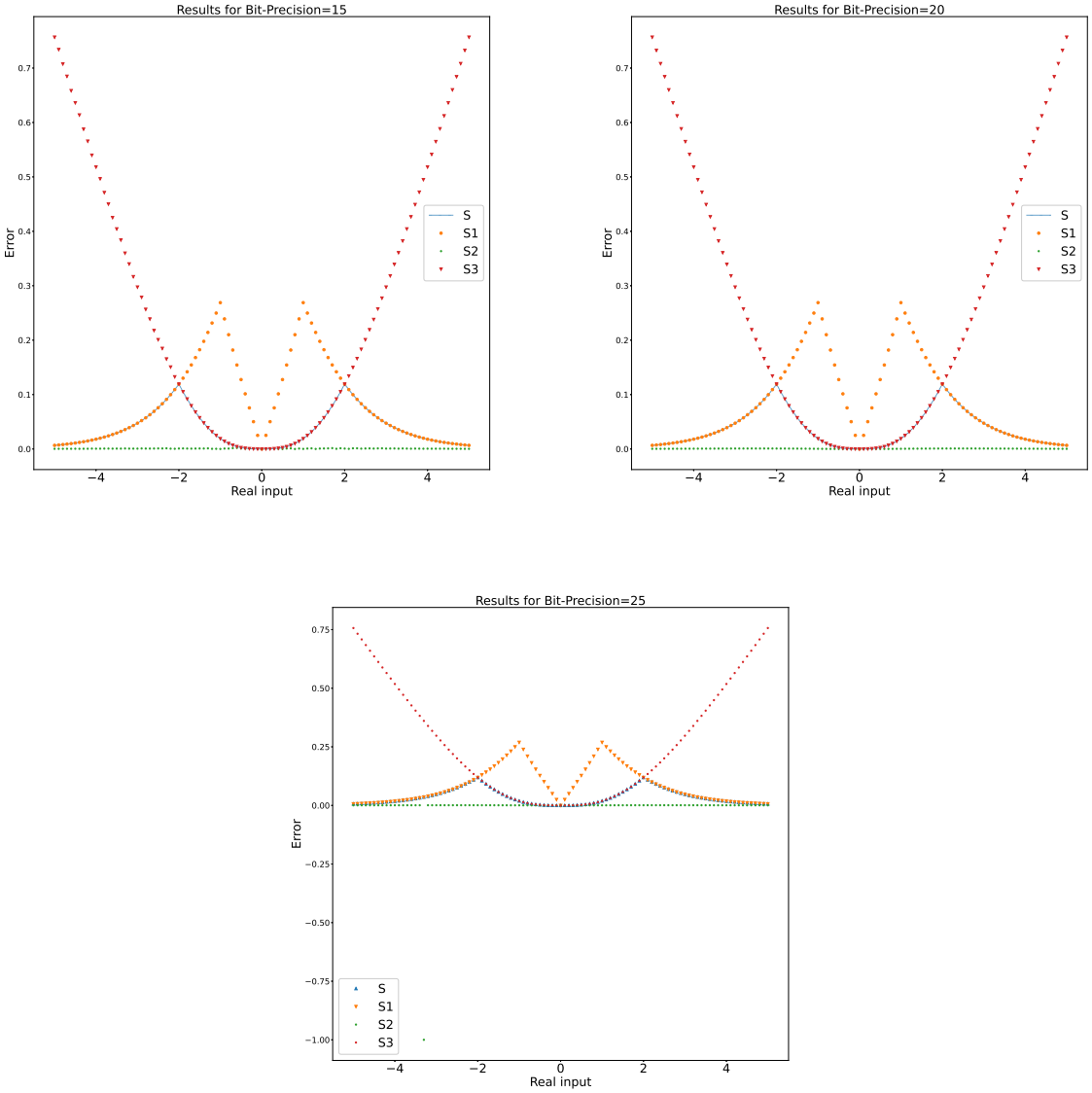

Figure 6.6: Comparison of the absolute value errors of the approximations of sigmoid in the ring $\mathbb{Z}_{2^{60}}$ with $15, 20$ and 25 precision. The error is measured using the output of the sigmoid function in clear.

Table 6.1: A comparative table of the wall and CPU time of execution in seconds, of the Rabbit protocol and the numerical approximation of the sign function on 2000 real secrets. The numerical approximation is timed for $10, 15$ and 20 iterations denoted as Num_{10}, Num_{15} and Num_{20} respectively. the experiments do not show that the ring size affects the computation even though it should as the rabbit protocol's complexity depends on the size of the ring. We only notice that using the numerical comparison is more efficient, and the more iterations the more time the execution requires.

Ring	Precision	Time measured	Rabbit	Num_{10}	Num_{15}	Num_{20}
$\mathbb{Z}_{2^{32}}$	8	Wall	0.283	0.048	0.062	0.085
$\mathbb{Z}_{2^{32}}$	8	CPU	0.244	0.032	0.042	0.056
$\mathbb{Z}_{2^{32}}$	10	Wall	0.313	0.043	0.062	0.084
$\mathbb{Z}_{2^{32}}$	10	CPU	0.275	0.032	0.050	0.062
$\mathbb{Z}_{2^{32}}$	12	Wall	0.337	0.055	0.072	0.086
$\mathbb{Z}_{2^{32}}$	12	CPU	0.278	0.036	0.050	0.063
$\mathbb{Z}_{2^{60}}$	15	Wall	0.469	0.044	0.60	0.700
$\mathbb{Z}_{2^{60}}$	15	CPU	0.412	0.35	0.044	0.060
$\mathbb{Z}_{2^{60}}$	20	Wall	0.517	0.046	0.58	0.79
$\mathbb{Z}_{2^{60}}$	20	CPU	0.482	0.035	0.044	0.060
$\mathbb{Z}_{2^{60}}$	25	Wall	0.511	0.044	0.057	0.079
$\mathbb{Z}_{2^{60}}$	25	CPU	0.473	0.033	0.043	0.054

overflowed. Unlike the computations in the smaller ring, there are fewer truncation errors, this is due to the fact that the ring is bigger which lowers the probability of having truncations go wrong. We can notice that with 25 bits of precision, there are some errors due to truncations because as mentioned before, with higher precision the probability of wrong truncations is higher.

To evaluate the impact of the approximations, we use each to compute a logistic regression on dummy data. In table 6.3 we measure the time of execution of training the logistic regression using the four approximations S, S^1, S^2, S^3 of the sigmoid function and the accuracy of each trained model. The results show that the ring size affects the training time of the algorithms that use S and S^1 as they rely on the Rabbit comparison whose complexity is expressed in the size of the ring. We notice that the algorithms that use S and S^1 have the same time of execution, which is expected because they are defined with three linear functions and two comparisons. The algorithm trained using the approximation S^2 takes less time than S and S^1, but naturally, the most efficient algorithm is the one that uses the last approximation S^3 as it consists of linear (communication-free) operations.

With 8 bits of precision, the parameters of the model trained with S^2 didn't return the expected output, this is due to truncation errors. We notice that the models trained with S^1 have the highest accuracy (0.90) whereas the model trained in clear on the same data and with the same number of iterations and epochs has an accuracy of 0.88 – and so do the models trained with the approximations S, S^2, S^3. Indeed, the function S^1 had the highest error compared to the other three, which explains why it behaves differently when training the model. This just means that the function S^1 made the convergence faster for this specific dataset.

Table 6.2: A comparative table of the wall and CPU time of execution in seconds, of the approximations of sigmoid on 100 real secrets.

Ring	Precision	Time measured	S	S^1	S^2	S^3
$\mathbb{Z}_{2^{32}}$	8	Wall	0.093	0.103	0.035	0.0008
$\mathbb{Z}_{2^{32}}$	8	CPU	0.069	0.073	0.020	0.0005
$\mathbb{Z}_{2^{32}}$	10	Wall	0.124	0.126	0.055	0.001
$\mathbb{Z}_{2^{32}}$	10	CPU	0.094	0.096	0.032	0.0006
$\mathbb{Z}_{2^{32}}$	12	Wall	0.126	0.138	0.051	0.001
$\mathbb{Z}_{2^{32}}$	12	CPU	0.089	0.095	0.033	0.0006
$\mathbb{Z}_{2^{60}}$	15	Wall	0.171	0.181	0.049	0.0008
$\mathbb{Z}_{2^{60}}$	15	CPU	0.125	0.131	0.031	0.0004
$\mathbb{Z}_{2^{60}}$	20	Wall	0.180	0.183	0.051	0.0008
$\mathbb{Z}_{2^{60}}$	20	CPU	0.132	0.132	0.030	0.0004
$\mathbb{Z}_{2^{60}}$	25	Wall	0.193	0.179	0.045	0.0006
$\mathbb{Z}_{2^{60}}$	25	CPU	0.138	0.129	0.027	0.0003

Table 6.3: A comparative table of the wall and CPU time of execution in seconds, of training a logistic regression on a dummy dataset using the four different approximations of the sigmoid function, as well as the accuracy of each trained model.

Ring	Precision	Measure	S	S^1	S^2	S^3
$\mathbb{Z}_{2^{32}}$	8	Wall	0.580	0.596	0.235	0.014
$\mathbb{Z}_{2^{32}}$	8	CPU	0.431	0.446	0.151	0.010
$\mathbb{Z}_{2^{32}}$	8	Accuracy	0.91	0.97	0.55	0.94
$\mathbb{Z}_{2^{32}}$	10	Wall	0.589	0.556	0.223	0.015
$\mathbb{Z}_{2^{32}}$	10	CPU	0.414	0.390	0.132	0.010
$\mathbb{Z}_{2^{32}}$	10	Accuracy	0.88	0.97	0.88	0.88
$\mathbb{Z}_{2^{60}}$	15	Wall	1.104	1.102	0.243	0.015
$\mathbb{Z}_{2^{60}}$	15	CPU	0.759	0.776	0.164	0.012
$\mathbb{Z}_{2^{60}}$	15	Accuracy	0.88	0.97	0.88	0.88
$\mathbb{Z}_{2^{60}}$	20	Wall	1.085	1.083	0.242	0.016
$\mathbb{Z}_{2^{60}}$	20	CPU	0.733	0.732	0.141	0.011
$\mathbb{Z}_{2^{60}}$	20	Accuracy	0.88	0.97	0.88	0.88

6.4 Further Discussion and Conclusion

In this chapter, we explained how the approximations in secure multi-party computation affect the computation and how the truncation protocol that we use affects two-party computation in the fixed-point precision setting. We also presented alternatives for the equality test, for the sign function, and for some activation functions. We then implemented these primitives for machine learning applications, in a socket network for two players.

We explained how we implemented the framework. Its construction was motivated by the lack of a low-level and basic framework that can be used for research purposes with fine-grained control over all the functions including the most basic ones such as secret-sharing and multiplications, as well as having a way to send messages across the network. Indeed some of the protocols and algorithms that are implemented are not optimal, for example, the matrix-matrix multiplications and the prefix-OR functions could be implemented in a more efficient way.

Furthermore, the current piecewise approximations of sigmoid might give good results on classification tasks but might fail for regression analysis. All the tests were conducted on classification tasks where the exact output of sigmoid did not matter as much as if its image is higher or lower than $\frac{1}{2}$. For other machine learning tasks the output of sigmoid might matter more and the piecewise approximations might not be the best way to obtain the desired results. Therefore, a direction of research would be to check the impact of the approximation on other machine learning tasks.

Suppose we want to compute a real function like the inverse or the square root of an element x. Instead of using ϕ to find the zeros $\frac{1}{x}$ of a real function f, the idea would be to define polynomial $\hat{f}$ in the finite ring $\mathbb{Z}_{2^l}$ such that its roots are equal to $\phi(\frac{1}{x})$. We would then use Hensel's lemma and Newton's iterations to efficiently and quickly find the roots of the polynomial $\hat{f}$, hence the inverse of x with more precision.

References

[Aba+16] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. "Deep Learning with Differential Privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM, 2016, pp. 308–318. DOI: 10.1145/2976749.2978318. URL: https://doi.org/10.1145/2976749.2978318 (cited on page 56).

[Aga+22] Amit Agarwal, Stanislav Peceny, Mariana Raykova, Phillipp Schoppmann, and Karn Seth. "Communication Efficient Secure Logistic Regression". In: *IACR Cryptol. ePrint Arch.* (2022), p. 866. URL: https://eprint.iacr.org/2022/866 (cited on page 8).

[Agr+19] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. "QUOTIENT: Two-Party Secure Neural Network Training and Prediction". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM, 2019, pp. 1231–1247. DOI: 10.1145/3319535.3339819 (cited on page 11).

[Ami+22] Kareem Amin, Jennifer Gillenwater, Matthew Joseph, Alex Kulesza, and Sergei Vassilvitskii. "Plume: Differential Privacy at Scale". In: *CoRR* abs/2201.11603 (2022). arXiv: 2201.11603 (cited on page 5).

[Aon+15] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. "Fast and Secure Linear Regression and Biometric Authentication with Security Update". In: *IACR Cryptol. ePrint Arch.* (2015), p. 692. URL: http://eprint.iacr.org/2015/692 (cited on page 6).

[Bal+19] Marshall Ball, Brent Carmer, Tal Malkin, Mike Rosulek, and Nichole Schimanski. "Garbled Neural Networks are Practical". In: *IACR Cryptol. ePrint Arch.* (2019), p. 338. URL: https://eprint.iacr.org/2019/338 (cited on page 80).

[Bea91] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Springer, 1991, pp. 420–432. DOI: 10.1007/3-540-46766-1_34 (cited on page 25).

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by Janos Simon. ACM, 1988, pp. 1–10. DOI: 10.1145/62212.62213 (cited on pages 4, 21).

[Ber44] Joseph Berkson. “Application of the logistic function to bio-assay”. In: *Journal of the American statistical association* 39.227 (1944), pp. 357–365 (cited on page 29).

[Ber15] Laure Berti-Équille. “Data veracity estimation with ensembling truth discovery methods”. In: *2015 IEEE International Conference on Big Data (IEEE BigData 2015), Santa Clara, CA, USA, October 29 - November 1, 2015*. IEEE Computer Society, 2015, pp. 2628–2636. DOI: 10.1109/BigData.2015.7364062 (cited on pages 9, 58).

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. “Functional Encryption: Definitions and Challenges”. In: *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*. Ed. by Yuval Ishai. Vol. 6597. Lecture Notes in Computer Science. Springer, 2011, pp. 253–273. DOI: 10.1007/978-3-642-19571-6_16 (cited on page 8).

[BP20] David Byrd and Antigoni Polychroniadou. “Differentially private secure multi-party computation for federated learning in financial applications”. In: *ICAIF ’20: The First ACM International Conference on AI in Finance, New York, NY, USA, October 15-16, 2020*. Ed. by Tucker Balch. ACM, 2020, 16:1–16:9. DOI: 10.1145/3383455.3422562 (cited on pages 7, 13, 14).

[Cam18] HHM Campmans. “Optimizing Convolutional Neural Networks in Multi-Party Computation”. PhD thesis. Master’s thesis, Dept of Mathematics and Computer Science, TU Eindhoven, 2018 (cited on pages 73, 77).

[CS10] Octavian Catrina and Amitabh Saxena. “Secure Computation with Fixed-Point Numbers”. In: *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers*. Ed. by Radu Sion. Vol. 6052. Lecture Notes in Computer Science. Springer, 2010, pp. 35–50. DOI: 10.1007/978-3-642-14577-3_6 (cited on pages 16, 39, 59, 77).

[CMS11] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. “Differentially Private Empirical Risk Minimization”. In: *J. Mach. Learn. Res.* 12 (2011), pp. 1069–1109. DOI: 10.5555/1953048.2021036. URL: https://dl.acm.org/doi/10.5555/1953048.2021036 (cited on pages X, 5, 13, 46).

[Che+19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. “Efficient Multi-Key Homomorphic Encryption with Packed Ciphertexts with Application to Oblivious Neural Network Inference”. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. ACM, 2019, pp. 395–412. DOI: 10.1145/3319535.3363207 (cited on page 13).

[Che+20] Hao Chen, Miran Kim, Ilya P. Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. “Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning”. In: *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*. Lecture Notes in Computer Science. Springer, 2020 (cited on page 13).

[Chi+16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. “A Homomorphic LWE Based E-voting Scheme”. In: *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*. Ed. by Tsuyoshi Takagi. Vol. 9606. Lecture Notes in Computer Science. Springer, 2016, pp. 245–265. DOI: 10.1007/978-3-319-29360-8_16 (cited on pages 9, 15).

[Coc+15] Martine De Cock, Rafael Dowsley, Anderson C. A. Nascimento, and Stacey C. Newman. "Fast, Privacy Preserving Linear Regression over Distributed Datasets based on Pre-Distributed Data". In: *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, AISec 2015, Denver, Colorado, USA, October 16, 2015*. Ed. by Indrajit Ray, Xiaofeng Wang, Kui Ren, Christos Dimitrakakis, Aikaterini Mitrokotsa, and Arunesh Sinha. ACM, 2015, pp. 3–14. DOI: 10.1145/2808769.2808774. URL: https://doi.org/10.1145/2808769.2808774 (cited on page 8).

[Coc+20] Martine De Cock, Rafael Dowsley, Anderson C. A. Nascimento, Davis Railsback, Jianwei Shen, and Ariel Todoki. "High Performance Logistic Regression for Privacy-Preserving Genome Analysis". In: *IACR Cryptol. ePrint Arch.* (2020), p. 171. URL: https://eprint.iacr.org/2020/171 (cited on page 8).

[CDN15] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. ISBN: 9781107043053 (cited on pages 12, 15, 22, 23, 27, 49).

[Dah+18] Morten Dahl, Jason Mancuso, Yann Dupis, Ben Decoste, Morgan Giraud, Ian Livingstone, Justin Patriquin, and Gavin Uhma. "Private Machine Learning in TensorFlow using Secure Computation". In: *CoRR* abs/1810.08130 (2018). arXiv: 1810.08130 (cited on page 16).

[Dam+19] Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. "New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1102–1120. DOI: 10.1109/SP.2019.00078 (cited on pages 14, 46).

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. "ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation". In: *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015 (cited on page 11).

[Den12] Li Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]". In: *IEEE Signal Process. Mag.* 29.6 (2012), pp. 141–142. DOI: 10.1109/MSP.2012.2211477 (cited on page 65).

[Dil+17] Joshua V. Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matthew D. Hoffman, and Rif A. Saurous. "TensorFlow Distributions". In: *CoRR* abs/1711.10604 (2017). arXiv: 1711.10604 (cited on page 16).

[Dwo+06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*. Ed. by Shai Halevi and Tal Rabin. Vol. 3876. Lecture Notes in Computer Science. Springer, 2006, pp. 265–284. DOI: 10.1007/11681878_14 (cited on pages 5, 34).

[DR14] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Found. Trends Theor. Comput. Sci.* 9.3-4 (2014), pp. 211–407. DOI: 10.1561/0400000042 (cited on pages IX, 34, 68).

[Esc+20] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. "Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits". In: *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. Lecture Notes in Computer Science. Springer, 2020, pp. 823–852. DOI: 10.1007/978-3-030-56880-1_29 (cited on pages 14, 16, 39, 41, 77, 78, 80).

[FC02] Raphael Feraud and Fabrice Clerot. "A methodology to explain neural network classification". In: *Neural Networks* 15.1 (2002), pp. 237–246. DOI: 10.1016/S0893-6080(01)00127-7 (cited on page 10).

[FI12] Asja Fischer and Christian Igel. "An Introduction to Restricted Boltzmann Machines". In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications - 17th Iberoamerican Congress, CIARP 2012, Buenos Aires, Argentina, September 3-6, 2012. Proceedings*. Ed. by Luis Álvarez, Marta Mejail, Luís Gómez Déniz, and Julio C. Jacobo. Vol. 7441. Lecture Notes in Computer Science. Springer, 2012, pp. 14–36. DOI: 10.1007/978-3-642-33275-3_2 (cited on pages 10, 31, 85).

[Gal+10] Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. "Corroborating information from disagreeing views". In: *Proceedings of the Third International Conference on Web Search and Web Data Mining, WSDM 2010, New York, NY, USA, February 4-6, 2010*. Ed. by Brian D. Davison, Torsten Suel, Nick Craswell, and Bing Liu. ACM, 2010, pp. 131–140. DOI: 10.1145/1718487.1718504 (cited on pages X, 9, 15, 32–35, 58, 64, 65).

[Gam85] Taher El Gamal. "A public key cryptosystem and a signature scheme based on discrete logarithms". In: *IEEE Trans. Inf. Theory* 31.4 (1985), pp. 469–472. DOI: 10.1109/TIT.1985.1057074 (cited on pages 4, 10).

[Gao+19] Dashan Gao, Yang Liu, Anbu Huang, Ce Ju, Han Yu, and Qiang Yang. "Privacy-preserving Heterogeneous Federated Transfer Learning". In: *2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9-12, 2019*. Ed. by Chaitanya K. Baru, Jun Huan, Latifur Khan, Xiaohua Hu, Ronay Ak, Yuanyuan Tian, Roger S. Barga, Carlo Zaniolo, Kisung Lee, and Yanfang (Fanny) Ye. IEEE, 2019, pp. 2552–2559. DOI: 10.1109/BigData47090.2019.9005992 (cited on pages 8, 14).

[Gen09] Craig Gentry. "Fully homomorphic encryption using ideal lattices". In: *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*. Ed. by Michael Mitzenmacher. ACM, 2009, pp. 169–178. DOI: 10.1145/1536414.1536440 (cited on page 4).

[GTJ22] Ali Reza Ghavamipour, Fatih Turkmen, and Xiaoqian Jiang. "Privacy-preserving logistic regression with secret sharing". In: *BMC Medical Informatics Decis. Mak.* 22.1 (2022), p. 89. DOI: 10.1186/s12911-022-01811-y (cited on pages 6, 14).

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. Ed. by Alfred V. Aho. ACM, 1987, pp. 218–229. DOI: 10.1145/28395.28420 (cited on page 4).

[Han07] David J Hand. "Principles of data mining". In: *Drug safety* 30 (2007), pp. 621–622 (cited on page 28).

[Har+17] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption". In: *CoRR* abs/1711.10677 (2017). arXiv: 1711.10677 (cited on pages 7, 14).

[Hin12] Geoffrey E. Hinton. "A Practical Guide to Training Restricted Boltzmann Machines". In: *Neural Networks: Tricks of the Trade - Second Edition*. Ed. by Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller. Vol. 7700. Lecture Notes in Computer Science. Springer, 2012, pp. 599–619. DOI: 10.1007/978-3-642-35289-8_32 (cited on pages 10, 31, 85).

[IPU] IPUMS. *Integrated Public Use Microdata Series (IPUMS) - International*. https://international.ipums.org/international/index.shtml (cited on page 50).

[Jay+18] Bargav Jayaraman, Lingxiao Wang, David Evans, and Quanquan Gu. "Distributed Learning without Distress: Privacy-Preserving Empirical Risk Minimization". In: (2018). Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, pp. 6346–6357. URL: https://proceedings.neurips.cc/paper/2018/hash/7221e5c8ec6b08ef6d3f9ff3ce6eb1d1-Abstract.html (cited on pages 7, 12, 14).

[JZG22] Xue Jiang, Xuebing Zhou, and Jens Grossklags. "Comprehensive Analysis of Privacy Leakage in Vertical Federated Learning During Prediction". In: *Proc. Priv. Enhancing Technol.* 2022.2 (2022), pp. 263–281. DOI: 10.2478/popets-2022-0045 (cited on pages 7, 14).

[Jun+13] Taeho Jung, XuFei Mao, Xiang-Yang Li, Shaojie Tang, Wei Gong, and Lan Zhang. "Privacy-preserving data aggregation without secure channel: Multivariate polynomial evaluation". In: *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*. IEEE, 2013, pp. 2634–2642. DOI: 10.1109/INFCOM.2013.6567071 (cited on page 9).

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. "GAZELLE: A Low Latency Framework for Secure Neural Network Inference". In: *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. Ed. by William Enck and Adrienne Porter Felt. USENIX Association, 2018, pp. 1651–1669. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar (cited on page 10).

[KW15] Liina Kamm and Jan Willemson. "Secure floating point arithmetic and private satellite collision analysis". In: *Int. J. Inf. Sec.* 14.6 (2015), pp. 531–548. DOI: 10.1007/s10207-014-0271-8 (cited on page 73).

[Kan+20] Yilin Kang, Yong Liu, Ben Niu, Xinyi Tong, Likun Zhang, and Weiping Wang. "Input Perturbation: A New Paradigm between Central and Local Differential Privacy". In: *CoRR* abs/2002.08570 (2020). arXiv: 2002.08570 (cited on pages 5, 14).

[KK02] David G. Kleinbaum and Mitchel Klein. *Logistic Regression: A Self-Learning Text*. Ed. by K. Dietz, M. Gail, K. Krickeberg, J. Samet, and A. Tsiatis. Second. Vol. 53. Springer, 2002, p. 513 (cited on pages 6, 31).

[KÖA23] Jakub Klemsa, Melek Önen, and Yavuz Akin. "A Practical TFHE-Based Multi-Key Homomorphic Encryption with Linear Complexity and Low Noise Growth". In: *IACR Cryptol. ePrint Arch.* (2023), p. 65. URL: https://eprint.iacr.org/2023/065 (cited on page 13).

[Kno+21] Brian Knott, Shobha Venkataraman, Awni Y. Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. "CrypTen: Secure Multi-Party Computation Meets Machine Learning". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan. 2021, pp. 4961–4973 (cited on pages 9, 11, 40, 41, 59, 64, 83).

[Lam+22] Maximilian Lam, Michael Mitzenmacher, Vijay Janapa Reddi, Gu-Yeon Wei, and David Brooks. "Tabula: Efficiently Computing Nonlinear Activation Functions for Secure Neural Network Inference". In: *CoRR* abs/2203.02833 (2022). DOI: 10.48550/arXiv.2203.02833. arXiv: 2203.02833 (cited on page 80).

[Li+12] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. "Truth Finding on the Deep Web: Is the Problem Solved?" In: *Proc. VLDB Endow.* 6.2 (2012), pp. 97–108. DOI: 10.14778/2535568.2448943. URL: http://www.vldb.org/pvldb/vol6/p97-li.pdf (cited on pages 9, 58).

[LDZ22] Yaliang Li, Bolin Ding, and Jingren Zhou. "A Practical Introduction to Federated Learning". In: *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*. Ed. by Aidong Zhang and Huzefa Rangwala. ACM, 2022, pp. 4802–4803. DOI: 10.1145/3534678.3542631 (cited on page 58).

[Li+15] Yaliang Li, Jing Gao, Chuishi Meng, Qi Li, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. "A Survey on Truth Discovery". In: *SIGKDD Explor.* 17.2 (2015), pp. 1–16. DOI: 10.1145/2897350.2897352 (cited on page 58).

[Li+16] Yaliang Li, Qi Li, Jing Gao, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. "Conflicts to Harmony: A Framework for Resolving Conflicts in Heterogeneous Data by Truth Discovery". In: *IEEE Trans. Knowl. Data Eng.* 28.8 (2016), pp. 1986–1999. DOI: 10.1109/TKDE.2016.2559481 (cited on pages 10, 15).

[LZJ14] Yu Li, Yuan Zhang, and Yue Ji. "Privacy-Preserving Restricted Boltzmann Machine". In: *Comput. Math. Methods Medicine* 2014 (2014), 138498:1–138498:7. DOI: 10.1155/2014/138498 (cited on page 10).

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption". In: *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*. Ed. by Howard J. Karloff and Toniann Pitassi. ACM, 2012, pp. 1219–1234. DOI: 10.1145/2213977.2214086 (cited on page 13).

[Mak+21] Eleftheria Makri, Dragos Rotaru, Frederik Vercauteren, and Sameer Wagh. "Rabbit: Efficient Comparison for Secure Multi-Party Computation". In: *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part I*. Ed. by Nikita Borisov and Claudia Díaz. Vol. 12674. Lecture Notes in Computer Science. Springer, 2021, pp. 249–270. DOI: 10.1007/978-3-662-64322-8_12 (cited on pages 17, 39, 41, 42, 70, 78–80).

[Met20] Meta. *Private Lift Measurement, https://github.com/w3c/web-advertising/blob/main/private-lift-measurement-conceptual-overview.md*. 2020 (cited on page 3).

[Mia+15] Chenglin Miao, Wenjun Jiang, Lu Su, Yaliang Li, Suxin Guo, Zhan Qin, Houping Xiao, Jing Gao, and Kui Ren. "Cloud-Enabled Privacy-Preserving Truth Discovery in Crowd Sensing Systems". In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems, SenSys 2015, Seoul, South Korea, November 1-4, 2015*. Ed. by Junehwa Song, Tarek F. Abdelzaher, and Cecilia Mascolo. ACM, 2015, pp. 183–196. DOI: 10.1145/2809695.2809719 (cited on pages 9, 15).

[Mit97] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. ISBN: 978-0-07-042807-2. URL: https://www.worldcat.org/oclc/61321007 (cited on page 27).

[MR18] Payman Mohassel and Peter Rindal. "ABY^3: A Mixed Protocol Framework for Machine Learning". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang. ACM, 2018, pp. 35–52. DOI: 10.1145/3243734.3243760 (cited on pages 9, 11, 12, 14, 59).

[MZ17] Payman Mohassel and Yupeng Zhang. "SecureML: A System for Scalable Privacy-Preserving Machine Learning". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 19–38. DOI: 10.1109/SP.2017.12 (cited on pages 8, 12, 14, 16, 39, 47, 76, 77, 83, 87).

[MPV01] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. *Introduction to linear regression analysis*. Wiley series in probability and statistics. New York, NY: Wiley, 2001 (cited on pages 6, 29).

[NBK15] Divya G. Nair, V. P. Binu, and G. Santhosh Kumar. "An Improved E-voting scheme using Secret Sharing based Secure Multi-party Computation". 2015. arXiv: 1502.07469 (cited on pages 9, 15).

[NHC22] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. "Local and Central Differential Privacy for Robustness and Privacy in Federated Learning". In: (2022) (cited on page 10).

[Nas+20] Reza Nasirigerdeh, Reihaneh Torkzadehmahani, Julian Matschinske, Tobias Frisch, Markus List, Julian Späth, Stefan Weiß, Uwe Völker, Dominik Heider, Nina Kerstin Wenke, et al. "sPLINK: a federated, privacy-preserving tool as a robust alternative to meta-analysis in genome-wide association studies". In: *BioRxiv* (2020) (cited on pages 6, 14).

[NNM07] Khalida Inayat Noor, Muhammad Aslam Noor, and Shaher Momani. "Modified Householder iterative method for nonlinear equations". In: *Appl. Math. Comput.* 190.2 (2007), pp. 1534–1539. DOI: 10.1016/j.amc.2007.02.036 (cited on page 40).

[Pai99] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Springer, 1999, pp. 223–238. DOI: 10.1007/3-540-48910-X_16 (cited on page 4).

[PY10] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning”. In: *IEEE Trans. Knowl. Data Eng.* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191 (cited on page 8).

[Pap+17] Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian J. Goodfellow, and Kunal Talwar. “Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=HkwoSDPgg (cited on pages 7, 68).

[Pap+18] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. “Scalable Private Learning with PATE”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=rkZB1XbRZ (cited on pages 7, 68).

[Pas+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: (2019). Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d’Alché-Buc, Emily B. Fox, and Roman Garnett, pp. 8024–8035 (cited on page 16).

[Pen+22] Sikha Pentyala, Davis Railsback, Ricardo Maia, Rafael Dowsley, David Melanson, Anderson C. A. Nascimento, and Martine De Cock. “Training Differentially Private Models with Secure Multiparty Computation”. In: *IACR Cryptol. ePrint Arch.* (2022) (cited on pages 9, 13, 14, 68).

[Ria+18] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. “Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications”. In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*. Ed. by Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim. ACM, 2018, pp. 707–721. DOI: 10.1145/3196494.3196522 (cited on page 11).

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. In: *Commun. ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342 (cited on page 4).

[RW19] Dragos Rotaru and Tim Wood. “MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security”. In: *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings*. Ed. by Feng Hao, Sushmita Ruj, and Sourav Sen Gupta. Vol. 11898. Lecture Notes in Computer Science. Springer, 2019, pp. 227–249. DOI: 10.1007/978-3-030-35423-7_12 (cited on pages 41, 78).

[Ryf19] Théo Ryffel. *Approximate models*. https://github.com/LaRiffle/approximate-models/. 2019 (cited on page 40).

[Ryf+22] Théo Ryffel, Pierre Tholoniat, David Pointcheval, and Francis R. Bach. "AriaNN: Low-Interaction Privacy-Preserving Deep Learning via Function Secret Sharing". In: *Proc. Priv. Enhancing Technol.* 2022.1 (2022), pp. 291–316. DOI: 10.2478/popets-2022-0015 (cited on pages 42, 43, 47–49).

[Ryf+18] Théo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. "A generic framework for privacy preserving deep learning". In: *CoRR* abs/1811.04017 (2018). arXiv: 1811.04017 (cited on pages 9, 11, 16, 40, 59).

[SKB22] Angelo Saadeh, Vaibhavi Kumari, and Stéphane Bressan. "Epsilon-Differentially Private and Fully Secure Logistic Regression on Vertically Split Data". In: *4th International Conference on Data Intelligence and Security, ICDIS 2022, Shenzhen, China, August 24-26, 2022*. IEEE, 2022, pp. 188–193. DOI: 10.1109/ICDIS55630.2022.00036 (cited on pages IX, 9, 13, 14, 45, 68, 82).

[SSB23] Angelo Saadeh, Pierre Senellart, and Stéphane Bressan. *Confidential Truth Finding with Multi-Party Computation (Extended Version)*. 2023. arXiv: 2305.14727 [cs.CR] (cited on pages X, 15).

[SMH07] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey E. Hinton. "Restricted Boltzmann machines for collaborative filtering". In: *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*. Ed. by Zoubin Ghahramani. Vol. 227. ACM International Conference Proceeding Series. ACM, 2007, pp. 791–798. DOI: 10.1145/1273496.1273596 (cited on pages 10, 85).

[San+04] Ashish P. Sanil, Alan F. Karr, Xiaodong Lin, and Jerome P. Reiter. "Privacy preserving regression modelling via distributed computation". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*. Ed. by Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel. ACM, 2004, pp. 677–682. DOI: 10.1145/1014052.1014139 (cited on pages 7, 14).

[Sha79] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176 (cited on page 4).

[Sho+17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. "Membership Inference Attacks Against Machine Learning Models". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 3–18. DOI: 10.1109/SP.2017.41 (cited on pages 5, 45).

[SNT07] Aleksandra B. Slavkovic, Yuval Nardi, and Matthew M. Tibbits. "Secure Logistic Regression of Horizontally and Vertically Partitioned Distributed Databases". In: *Workshops Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*. IEEE Computer Society, 2007, pp. 723–728. DOI: 10.1109/ICDMW.2007.114 (cited on pages 8, 14).

[Sri+14] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15.1 (2014), pp. 1929–1958. DOI: 10.5555/2627435.2670313. URL: https://dl.acm.org/doi/10.5555/2627435.2670313 (cited on pages 5, 14).

[Taj+20] Razane Tajeddine, Joonas Jälkö, Samuel Kaski, and Antti Honkela. "Privacy-preserving Data Sharing on Vertically Partitioned Data". In: *CoRR* abs/2010.09293 (2020). arXiv: 2010.09293 (cited on pages 7, 12, 14).

[Thi+19] Kimberley Thissen, Ir LAM Schoenmakers, Ir RP Koster, and Ir PP van Liesdonk. "Achieving Differential Privacy in Secure Multiparty Computation". PhD thesis. Eindhoven University of Technology, 2019 (cited on pages 7, 48, 68).

[WGC19] Sameer Wagh, Divya Gupta, and Nishanth Chandran. "SecureNN: 3-Party Secure Computation for Neural Network Training". In: *Proc. Priv. Enhancing Technol.* 2019.3 (2019), pp. 26–49. DOI: 10.2478/popets-2019-0035 (cited on pages 11, 12, 16, 83).

[Wag+21] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. "Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning". In: *Proc. Priv. Enhancing Technol.* 2021.1 (2021), pp. 188–208. DOI: 10.2478/popets-2021-0011 (cited on page 11).

[Wan+20] Chang Wang, Jian Liang, Mingkai Huang, Bing Bai, Kun Bai, and Hao Li. "Hybrid Differentially Private Federated Learning on Vertically Partitioned Data". In: *CoRR* abs/2009.02763 (2020). arXiv: 2009.02763 (cited on pages 7, 12, 14).

[Wan+19] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. "Adaptive Federated Learning in Resource Constrained Edge Computing Systems". In: *IEEE J. Sel. Areas Commun.* 37.6 (2019), pp. 1205–1221. DOI: 10.1109/JSAC.2019.2904348 (cited on page 6).

[Wu+16] Genqiang Wu, Yeping He, JingZheng Wu, and Xianyao Xia. "Inherit Differential Privacy in Distributed Setting: Multiparty Randomized Function Computation". In: (2016), pp. 921–928. DOI: 10.1109/TrustCom.2016.0157 (cited on pages 7, 48).

[XYW21] Depeng Xu, Shuhan Yuan, and Xintao Wu. "Achieving Differential Privacy in Vertically Partitioned Multiparty Learning". In: *2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, December 15-18, 2021*. Ed. by Yixin Chen, Heiko Ludwig, Yicheng Tu, Usama M. Fayyad, Xingquan Zhu, Xiaohua Hu, Suren Byna, Xiong Liu, Jianping Zhang, Shirui Pan, Vagelis Papalexakis, Jianwu Wang, Alfredo Cuzzocrea, and Carlos Ordonez. IEEE, 2021, pp. 5474–5483. DOI: 10.1109/BigData52589.2021.9671502 (cited on pages 7, 14).

[Xu+21] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, James Joshi, and Heiko Ludwig. "FedV: Privacy-Preserving Federated Learning over Vertically Partitioned Data". In: *AISec@CCS 2021: Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security, Virtual Event, Republic of Korea, 15 November 2021*. Ed. by Nicholas Carlini, Ambra Demontis, and Yizheng Chen. ACM, 2021, pp. 181–192. DOI: 10.1145/3474369.3486872 (cited on pages 8, 14).

[Yao82] Andrew Chi-Chih Yao. "Protocols for Secure Computations (Extended Abstract)". In: *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. IEEE Computer Society, 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38 (cited on page 4).

[Yao86] Andrew Chi-Chih Yao. "How to Generate and Exchange Secrets (Extended Abstract)". In: *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*. IEEE Computer Society, 1986, pp. 162–167. DOI: 10.1109/SFCS.1986.25 (cited on pages 4, 21).

[YHY08] Xiaoxin Yin, Jiawei Han, and Philip S. Yu. "Truth Discovery with Multiple Conflicting Information Providers on the Web". In: vol. 20. 6. 2008, pp. 796–808. DOI: 10.1109/TKDE.2007.190745 (cited on pages 9, 15, 32).

[Ypm95] Tjalling J. Ypma. "Historical Development of the Newton-Raphson Method". In: *SIAM Rev.* 37.4 (1995), pp. 531–551. DOI: 10.1137/1037125 (cited on page 40).

[Yu+22] Bin Yu, Wenjie Mao, Yihan Lv, Chen Zhang, and Yu Xie. "A survey on federated learning in data mining". In: *WIREs Data Mining Knowl. Discov.* 12.1 (2022). DOI: 10.1002/widm.1443 (cited on page 58).

[Zam21] Zama. *Concrete ML, https://github.com/zama-ai/concrete-ml*. 2021 (cited on pages 9, 11).

[Zha+17] Jiaqi Zhang, Kai Zheng, Wenlong Mou, and Liwei Wang. "Efficient Private ERM for Smooth Objectives". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. Ed. by Carles Sierra. ijcai.org, 2017, pp. 3922–3928. DOI: 10.24963/ijcai.2017/548 (cited on pages 5, 14).

[Zha+12] Jun Zhang, Zhenjie Zhang, Xiaokui Xiao, Yin Yang, and Marianne Winslett. "Functional Mechanism: Regression Analysis under Differential Privacy". In: *Proc. VLDB Endow.* 5.11 (2012), pp. 1364–1375. DOI: 10.14778/2350229.2350253. URL: http://vldb.org/pvldb/vol5/p1364%5C_junzhang%5C_vldb2012.pdf (cited on pages X, 5, 12–14, 36, 37, 46).

[Zha+22] Xiaojin Zhang, Yan Kang, Kai Chen, Lixin Fan, and Qiang Yang. "Trading Off Privacy, Utility and Efficiency in Federated Learning". In: *CoRR* abs/2209.00230 (2022). DOI: 10.48550/arXiv.2209.00230. arXiv: 2209.00230 (cited on pages 8, 14).

[ZDW18] Yifeng Zheng, Huayi Duan, and Cong Wang. "Learning the Truth Privately and Confidently: Encrypted Confidence-Aware Truth Discovery in Mobile Crowdsensing". In: *IEEE Trans. Inf. Forensics Secur.* 13.10 (2018), pp. 2475–2489. DOI: 10.1109/TIFS.2018.2819134 (cited on pages 10, 15).

[Zhe+20] Yifeng Zheng, Huayi Duan, Xingliang Yuan, and Cong Wang. "Privacy-Aware and Efficient Mobile Crowdsensing with Truth Discovery". In: *IEEE Trans. Dependable Secur. Comput.* 17.1 (2020), pp. 121–133. DOI: 10.1109/TDSC.2017.2753245 (cited on pages 10, 15).

List of Figures

List of Tables

List of Algorithms

ECOLE DOCTORALE

Titre : Les applications du calcul multipartite sécurisé en apprentissage automatique

Mots clés : Calcul multipartite sécurisé ; apprentissage automatique ; analyse de données ; confidentialité

Résumé : La préservation des données privées dans l'apprentissage automatique et l'analyse des données devient de plus en plus importante à mesure que la quantité d'informations personnelles sensibles collectées et utilisées par les organisations continue de croître. Cela pose le risque d'exposer des informations personnelles sensibles à des tiers malveillants, ce qui peut entraîner un vol d'identité, une fraude financière ou d'autres types de cybercriminalité. Les lois contre l'utilisation des données privées sont importantes pour protéger les individus contre l'utilisation et le partage de leurs informations. Cependant, ce faisant, les lois sur la protection des données limitent les applications des modèles d'apprentissage automatique, et certaines de ces applications pourraient sauver des vies, comme dans le domaine médical. Le calcul multipartite sécurisé (MPC) permet à plusieurs partis de calculer collaborativement une fonction sur leurs entrées sans avoir à révéler ou à échanger les données elles-mêmes. Cet outil peut être utilisé pour entraîner et utiliser des modèles d'apprentissage automatique collaboratif lorsqu'il existe des problèmes de confidentialité concernant l'échange d'ensembles de données sensibles entre différentes entités.
Dans cette thèse, nous (I) utilisons des algorithmes de calcul multipartite sécurisés existants et en développons de nouveaux, (II) introduisons des approximations cryptographiques des fonctions couramment utilisées en apprentissage automatique, et (III) complémentons le calcul multipartite sécurisé avec d'autres outils de confidentialité. Ce travail est effectué dans le but de mettre en œuvre des algorithmes d'apprentissage automatique et d'analyse de données préservant la confidentialité.
Notre travail et nos résultats expérimentaux montrent qu'en exécutant les algorithmes à l'aide du calcul multipartite sécurisé, la confidentialités des données est préservée et l'exactitude du résultat est satisfait. En d'autres termes, aucun parti n'a accès aux informations d'un autre et les résultats obtenus par les modèles d'apprentissage automatique et des algorithmes d'analyse de données sont les mêmes par rapport aux résultats des algorithmes exécutés sur données non chiffrés.
Dans son ensemble, cette thèse offre une vision globale du calcul multipartite sécurisé pour l'apprentissage automatique, démontrant son potentiel à révolutionner le domaine. Cette thèse contribue au déploiement et à l'acceptabilité du calcul multipartite sécurisé en apprentissage automatique et en analyse de données.

Title : Applications of secure multi-party computation in machine learning

Keywords : Secure multi-party computation ; machine learning ; data analysis ; privacy

Abstract : Privacy-preserving in machine learning and data analysis is becoming increasingly important as the amount of sensitive personal information collected and used by organizations continues to grow. This poses the risk of exposing sensitive personal information to malicious third parties - which can lead to identity theft, financial fraud, or other types of cybercrime. Laws against the use of private data are important to protect individuals from having their information used and shared. However, by doing so, data protection laws limit the applications of machine learning models, and some of these applications could be life-saving - like in the medical field. Secure multi-party computation (MPC) allows multiple parties to jointly compute a function over their inputs without having to reveal or exchange the data itself. This tool can be used for training collaborative machine learning models when there are privacy concerns about exchanging sensitive datasets between different entities.
In this thesis, we (I) use existing and develop new secure multi-party computation algorithms, (II) introduce cryptography-friendly approximations of common machine functions, and (III) complement secure multi-party computation with other privacy tools. This work is done in the goal of implementing privacy-preserving machine learning and data analysis algorithms.
Our work and experimental results show that by executing the algorithms using secure multi-party computation both security and correctness are satisfied. In other words, no party has access to another's information and they are still being able to collaboratively train machine learning models with high accuracy results, and to collaboratively evaluate data analysis algorithms in comparison with non-encrypted datasets. Overall, this thesis provides a comprehensive view of secure multi-party computation for machine learning, demonstrating its potential to revolutionize the field. This thesis contributes to the deployment and acceptability of secure multi-party computation in machine learning and data analysis.

Institut Polytechnique de Paris
91120 Palaiseau, France