Practical Privacy-Preserving Machine Learning using Fully Homomorphic Encryption

Michael Brand
School of Computing Technologies, RMIT University
Melbourne, Vic, Australia
INCERT GIE
Leudelange, Luxembourg
michael.brand@rmit.edu.au

Gaëtan Pradel*
Royal Holloway, University of London
Information Security Group
Egham, United Kingdom
INCERT GIE
Leudelange, Luxembourg
gpradel@incert.lu

ABSTRACT

Machine learning is a widely-used tool for analysing large datasets, but increasing public demand for privacy preservation and the corresponding introduction of privacy regulations have severely limited what data can be analysed, even when this analysis is for societal benefit. Homomorphic encryption, which allows computation on encrypted data, is a natural solution to this dilemma, allowing data to be analysed without sacrificing privacy. Because homomorphic encryption is computationally expensive, however, current solutions are mainly restricted to use it for inference and not training.

In this work, we present a practically viable approach to privacy-preserving machine learning training using fully homomorphic encryption. Our method achieves fast training speeds, taking less than 45 seconds to train a binary classifier over thousands of samples on a single mid-range computer, significantly outperforming state-of-the-art results.

KEYWORDS

Privacy, Fully Homomorphic Encryption, Machine Learning Training, Support Vector Machines

1 INTRODUCTION

In recent years, Machine Learning (ML) techniques have become increasingly popular for analysing and extracting insights from large and complex datasets. With the rise of powerful computing systems and the availability of vast amounts of data from various domains, ML has found applications in numerous fields such as healthcare, finance, biometric recognition, surveillance and many others. As examples, ML is used in the medical field to analyse patient data to help with disease diagnosis and treatment, while in finance, it is used for fraud detection, risk assessment, and trading strategies [2, 48].

However, the use of such techniques on sensitive data has raised significant concerns regarding data privacy [3, 36]. For example, medical data contains highly sensitive personal information and financial data includes confidential transactional details that require protection. In response, privacy-related regulations such as the *General Data Protection Regulation* (GDPR) [22] in the European Union and the *Health Insurance Portability and Accountability Act* (HIPAA) [12] in the United States of America have been enacted, which require data holders to preserve the privacy of sensitive or personally-identifiable data.

Numerous techniques have been explored for enhancing privacy in data processing and analysis. These include cryptography, differential privacy, federated learning, and the use of secure enclaves [36, 56], with many tasks requiring a combination of such tools [39, 54].

In our work, we chose to tackle the privacy challenge using *homomorphic encryption* (HE) [40]. Introduced in 1978 by Rivest et al. [52], HE enables computations to be performed on encrypted data without requiring decryption, so that the privacy of the data can be preserved throughout the entire analysis process, i.e. both while the data is at rest and in use. Specifically, our goal was to use HE in order to enable privacy in Machine Learning, by introducing practical tools that will allow the training of new Machine Learning models to be performed over encrypted data.

The idea of Privacy Preserving Machine Learning (PPML) is, itself, not new [1]. However, despite numerous notable advancements made over the past two decades (see [36] for a survey), PPML remains computationally challenging. As an example, consider recent results by Chillotti et al. [19] that presented PPML inference using the `Concrete` library [18] which implements a fast variant of the CGGI scheme [17], a fully homomorphic encryption (FHE) scheme. Despite great efforts by the designers to render FHE practical, the private inference performed by their PPML models of various complexity still impose a significant computational burden. On a standard personal computer, using 128-bit equivalent security, the computation time required for even a single FHE-encrypted inference ranged from over 100 seconds to almost 500 seconds, representing a non-trivial computational expense. Furthermore, their most complex algorithm for performing FHE-encrypted inference still exhibits degradation of more than 10% in classification accuracy, compared to inference on unencrypted data.

Moreover, similar to other related works, Chillotti et al. solely addresses the use of encrypted data for ML inference. Training is done exclusively in plaintext. This is because training ML models is a far more computationally-intensive task than inference, and performing it on encrypted data is unavoidably much slower and

*Corresponding author

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–15
© YYYY Copyright held by the owner/author(s).
https://doi.org/XXXXXXX.XXXXXXX

more resource-intensive still, rendering it impractically slow under most current approaches.

In this work, we focus exclusively on tackling the practicality of the much-heavier model training phase. We investigate training ML models on encrypted data using levelled FHE, which is a relatively light flavour of FHE computation. (See Section 2.1 for an explanation of levelled FHE and its relation to other types of HE computation.)

Levelled FHE can be used whenever the desired algorithm's multiplicative depth can be bounded in advance. We show how such bounding is always possible to do in the context of PPML training, and how we were able to design our algorithms so as to minimise the computational depth required for such training.

To demonstrate our technique in a simple ML context, we selected the *Support Vector Machine* (SVM) learning algorithm as our underlying ML model to be trained.

SVM is a widely used algorithm for classifying data in machine learning due to its ability to handle complex datasets and produce robust predictions.

Here, we show how to train a linear SVM, although the same basic technique can also be applied to more complex models.[1]

In order to demonstrate practically feasible model training under FHE, overcoming the computational challenges involved, we employ a well-established technique known as the *client-assisted* (CA) computation model. The existing approaches for CA are reviewed in Section 1.3, and our own flavour of it is explained in Section 1.4.

In this paper, we show that CA computation can be used to both significantly enhance the speed of PPML and to expand the scope of what can be performed under levelled FHE, to the extent that training machine learning models over encrypted data becomes practical and scalable.

1.1 Problem statement

1.1.1 Motivating example. Consider the following scenario from the medical world, in which a research institute wants to develop a new tool for diagnosing a particular condition, a tool which the research institute then intends to sell to hospitals. The hospitals are expected to perform various tests in order to measure multiple patient attributes, and the tool will use the test results in order to determine the likelihood that the patient is suffering from the condition in question. This will offer the hospitals better diagnoses, and enable them to be more judicious in the choice of follow-up tests, thus eliminating more invasive testing where it is superfluous and improving patient care.

For this, the research institute wants to develop a new ML model that will drive the diagnosis tool. To train such a model, the research institute would optimally need access to the widest possible patient population. For each patient, the research institute would need both the patient's initial test results and their final diagnosis. The test results will form the attributes fed into the model, and the final diagnosis will be the model's target variable.

Hospitals have access to all this information. However, due to privacy regulations, the hospitals cannot share it with the research institute.

[1]During research, we also trained our SVM with non-linear features, but these did not improve overall accuracy, so we do not report the results here.

The patients themselves have access to their own information, but while they may entrust their own health providers with handling it, not all of them will wish to entrust the research institute with the same data.

One can imagine here three potential populations of patients. The first is the patients who do not want to assist the research institute in building its models. For them, privacy regulation ensures that the data is theirs, and cannot be used without their consent. The second is the patients who feel free to provide their data (including both test results and final diagnosis) to the research institute, for the purpose of developing the model. This is the group that traditionally models were built on, and, in our scenario, can still be used, e.g., as a verification group and to provide various types of sanity-checking for the newly-developed model.

We believe that this still leaves out, however, a large portion of the population. This is namely the population of patients who do wish to assist in the development of the new diagnosis tool, but do not wish to expose for this their personal details. They wish their details to remain private, and wish their participation in the model training population to remain anonymous.

It is this third group that our research results aim to cater for. Our PPML paradigm allows these patients to contribute their data to potentially-life-saving research, without having to sacrifice for this their privacy. Similar instances of this problem, which can be tackled with our methods, appear also beyond medical research, wherever data analysis is done with the ultimate goal of powering societal benefits. Allowing this third group of participants into such models will enable models to be trained on larger training sets, and ones less biased by participant self-selection.

1.1.2 Privacy model. The motivating example above dictates the privacy model that our solution must work under. This privacy model is somewhat more nuanced than typically used in cryptography, in that parties are not simply considered "trusted" or "untrusted". Rather, each of the actors in our model is considered trusted in certain dimensions by certain other actors.

To describe the privacy model explicitly, let us first introduce the relevant actors.

Client: This is the research institute in our motivating example. It is the party building the new model.

Model users: These are the hospitals in our example. They ultimately receive the model and use it. In our example, hospitals use the model in order to diagnose their own new patients, after the model is fully developed.

Data Owners: These are the patients in our motivating example. They own their own data, which the model is then trained on.

Cloud: This final actor is introduced here for the first time. It was not mentioned in our motivating example because this actor is not part of our problem scenario. Instead, it is part of our solution design. The Cloud performs the majority of the processing in computing the new model. In our example, this role would presumably be taken by a cloud service provider hired by the research institute for this purpose.

Because the data being processed in this problem scenario belongs to the Data Owners, they are the party that determines, for the most part, who is trusted to what. Specifically, the trust mapping of the various actors is as follows.

Client: The Data Owners are willing to have their data used for the purpose of building a new ML model. The end model will be visible to the Client, and the Data Owners are equally willing to share with the Client also other non-identifiable statistics for this purpose. As a general rule, if a value is the sum of a particular statistic, computed symmetrically over the data of each Data Owner, it is usually considered non-identifiable, and permitted for the Client to see. The Client must not be exposed to personal data, however, including both personal attributes (e.g., test results), and the value of the target variable for each participant. In particular, the Client does not know who is participating in their model build.

Model users: The Model Users receive the model only once it is fully constructed. It is at this point no longer considered privacy sensitive from a Data Owner point of view. Model users use the derived model in plaintext: when a new patient is treated at a hospital, the treating doctors are expected to know both their test results and any diagnoses in order to provide optimal treatment. (New patients will consent to this, as is the case in any medical treatment.)

Data Owners: Data Owners are clearly exposed to their own data. However, Data Owners have no trust relationship with any other Data Owner. Whether a person participates as a Data Owner in the construction of any new diagnosis tool, and what data they contribute, is entirely their own private information.

Cloud: The trust model for the Cloud is the most nuanced. Data Owners do not trust the Cloud with any of their personal data, and not even with general statistics about their data. However, for the Cloud to be able to process Data Owners' data, the Data Owners must trust the Cloud in two important ways. First, Data Owners (and by extension also the rest of this data ecosystem) must trust the Cloud to truthfully process their data as per the instructions given to the Cloud by the Client. The Cloud is expected to not maliciously plant bad data into the model. Second, Data Owners must trust the Cloud not to collude with the Client against the wishes of the Data Owners. In both cases, this is on a par with the trust that one normally extends to cloud hosting services, and underpins much of the viability of the Internet economy at large.

As can be seen, in this problem scenario it is only model training that requires a privacy-preserving implementation. Once the model is trained, it can be viewed in plaintext by the Client, which allows the Client to package it as efficiently-running software for large-scale deployment. It will be run on plaintext data, and will return plaintext results. It is still possible that the Client, when packaging the model in software, may choose to use various software, hardware and/or legal protections in order to protect the Client's investment and prevent the model from being, e.g., exploited by commercial competitors, but any such protection is beyond the scope of our present work.

1.2 General solution design

Homomorphic encryption typically involves the generation of a key triplet. This key triplet is composed of a *secret key* used for decryption, a *public key* for encryption, and an *evaluation key* for manipulating the encrypted data such as by performing on it arithmetic operations (but not enabling encryption or decryption).

In our general solution design, which addresses the above problem statement, it is the Client who generates the key triplet. While the secret key is kept solely by the Client, the public key is made public (and by this reaches any Data Owners who may wish to participate in the model build), and the evaluation key is shared with the Cloud. This is the only case of direct communication between the Client and the Data Owners.

Data Owners who wish to participate can now do so by encrypting their data and sending it directly to the Cloud for processing. Because the Cloud has no access to the decryption key, their privacy is preserved.[2]

In principle, the Cloud could have at this point performed all processing under homomorphic encryption, sending only the final output to the Client, for the Client to decrypt. In our solution, however, we show that far better results can be obtained by use of a client-assisted computation model. In this model, the Cloud can communicate information to and from the Client. We show that despite the fact that all such communication is required to be encrypted, privacy non-sensitive general statistics, it can still be used effectively to speed up the work of the Cloud.

1.3 Related work

1.3.1 Privacy during inference. Cryptographic privacy-enhancing technologies (PETS) are many, including, e.g., such tools as functional encryption and garbled circuits [56]. The following is a review of literature related specifically to HE-powered PPML, which we use to introduce some of the techniques we have built upon. Within this literature, it is notable that several works employ distributed computation in conjunction with HE to maintain privacy.

We begin by presenting the research efforts that concentrate solely on the inference phase of PPML, which employ FHE for encryption without incorporating distributed computation.

A pioneering work in this field was Gilad-Baschrach et al.'s CryptoNets [24], from 2016. Here, a neural network model that was previously trained on clear data was shown to be usable for prediction over new encrypted inputs, by using FHE encryption for the data and performing the prediction under this encryption. The solution was applied on the MNIST dataset [34], demonstrating a 99% accuracy for a throughput of $59,000$ predictions per hour.

One of the main challenges that Gilad-Baschrach et al. had to address in order to use HE in neural networks is that all homomorphic operations that are enabled even by FHE are polynomial evaluations on the encrypted inputs. This makes it difficult to compute in the encrypted domain even such common functions as taking the square root of a number, comparing numbers, or even testing for equality. [11].

[2]Technologies that hide, e.g., Data Owners' IP addresses when such data is uploaded to the Cloud are beyond the scope of the present paper, but can certainly be used to ensure that Data Owners do not need to reveal any of their private identifiers to the Cloud for this purpose.

Unfortunately, such functions form a fundamental part of neural networks, e.g. appearing in typical neuron activation functions [44].

The solution used by CryptoNets was *polynomial substitution*. Namely, the (non-polynomial) sigmoid activation function of the target neural network was replaced by squaring, a simple-to-compute, low-degree polynomial function.

An alternative method to compute a non-polynomial function over encrypted data in HE-powered PPML is to prepare in advance a look-up table for the function over a discrete set of values. A solution of this type was presented by Crawford et al. [21].

Another commonly used technique, which is the one we applied in our work, is low-degree polynomial approximation. This has been used, e.g., by Chabanne et al. [13] to extend CryptoNets. Their neural network architecture was implemented with a ReLU activation function that was approximated using a polynomial function to support computation over encrypted data. In this paper, we present a novel method that enabled us to approximate the ReLU function significantly more accurately, however.

An important component in effective use of low-degree polynomial approximations is *batch normalisation*, a technique that aligns the distribution of inputs to the approximated function, thus guarding against function evaluations outside the well-approximated domain. This, too, is a technique employed by Chabanne et al. that our paper incorporates as well.

The authors achieved a model with minimal accuracy degradation between private and non-private inference. However, no timing results were reported.

In 2019, Hesamifard et al. [28] presented another work based on the use of polynomial approximations of the ReLU function. A range of approximation techniques was evaluated in the paper, including the application of numerical approximation, the Taylor series method, Chebyshev polynomials (in both a standard and a modified form) and an approach based on the derivative of the ReLU function. They found that the more successful approach is the latter, which is based on the observation that the derivative of the ReLU function is a step function, and that the sigmoid function can be used as a smooth approximation to it. Thus, they approximated the sigmoid function with a polynomial and then integrated it to obtain an approximation of the ReLU function. By contrast, our method is based on approximating the "second derivative" of the ReLU function using Chebyshev polynomials, and then integrating it twice. Despite using only a low-degree polynomial approximation, Hesamifard et al. achieved 99% accurary on the MNIST dataset, and were able to run at a throughput of $164,000$ predictions per hour, outperforming CryptoNets, even with Chabanne et al.'s extensions.

In 2022, Lee et al. [35] introduced privacy-preserving ML inference that uses RNS-CKKS [15], a bootstrapped variant of the CKKS scheme [16]. CKKS was itself an important milestone in PPML, in that it was the first HE scheme that could perform floating point computation, thus being practically applicable for defining ML algorithms. While the accuracy of the model on encrypted data was high, at three hours per classification it was still much too slow to be practically usable, highlighting the limitations of using unqualified, non-levelled FHE schemes for ML.

Similar to the works on neural networks, PPML inference using FHE has also been implemented over SVM models [6, 30]. Most recently, Badawi et al. [5] employed CKKS with 128-bit equivalent security parameters for SVM prediction. The authors reached an execution time of 1.25 seconds per prediction on a multi-core CPU platform without accuracy degradation due to the use of FHE.

All solutions presented so far, despite tackling only the inference phase on encrypted data, remain relatively slow. As discussed, one option is to partner HE and distributed computation in order to enhance the overall performance and get closer to real-world applicability. While this can be done via standard parallelisation methods, other alternatives, with their own advantages and disadvantages, exist.

An example is *Secure Multi-Party Computation* (MPC) [25]. MPC methods rely on the separation of computation in a way that each party individually is not exposed to private information, but a collusion of all parties breaks any privacy preservation. Such methods are typically characterised by much lighter computations, as compared with FHE methods, but heavy communication loads.

MPC methods were first typically restricted to only work under *partial homomorphic encryption* (PHE) schemes. These are schemes that support only one type of group operation on encrypted data, are not universal in their computation capabilities, and have been shown to not be resilient against quantum attacks [8]. PHE is discussed in greater detail in Section 2.

The earliest works combining HE with MPC are Barni et al. [7] and Orlandi et al. [45], who managed to execute the inference phase of a small neural network in around 10 seconds. These performance results were obtained thanks to an interactive protocol involving a server and a client, and utilise an additive PHE scheme.

Even without MPC, the use of PHE instead of FHE has been a popular avenue for obtaining speed increases, and this due to the significantly more lightweight nature of PHE operations. Solutions such as MiniONN [37], Chameleon [51] and Gazelle [32] all introduced increasingly performant methods compared to their predecessors. However, all were ultimately limited by their reliance on PHE.

The use of FHE becomes necessary for computing more complex operations over encrypted data, or when quantum resilience is an issue: most FHE schemes used in the literature are quantum-safe [4].

In 2020, Boemer et al. [9] presented a hybrid FHE-MPC framework called MP2ML for ML inference in the encrypted domain using the CKKS scheme, extending the nGraph-HE compiler [10]. They obtained similar results compared to state-of-the-art work but with stronger security properties.

1.3.2 Privacy during model training. All work presented so far addresses only the inference phase of ML on encrypted data and does not cover the training phase. The training of ML models poses greater computational challenges than the inference phase, and only few works have endeavoured to address it.

In 2019, Nandakumar et al. [43] showed that it is possible to train a neural network on encrypted data without any form of distributed computation. They used the HElib library [27] to implement a Stochastic Gradient Descent (SGD)-based training of a neural network, and demonstrated that the computational efficiency of encrypted training can be improved using several techniques such as simplifying the network, selecting an appropriate data representation, and optimising data packing within the ciphertexts.

The results of Vizitiu et al. [59] are presently the best in class in terms of speed. They performed both training and inference of deep learning-based applications on encrypted data also without relying on either MPC or CA. A customized version of the MORE encryption scheme [33] was employed by the authors, which was specifically designed to operate on floating-point data in order to accommodate ML applications. In this way, they were able to obtain the same accuracy as in the unencrypted domain at the cost of a 30-fold increase in training time (which is a remarkably small factor). However, their solution comes with some caveats: the MORE encryption scheme is subject to some security concerns [57, 58] which do not apply to the CKKS scheme, the algorithm we use in this work.

Nevertheless, most works in the field use either MPC or CA, and attain by this results that are an order of magnitude faster than those of Nandakumar et al..

One such work is SecureML by Mohassel and Zhang [41]. They showed how to perform ML training and inference in a privacy-preserving way in a two-party computation model, i.e. one where Data Owners share their sensitive data in encrypted form among two non-colluding servers that share the computational work. In their study, the authors achieved model training times on the order of several thousand seconds in offline settings, and hundreds of seconds online. The difference between their work and our approach is two-fold. First, they did not use FHE, as we do, but rather a combination of garbled circuits and PHE. Second, they use MPC rather than the CA model.

In 2018, Hesamifard et al. [29] presented CryptoDL, a solution for both PPML training and inference, using FHE. This solution is much closer to our approach in that it employs levelled FHE, is built over CA computation instead of MPC, only uses client-server communication when necessary (rather than at each operation, as SecureML does), and relies on polynomial approximations. They used a more sophisticated neural network than us, which they tested on three small datasets. However, they used only 80-bit equivalent security, which enabled them to utilise much faster homomorphic operations, and they parallelised model training over the cores of a twelve-cores machine, allowing them significantly more CPU power. Even so, our 128-bit equivalent results obtained on a dataset with almost double the number of features were still 55% faster (when accounting for the dataset size differences), and showed better accuracy.

Park et al. [47] presented for the first time SVM training using FHE. (See also [46].) They used the CKKS scheme and its SIMD architecture in their implementation, and performed PPML training on various datasets, including the Wisconsin Breast Cancer (WBC) dataset [60], which we also trained on. Although their approach was similar to ours, we were able to achieve significantly better performance, reducing per-iteration times by a factor of 4.6 and reaching convergence in 40% less iterations, despite working on more features and attaining better model accuracy.

1.4 Contributions

We present a practically viable PPML training method that uses FHE in the CA computational model.

The method reaches practical speeds in running privacy-preserving model training: less than 45 seconds were necessary to train a binary classifier on $8,000$ samples, each with 30 numerical features, when running on a single computer (we used a Dell XPS 15 laptop and an Ubuntu workstation), without any use of parallelisation or GPU acceleration. Training complexity is linear in the number of features and the number of ciphertexts necessary to hold all samples. In our design, one ciphertext vector can hold 8,192 samples, so only a single vector was required for each feature.

When training on a public medical dataset, the WBC dataset, we show that such training on encrypted data yields a model with 99% accuracy. (As per our problem statement, actually running the classification under encryption was not a goal of this project, so this 99% was evaluated in plaintext on a held out set.)

To obtain our results, we incorporated a set of carefully chosen techniques and design considerations that prioritise pragmatic concerns related to computational efficiency and processing capacity:

State-of-the-art cryptography: We opted for the levelled version of the CKKS FHE scheme. CKKS competes with state-of-the-art time-performance [31] and is able to process floating-point data directly.

Vectorised computations: CKKS operates in a SIMD architecture, allowing for the processing of multiple operands in a single instruction. Our algorithm has been optimised with SIMD operations in mind, further improving overall performance.

Client-assisted computational model: We carefully partitioned the computation so that all "heavy lifting" is done by the Cloud, which mass-processes data under encryption. Flow control and the computation of run-state parameters, however, are all relegated to the Client. In this way, not only does each party in the computation lean on its strengths, the overall communication between them is also minimised.

Frugality in the polynomial approximations: We used only low-degree polynomial approximations of the non-linear functions necessary for the machine learning training. This allowed us to work with low CKKS parameters that support only a small multiplication-depth but translate to fast FHE operations and small-sized ciphertexts, without compromising on security strength. (Our parameters were set for 128-bit equivalent security.) A novel approximation technique using Chebyshev polynomials allowed us to obtain sufficiently accurate approximations even under the degree constraint.

Normalisation: Polynomial approximations might only be valid on a certain interval. To keep the data within the target interval and mitigate precision issues and overflows, we used a batch normalisation technique on the data.

Application of rapid convergence methods: We used a restricted version of Newton iterations to speed-up the convergence of our training algorithm, in order to be able to work with less algorithm iterations and obtain an overall-faster implementation. This is as opposed to gradient descent or SGD methods.

The selection and use of these techniques and methods serve as the principal contributions of this paper, providing a blueprint for developing practical PPML training through FHE within the client-assisted computational model.

1.5 Outline of the paper

The rest of this paper is organised as follows.

Section 2 introduces the necessary notions in homomorphic encryption and machine learning. Section 3 presents the technical details of our approach. Section 4 describes our experimental setup and results. Lastly, a short conclusions section follows.

2 PRELIMINARIES

2.1 Homomorphic encryption

Homomorphic encryption is a type of encryption that enables computations over encrypted data without ever decrypting it. It has been considered since 1978 [52] as the Holy Grail of cryptography [26].

The first HE schemes, such as RSA [53], allowed only a single type of operation, typically either addition or multiplication, to be computed on encrypted data. Such systems are known as *partially homomorphic encryption* (PHE) schemes. While holding the promise for enabling practical computation on encrypted data [49], PHE schemes are inherently limited. A PHE scheme that only supports addition and subtraction, for example, can only ever compute linear functions over its inputs, which is insufficient for general-purpose computing, or even for the training of the simplest of ML models.

To address the limitations of PHE schemes and support Turing-complete computation over encrypted data, HE schemes have been developed that support full ring operations (i.e., both additions and multiplications) over encrypted data. Such schemes are called *fully homomorphic encryption* (FHE) schemes.

Many FHE schemes support any number of additions but only computations of up to a predefined multiplication depth. These are known as *levelled* FHE schemes. Despite this apparent restriction, levelled FHE schemes demonstrate great performance, particularly when compared to unqualified FHE schemes that support unlimited computations over encrypted data.

Such unlimited FHE schemes were pioneered by Gentry [23] in 2009, through a technique known as *bootstrapping*. The fundamental concept behind bootstrapping is that the decryption algorithm for a given ciphertext can be executed as an algorithm under levelled FHE. This enables the ciphertext to be "refreshed", effectively removing the limitation on what can be computed over encrypted data, but at the cost of transforming each homomorphic operation into a heavy bootstrapping process.

For the most part, the availability of an unlimited number of multiplication operations is not necessary: if one knows in advance what computation one is attempting, one only needs to set the levelled-FHE parameters so as to allow the desired number of multiplications, and this usually results in order-of-magnitude time savings.

In our implementation, we broke up the computation of Newton iterations so that each homomorphic computation only needs to calculate one step of the iteration. As a result, the total needed computation depth is known in advance, regardless of the number of iterations. We were consequently able to utilise fast, levelled-homomorphic operations.

Specifically, we use the CKKS FHE scheme in its levelled version. CKKS is a new type of FHE scheme, considered as the precursor of the "fourth-generation" [40] of FHE. The security of this scheme is based on the hardness of the Ring variant of the Learning With Errors (RLWE) problem [38, 50]. RLWE has been proven to be at least as hard as certain worst-case lattice problems. These problems are considered to be computationally hard problems for both classical and quantum computers.

Unlike its predecessors, CKKS is suited for non-integer computation, supporting approximate arithmetic over encrypted real and complex numbers: its arithmetic is performed on block floating point [42] numbers. This crucial property makes CKKS a highly suitable option for implementing ML algorithms, which typically work with floating-point data.

In addition to its natural compatibility with the floating-point domain, the CKKS scheme incorporates a SIMD architecture. This allows the scheme to perform computations on multiple data points at once, greatly enhancing its processing speed and efficiency for computations that can be formulated as vectorised operations.

2.2 Machine learning

Supervised learning is a process in which ML models are trained to map vectors of input *features* into output *labels* based on a *training set* that contains examples of features-label pairs, known as *samples*. During training, the ML algorithm learns from the labelled samples a *model* that can predict the output label for new, unseen input feature vectors. The process of training ML models involves finding a set of parameters that optimises (in some sense) the performance of the model on the training data. Once the model has been trained, it can be applied to new, unseen data to make predictions about the output label.

Training such an ML model typically involves defining a *loss* function that measures the difference between the predicted output of the model and the true output, and then using an *optimisation algorithm* to minimise this loss. Such a loss function may also encapsulate other factors. For example, it may include penalties, known as *regularisation terms*, that are meant to keep the model from overfitting its training data (thus reducing its usefulness on new data), e.g. by penalising models for their complexity.

To test the usefulness of the model on new data and validate that it did not overfit its training data, the labelled dataset available is commonly separated into a *training set* and a *test set*. The main idea is then to minimise the ML model's loss over the training set and then evaluate how well it performs on the test set.

Even though classification accuracy can be a simple binary "correct" or "incorrect", the loss functions used in ML training are usually smooth, or at least differentiable, functions. This allows for efficient optimisation using gradient-based methods, such as *gradient descent*. In the absence of local optima, gradient-based methods can be used to find the optimal model parameters, even for general models.

Viewed in this context, our work can be seen as an exploration into efficient methods to perform function optimisation under HE, with general applicability across many ML model types.

The specific classifier that we use in this paper as an example is the linear binary classifier. This can be viewed as a linear Support Vector Machine (SVM) [20], although it can also be viewed as the training of a perceptron, which is the simplest example of a neural network.

In all cases, the purpose of training is to find a vector of weights, $(w_1, \cdots, w_n)$, and a bias, b, such that given a vector of inputs $(x_1, \cdots, x_n)$, the sign of

$$\sum_{i=1}^{n} x_i w_i + b$$

best predicts the class label associated with the input vector.

In vector notation, the equation for the separating hyperplane can be written as

$$\mathbf{x} \cdot \mathbf{w} + b = 0, \tag{1}$$

where $\mathbf{w}$ is the normal to the hyperplane.

3 PPML TRAINING

Let $x_{ij} \in \mathbb{R}$, with $i \in [1, m]$ and $j \in [1, \ell]$, be the value of feature j of the i^{th} training sample. These values will be arranged in a matrix $\mathbf{x}$, such that $\mathbf{x}_{i*}$ is a row vector of the entire i^{th} sample and $\mathbf{x}_{*j}$ is a column vector of the j^{th} feature. Let $\mathbf{y}$, a column vector of length m, be the target variable, encoding for each sample a class designation as either 1 or -1.

We use the following notation to describe the algorithm's execution in the CA model: values that are computed by the Client and sent to the Cloud will be denoted in double brackets ("$[\![x]\!]$"). All such communication will be in ciphertext, each value transmitted in its own, separate ciphertext, and each such ciphertext storing the communicated value across every one of its elements.

When elements of ciphertext vectors that are stored by the Cloud are summed together by the Cloud and sent to the Client, this will be denoted by the function "**SUM**()". Such summation is performed by Algorithm 1. In the algorithm, "$|v|$" indicates the element length of the vector v, and "$s \ll i$" indicates cyclic shift.

Algorithm 1: Vector summation algorithm.

```
Input  : A ciphertext vector, $v$.
Output : Ciphertext vector $s$, where each element contains
         $\sum_i v_i$.
$s \leftarrow v$
$i \leftarrow 1$
while $i < |v|$ do
    $s \leftarrow s + (s \ll i)$
    $i \leftarrow 2i$
return $s$
```

In our computing model, this is the only type of information that is allowed to be communicated from the Cloud to the Client. If the number of training samples exceeds what can be packed into a single vector, all vectors are first summed together, and then Algorithm 1 is run on the result. The output value, s, is returned to the Client as a ciphertext vector, and the Client decrypts it and extracts its value (from any element).

Operations outside brackets, meaning that they are executed by the Cloud, can only be linear operations. In order to visually distinguish variables that are stored by the Cloud, we use for these names that feature a hat notation (e.g., "$\hat{x}$"). The values m and ℓ, which are the only values known to the Cloud in plaintext, do not receive a hat notation.

If a computation involves no brackets at all, and therefore no communication, it can refer to a standalone computation either by the Cloud or by the Client. The two can be distinguished based on whether or not the result features a hat notation. As an example, if Algorithm 1 itself was written in this notation (rather than merely acting to explain the **SUM**() operator), all variables in it would have had hats in their names, because it is an algorithm only run by the Cloud.

The only other values used in our system are those that are pre-computed by the Data Owners prior to being uploaded to the Cloud. Each Data Owner i sends to the Cloud the following, each number encrypted in its own ciphertext:

- $-y_i$, and
- For each $j \in [1, \ell]$:
  - x_{ij},
  - x_{ij}^2, and
  - $x_{ij}y_i$.

Where these values which were communicated by the Data Owners are used in a formula, they are enclosed in braces ("{}").

Unlike communications from the Client, when a Data Owner sends a ciphetext vector to the Cloud, this vector includes the encrypted value in only its first element. When the Cloud uploads these, it uses cyclic shifting to map each Data Owner's information to its own unique position, then sums together vectors that underwent different shifts into a single vector, packing as many values as possible, from different Data Owners, that are all of the same value type.

We refer to this as *column-oriented storage.* It enables efficient use of CKKS's SIMD architecture for both intra-vector and inter-vector operations.

3.1 Normalisation

Because the vast majority of the computation in this algorithm is done over encrypted data, it is important throughout the algorithm to keep information from becoming too large (and susceptible to overflows) or too small (and susceptible to loss of precision). We account for this in the algorithm in two places. First, when the Data Owners upload their data, they do so in units of measurement that are appropriate, so that data values come at a consistent order of magnitude. This order of magnitude does not need to be too precisely prescribed, because all such values are immediately standardised when uploaded to the Cloud, so as to have zero mean and unit variance. Second, at each iteration of the algorithm, the Client communicates to the Cloud a scaling factor, γ, such that computations are effectively done on γx, rather than on x.

The first thing the algorithm needs to compute is therefore the mean and the variance for each feature, in order to remove them from the data. This is done as described in Algorithm 2. This algorithm removes the mean from the data in the Cloud, but only computes the standard deviation at the Client. It would have been possible to completely standardise the data by multiplying it by $1/\sigma$, but this would have cost us an extra multiplication. Instead, the multiplicative factor γ, which is applied on the data each iteration, already incorporates the σ factor when communicated from the Client to the Cloud.

Algorithm 2: Calculate the variance and remove the mean.

```
Input  : A feature vector, $\mathbf{x}_{*j}$.
Output : Mean-removed $\hat{\mathbf{x}}_{*j}$ and standard deviation $\sigma$.
$\mu \leftarrow \mathbf{SUM}\left(\{\mathbf{x}_{*j}\}\right)/m$
$\sigma \leftarrow \sqrt{\frac{\mathbf{SUM}\left(\{\mathbf{x}_{*j}^2\}\right)/m - [\![\mu^2]\!]}{m-1}}$
$\hat{\mathbf{x}}_{*j} \leftarrow \{\mathbf{x}_{*j}\} - [\![\mu]\!]$
return $\hat{\mathbf{x}}_{*j}, \sigma$
```

The purpose of multiplying by a factor γ at each iteration, beyond the issue of maintaining the integrity of the encrypted value as described above, is to allow us to apply on the data a non-linear function. In order to compute, for example, $h(x)$, where h is the hinge function, $h(x) = \max(x, 0)$, we replace h by a polynomial approximation, $\tilde{h}$, but any such polynomial approximation is only valid within a specific range. We have designed all our polynomial approximations to be accurate within the range $-1 \leq x \leq 1$. As a result, we continuously rescale our data to make sure that all of it is exactly within this range.

We try to maximise the spread of x values within the allowed range, because our approximations are also most effective away from zero. However, because within a single optimisation iteration values are modified incrementally, we actually scale to within the range $[-(1-\eta), 1-\eta]$, keeping a margin of η so as to allow breathing room for this. The value of η was set in our experiments at 0.2.

The calculation of the maximum $|x|$ value, in order to allow for the scaling, is also approximate, and is described in Algorithm 3. We use $k = 3$, which allows fairly accurate estimation of the maximum without running the risk of overflows, and is well within our multiplication budget.

Algorithm 3: Calculate the approximate maximum, softmax($\mathbf{x}_{*j}$).

```
Input  : A feature vector, $\mathbf{x}_{*j}$.
Output : The approximate $\max_i |\mathbf{x}_{ij}|$.
$\hat{t} \leftarrow \mathbf{x}_{*j}^2$
for $i \leftarrow 2$ to $k$ do
    $\hat{t} \leftarrow \hat{t}^2$
$softmax \leftarrow \mathbf{SUM}\left(\hat{t}\right)^{2^{-k}}$
return $softmax$
```

The initial choice of γ is done before the Client can have enough information from the Cloud in order to make sure that softmax can be calculated safely, so instead of Algorithm 3 we scale using the previously calculated standard deviation, σ, so that the range $[-(1-\eta), 1-\eta]$ covers s standard deviations, for a predefined s, dependent only on m. We chose s to be 3 in our experiments, because, given our m, if the features are normally distributed, this scaling ensures that less than one sample in expectation will exceed the designated range.

There is still a risk that some samples will slightly exceed the range, but not enough to cause problematic overflows, and slight deviations are handled by the use of an even-degree approximating polynomial for our loss function, which ensures that in the process of optimisation the optimising algorithm would steer to avoid such cases.

3.2 The training algorithm

We use an iterative training algorithm whose purpose is to find the classifier parameters $(\mathbf{w}, b)$ that minimise the loss function

$$L(\mathbf{w}, b) = \mathbf{SUM}_i\left(\tilde{h}(\{-y_i\}(\mathbf{x}_{i*} \cdot [\![\mathbf{w}]\!] + [\![b]\!]))\right). \tag{2}$$

This is an approximated form of hinge loss, which is a common loss function for training the perceptron algorithm. In our case, because the approximation is least accurate around zero, and smoothes over the function's transition to zero, it also introduces a built-in margin, so this loss can equally be thought of as the minimisation of slack variables in SVM optimisation.

This optimisation is subject to $\|\mathbf{w}\| = 1$, meaning that the $\mathbf{w}$ is a unit vector indicating the normal to the decision boundary hyperplane. Without this constraint, it would have been possible to improve the loss simply by scaling $\mathbf{w}$.

We note that the entire loss function $L(\mathbf{w}, b)$ can be computed with a multiplication depth of only 2 plus the multiplication depth of $\tilde{h}$. In our implementation, we use a $\tilde{h}$ polynomial of degree 14, which we compute using a multiplicative depth of 4.

Because hinge loss is a convex loss function (when not constraining the size of $\mathbf{w}$), it is well suited for iterative, gradient-based algorithms. Two such popular algorithms are gradient descent and Newton iterations. Gradient descent shifts the parameters in the direction of maximal improvement, but requires one to set a learning rate, α, for it. The iterations then become

$$(\mathbf{w}_{t+1}, b_{t+1}) \leftarrow (\mathbf{w}_t, b_t) - \alpha \nabla L(\mathbf{w}, b).$$

A too-small choice for α would lead to slow convergence, whereas a choice that is too large would cause the algorithm to overshoot the optimum and potentially lose stability.

Newton iterations, on the other hand, do not require such a manually set learning rate parameter, but do so at significant computational cost. A Newton iteration locally approximates the target function as a hyperboloid and sets the next iteration to the optimal point on the hyperboloid. It has been shown that when such iterations are within their basin of convergence, convergence is exponentially fast. However, computing the parameters of the hyperboloid requires a second derivative, i.e. a Hessian, which is an $\ell \times \ell$ matrix, so requires an amount of effort quadratic in the number of features.

We opted to use a third method, combining the best of both gradient descent and Newton iterations. Namely, at each iteration, we first compute the gradient, and then perform a Newton optimisation iteration just in the direction of the gradient. In other words, we approximate the loss function along the line of the gradient as a parabola, and set our new values to the optimum of that parabola. This algorithm, one-dimensional Newton iterations, can be thought of as a form of gradient descent where the α parameter is computed automatically. The complete algorithm is shown in Algorithm 4.

In Algorithm 4, "$\cdot$" indicates matrix multiplication, "$\times$" indicates element by element multiplication, and function application on a vector works also element by element. For the purposes of matrix

```
Algorithm 4: A Newton iteration.
  Input   : Input parameters $(\mathbf{w}_t, b_t)$.
  Output  : Next iteration parameters $(\mathbf{w}_{t+1}, b_{t+1})$.
  $\hat{\mathbf{c}} \leftarrow \{-\mathbf{y}\} \times (\hat{\mathbf{x}} \cdot [\![\mathbf{w}_t]\!] + [\![b_t]\!])$
  $\hat{\mathbf{d}} \leftarrow \tilde{h}'(\hat{\mathbf{c}})$
  for $j \leftarrow 1$ to $\ell$ do
      $\Delta w_j \leftarrow \mathbf{SUM}_i\left(\hat{d}_i\left(\{-y_i\}\hat{x}_{ij}\right)\right)$
  $\Delta b \leftarrow \mathbf{SUM}_i\left(\hat{d}_i\{-y_i\}\right)$
  $p \leftarrow \mathbf{\Delta w}^T \cdot \mathbf{w}_t$
  $\mathbf{\Delta w} \leftarrow \mathbf{\Delta w} - p\mathbf{w}_t$
  $D \leftarrow (\Delta b)^2 + \|\mathbf{\Delta w}\|^2$
  $H \leftarrow \mathbf{SUM}_i\left(\tilde{h}''(\hat{c}_i)(\hat{x}_{i*} \cdot [\![\mathbf{\Delta w}]\!] + [\![\Delta b]\!])^2\right)$
  $\alpha \leftarrow D/H$
  $(\mathbf{w}_{t+1}, b_{t+1}) \leftarrow (\mathbf{w}_t, b_t) - \alpha(\mathbf{\Delta w}, \Delta b)$
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_{t+1}/\|\mathbf{w}_{t+1}\|$
  return $(\mathbf{w}_{t+1}, b_{t+1})$
```

multiplication, the vectors $\mathbf{w}_t$, $\mathbf{w}_{t+1}$ and $\mathbf{\Delta w}$ are all taken to be column vectors. They are all of length ℓ.

In the algorithm, Δw_j acts as the partial derivative $\frac{\partial L(\mathbf{w},b)}{\partial w_j}$ while Δb acts as $\frac{\partial L(\mathbf{w},b)}{\partial b}$. We remove from $\mathbf{\Delta w}$, however, its component parallel to the $\mathbf{w}_t$ vector, in order for small steps in this direction to not change the size of the weight vector, thus improving the stability of the algorithm.

The polynomials $\tilde{h}'$ and $\tilde{h}''$ are the first and second derivatives of $\tilde{h}$, our approximation for the hinge function, and were computed analytically from $\tilde{h}$. The computation of all 3 functions was done ahead of time, and involves no communication.

The values D and H are the first and second derivatives in the direction of the gradient, so setting α as D/H is simply a standard Newton iteration in one dimension.

Newton iterations, because of their ability to perform large steps, are at risk of diverging when not near an optimum, for which reason in our implementation following this step the Client checks whether there is an improvement in the loss function. A deterioration in the loss function leads to halving α. However, this condition was never triggered in our training, and the algorithm converged after only 6–7 iterations even from a random initialisation. (And even faster when using the initialisation recommended in Section 3.3.)

We note that throughout this entire process the Cloud at no point needed to perform a computation with a multiplicative depth of more than 3 plus the depth of the polynomial. (Two multiplications are needed in order to compute $\hat{\mathbf{c}}$, the input to the polynomial, and one more in order to multiply the polynomial's output with any additional factors in a final multiplication step.) Scaling, in particular, does not increase the multiplicative depth of the algorithm: when scaling $\{-y_i\}(\hat{\mathbf{x}}_{i*} \cdot [\![\mathbf{w}_t]\!] + [\![b_t]\!])$ by a factor γ in order to use the result as an input to a function, we do this simply by having the Client scale the values $\mathbf{w}_t$ and b_t prior to communicating them. Where the output of the function needs to be scaled as well, this is done after the summation by the Cloud, by the Client, in plaintext, so once again does not incur any additional ciphertext multiplications.

3.3 Initialisation

Before algorithm iterations can commence, we must choose an initial parameter set to start with. There are many ways to do this. We have opted to set the bias b to zero and the initial weights w_j to be proportional to the covariance between the target labels and the j^{th} feature. If the feature noise is independent between features, this initialisation produces predictions that are least affected by noise.

The covariance is computed using the Data Owner-precalculated $x_{ij}y_i$ values. The specifics are almost identical to the way feature variance was computed in Algorithm 2. We do not repeat the description here.

3.4 Approximating h

The final part of the algorithm, which we have not discussed yet, is how to approximate the hinge function using a polynomial approximation with a small multiplication depth. In fact, instead of computing $\tilde{h}$ directly, we chose to approximate $\tilde{h}''$, or a function proportional to $\tilde{h}''$, instead, and compute $\tilde{h}$ by doubly integrating the result, scaling, and adding an affine function (this affine function being a degree of freedom in the integration). The scaling factor and the linear component of the affine function were chosen so that the negative portion of the function will be as flat as possible while the positive part will have a slope of 1. The optimisation algorithm itself is invariant to the choice of the additive constant, so in the actual implementation we used for it zero. This choice made it easier to ignore unused vector positions when summing function output values across an entire ciphertext vector. However, in order to demonstrate of the quality of the approximation, it is possible to substitute this zero by the additive constant that minimises the Root Mean Square Error (RMSE) in approximating the hinge function.

Figure 1 shows the plots of the hinge function h, the raw result of doubly integrating $\tilde{h}''$ and scaling, I, and our final polynomial approximation of $\tilde{h}$ after adding the appropriate affine function, including the additive constant that minimises its RMSE. The numbers in Table 2, which showcase the significant improvement in approximation accuracy afforded by our method, were computed using this same constant.

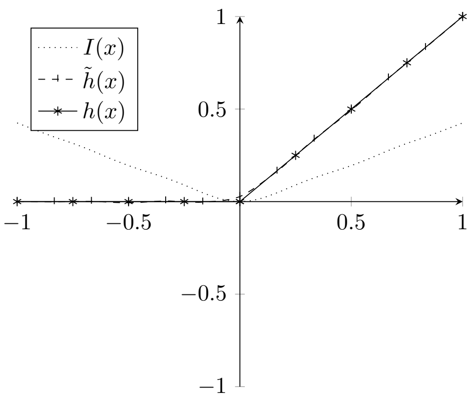


Figure 1: Plot of I, $\tilde{h}$ and h.

We note that while the additive constant used in the approximation is arbitrary, a good choice of the linear coefficient and scaling factor is critical. Hinge loss penalises the model when the predicted score is on the wrong side of the decision boundary (i.e., $y_i(\mathbf{x_i} \cdot \mathbf{w} + b) < 0$). When the predicted score is correct (i.e., $y_i(\mathbf{x_i} \cdot \mathbf{w} + b) > 0$), the loss is 0. Thus, correct classifications do not impact the optimisation algorithm which is merely trying to improve the incorrect classifications (i.e., to minimise the slack variables). As optimisation progresses, there should be more and more correctly classified pairs, and the misclassified pairs become a small minority. As a result, it is important for the correctly classified majority to not bias the optimisation and by this prevent proper handling of the remaining incorrectly classified samples. This was done by considering the smallest and largest positive zeros of the original (unscaled) I polynomial, and scaling the function so that the slope between them is exactly $1/2$. A linear coefficient of $1/2$ then aligns the corresponding negative zeroes to the same height in the final $\tilde{h}''$, while adjusting the positive slope to 1.

It remains to show how to approximate $\tilde{h}''$. In the case of the original hinge function h, the first derivative is a Heaviside step function and the second derivative is a Dirac δ function. To approximate this, we wanted to choose a polynomial that in the range $\epsilon \leq |x| \leq 1$ never exceeds some absolute value α, but at $x = 0$ reaches 1.

A well-known family of polynomials whose absolute values are bounded within the range $[-1, 1]$ is the family of Chebyshev polynomials. These are easy-to-compute polynomials, all of whose roots lie within the $[-1, 1]$ range, and all of whose maxima and minima reach a height of 1 and -1, respectively. At -1 and 1, the polynomials also reach the same height.

Consider now, for the Chebyshev polynomial T_n of some even degree n, the polynomial

$$Q(x) = \alpha T_n\left(s - (s+1)x^2\right),$$

where $s = \left(1+\epsilon^2\right)/\left(1-\epsilon^2\right)$. This polynomial stays within $[-\alpha, \alpha]$ in the entire range where $-1 \leq s - (s+1)x^2 \leq 1$, i.e. where $\sqrt{(s-1)/(s+1)} \leq |x| \leq 1$ holds. This is exactly the desired range $\epsilon \leq |x| \leq 1$. For each ϵ, there is an α for which $Q(0) = 1$. This presents a trade-off between our choice of ϵ (the range of the approximation) and α (the accuracy of the approximation). In our implementation we used $\alpha = 1/4$.

In the literature, polynomial approximations for non-linear functions are computed using either Taylor coefficients or Chebyshev coefficients, the latter of which is meant to reduce inaccuracies near the edges of the approximation range, this effect being known as Runge's Phenomenon. Our approximation method, however, using Chebyshev polynomials to directly constrain the maximal divergence of the approximation, has allowed us to reach superior results.

4 EXPERIMENTAL IMPLEMENTATION

4.1 Parameters

The results presented here are based on executions on a single machine with 32 GBs RAM, running an Intel(R) Core(TM) i7-8700 CPU at 3.20GHz, under the Ubuntu 21.10 operating system. We used the Wisconsin Breast Cancer (WBC) dataset [60], a public medical dataset, to generate our PPML model. This dataset contains real data on breast cancer tumours for 569 patients. The dataset is composed of 30 numerical features that describe measured attributes of each tumour, and one binary attribute indicating the final diagnosis of whether the tumour was subsequently designated 'malignant' or 'benign'. The final diagnosis, being our target variable, was mapped to 1 for 'benign' and -1 for 'malignant'.

Although WBC is a relatively small dataset, it was suitable for our proof-of-concept implementation. Our purpose was to demonstrate the feasibility of our approach, not to create a production system.[3]

In fact, because of the small number of samples in WBC, the vast majority of our ciphertext vectors remains unused when learning WBC in its native form, so at first glance our timing results may appear suboptimal. However, in follow-up experiments, we have supplemented WBC with synthetic repetitions of the same training samples, and our results show that even learning on 8,000 samples does not increase run-times. Furthermore, the algorithmic complexity of our homomorphic operations is invariant to the number of samples, so increasing the training set size has no effect on our choice of encryption parameters.

For a CKKS implementation, we used the SEAL open-source library [55] by Microsoft implemented in C/C++, which we accessed through the Python wrapper `SEAL-Python` [14].

We chose our parameters so as to provide 128-bit equivalent security and to allow a multiplicative depth of at least 7 (being 4 for the polynomial calculation and 3 for the Newton iterations). The minimal choice of such parameters, which led to the fastest execution times and smallest ciphertexts, allowed for a multiplicative depth of 8, thereby providing ample computational headroom for employing more complex algorithms. However, we did not use the extra headroom for the results presented here. The resulting ciphertext vectors when using our chosen parameters possessed a length of 8,192 ciphertext elements, which is why training execution time remains invariant even with the inclusion of additional samples.

4.2 Results

4.2.1 Execution times. We separated our PPML training process into three phases:

Encryption: Encryption of the samples,

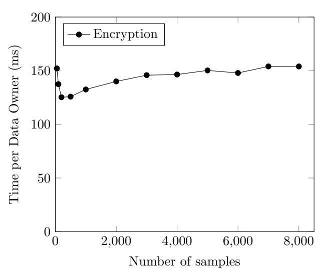
Data Ingest: Uploading the encrypted samples to the Cloud, and the samples' subsequent preprocessing and scaling, and

Training: Initialisation and iterations of the Newton-based algorithm.

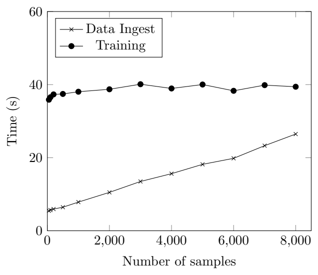
As is common practice, we partitioned the full WBC dataset into training and testing sets, with the former comprising 80% and the latter 20% of the data.

Three types of actors participate in the training phase: the Data Owners, the Client, and the Cloud. Encryption of the samples is performed by the Data Owners, and is, in a real-world application, a massively parallelised process in which each Data Owner only encrypts their own data. The other steps are performed jointly by

[3]We acknowledge that a linear SVM, even supplemented with non-linear features, is unable to handle more complex datasets such as the MNIST dataset. In follow-up work, we expand our solution to enable the learning of general models, which allow classification on full-sized, high-complexity datasets.

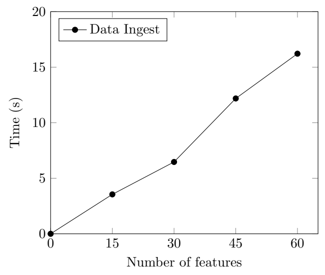


(a) Encryption execution time per Data Owner by number of samples.

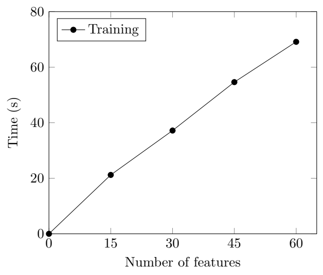


(b) Data Ingest and Training execution time by number of samples.

Figure 2: Execution time as a function of sample count.



(a) Data Ingest execution time by number of features.



(b) Training execution time by number of features.

Figure 3: Execution time as a function of feature count.

the Client and the Cloud, and Data Owners do not participate in them at all. The computational load for performing this algorithm should therefore be analysed from the perspective of the Data Owners and the Client separately (noting that the Client, in a real-world setting, is likely to be paying for the computational load of the Cloud as well). From the perspective of the Data Owners, only the time it takes to encrypt each Data Owner's own data is important, and we analyse this in terms of the time it takes to encrypt a single new sample. From the perspective of the Client, encryption time is immaterial, but the entirety of the rest of the process should be measured; we therefore analyse this portion of the training seperately to encryption. For completeness, we also include in Table 1 the full summed-up processing times, where total encryption time is given. In this view, encryption takes up the majority of our CPU time, although in practice this is unlikely to be a concern. Without encryption, our PPML training algorithm completes in under 45 seconds.

Table 1: PPML training execution times. Total execution time is given without ("w/o") and with ("w") encryption.

Phase	Execution time (s)
Encryption	56.70
Data Ingest	6.448
Model initialisation	2.194
Model optimisation	34.99
Total time (w/o)	**43.632**
Total time (w)	100.333

We investigated the scalability of our PPML training algorithm in relation to variations in the number of samples and features. For this, we executed the complete algorithm, including data encryption, with up to 8,000 samples and 60 features.

Figure 2 displays execution times as a function of the number of samples. This is separated by actor. Figure 2(a) displays the average execution time for a single Data Owner; Figure 2(b) shows combined execution times for the Client and Cloud.

These results demonstrate that, regardless of the number of Data Owners involved, computation time for each Data Owner remains constant at around 150 milliseconds. Training remains invariant with respect to the number of samples up to the point of saturating the ciphertext vector. However, any increase in the number of samples beyond the prescribed limit of 8,192 (as determined by our SEAL parameters) results in additional ciphertext vectors i.e. in a linear growth of the Training phase, and is fully parallelisable.

The results also show a linear growth in the length of the Data Ingest phase. This growth is only due to the uploading of the encrypted data by the Cloud. Once data is uploaded and packed into ciphertext vectors, its total processing time depends only on the number of ciphertext vectors, and therefore, as before, remains constant up to 8,192 samples.

Figure 3 illustrates the execution times in the Data Ingest and Training phases for a varying number of features.

As previously described, in our implementation we chose a column-oriented arrangement: we used the entire length of our ciphertext vectors to store as many data samples as we have, each vector housing the values of only a single feature, in order to effectively exploit the SIMD architecture inherent in the CKKS algorithm. As a result, execution times of the training phase exhibit a linear growth with the number of features, as depicted in Figure 3(b).

The Data Ingest phase grows linearly in both the number of features and the number of samples. The contribution of each feature to the computation time is approximately 250 milliseconds in the Data Ingest phase (for the total sample size of the WBC dataset) and approximately 1.2 seconds in the Training phase.

4.2.2 Accuracy. The efficacy of our PPML model is inherently dependent on the effectiveness of our polynomial approximation technique, which we utilise in order to compute the hinge function over encrypted data. Table 2 compares the quality of our polynomial approximation to related works and confirms the effectiveness of our approach. The results are due both to our approximation method and to the fact that our efficient use of multiplications elsewhere in the PPML training algorithm enabled us to incorporate here a full multiplicative-depth 4 polynomial approximation.

Using our polynomial approximation, we obtained an accuracy of 99.1% on our (decrypted) trained model over held-out data. For comparison, we trained the same model on clear data using common methods and obtained exactly the same accuracy results, proving the viability of our methods.

Figure 4 presents a confusion matrix of the results of our PPML-trained model, when applied over clear data.

Table 2: Comparison between related works and ours on polynomial approximation in the $[-1, 1]$ domain.

Work	Method	Function	Multiplicative-depth	RMSE	Final model accuracy (%)
[13]	Taylor series	ReLU	3	0.056	97.91
[29]	Orthogonal systems of polynomials	Sigmoid	1	N/A	96.3
[28]	Derivative and Chebyshev	ReLU	1	0.13	98.52
Ours	**Second derivative and Chebyshev**	**ReLU/hinge**	**4**	**0.0050**	**99.1**

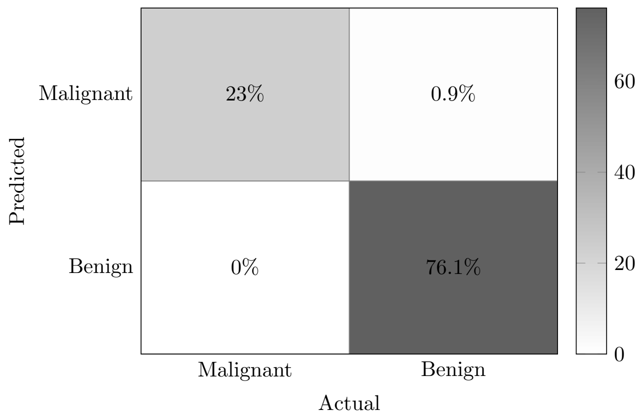


Figure 4: Confusion matrix of our PPML model.

4.2.3 Comparison with related work. Park et al.'s [47] is the work most comparable to ours and warrants a comprehensive comparison. This is given in Table 3.

Table 3: Comparison between Park et al. and our work.

Work	Number of used features	Number of iterations	Time per iteration (s)	Training (s)	Model final accuracy (%)
Park et al. [47]	9	10	33.545	335.45	97–98
Ours	**30**	**6**	**7.272**	**43.632**	**99.1**

5 CONCLUSION AND FUTURE WORK

In this paper, we presented a novel PPML training method using FHE, which achieves state-of-the-art processing speeds without sacrificing accuracy. Specifically, we trained a linear SVM classifier on 8, 000 samples, each containing 30 numerical features in under 45 seconds (not including the initial encryption) on a single computer, without any GPU acceleration, and attained 99% accuracy, which is a state-of-the-art result for the dataset used.

To achieve these results, we applied a collection of carefully selected techniques and design considerations that prioritise pragmatic concerns relating to computational efficiency and processing capacity. These include FHE over floating-point data, vectorised and parallelised computations, the client-assisted computation model, minimal communication between the client and the cloud, frugal polynomial approximations, and rapid convergence methods.

We propose that our techniques and methods can serve as a blueprint for real-world training of machine-learning models under the constraint of privacy-preservation, in settings where a client-assisted computation model can be used.

In ongoing research, we are expanding our solution to support generic algorithms, applicable for significantly larger datasets.

ACKNOWLEDGMENTS

The authors acknowledge INCERT for the funding of the project.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 2000. Privacy-Preserving Data Mining. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein (Eds.). ACM, 439–450. https://doi.org/10.1145/342009.335438

[2] Shamima Ahmed, Muneer M. Alshater, Anis El Ammari, and Helmi Hammami. 2022. Artificial intelligence and machine learning in finance: A bibliometric review. *Research in International Business and Finance* 61 (2022), 101646. https://doi.org/10.1016/j.ribaf.2022.101646

[3] Mohammad Al-Rubaie and J. Morris Chang. 2019. Privacy-Preserving Machine Learning: Threats and Solutions. *IEEE Secur. Priv.* 17, 2 (2019), 49–58. https://doi.org/10.1109/MSEC.2018.2888775

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

[5] Ahmad Al Badawi, Ling Chen, and Saru Vig. 2022. Fast homomorphic SVM inference on encrypted data. *Neural Comput. Appl.* 34, 18 (2022), 15555–15573. https://doi.org/10.1007/s00521-022-07202-8

[6] Jean-Claude Bajard, Paulo Martins, Leonel Sousa, and Vincent Zucca. 2020. Improving the Efficiency of SVM Classification With FHE. *IEEE Trans. Inf. Forensics Secur.* 15 (2020), 1709–1722. https://doi.org/10.1109/TIFS.2019.2946097

[7] Mauro Barni, Claudio Orlandi, and Alessandro Piva. 2006. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia & Security, MM&Sec 2006, Geneva, Switzerland, September 26-27, 2006*, Sviatoslav Voloshynovskiy, Jana Dittmann, and Jessica J. Fridrich (Eds.). ACM, 146–151. https://doi.org/10.1145/1161366.1161393

[8] Daniel J. Bernstein and Tanja Lange. 2017. Post-quantum cryptography. *Nat.* 549, 7671 (2017), 188–194. https://doi.org/10.1038/nature23461

[9] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. 2020. MP2ML: a mixed-protocol machine learning framework for private inference. In *ARES 2020: The 15th International Conference on Availability, Reliability and Security, Virtual Event, Ireland, August 25-28, 2020*, Melanie Volkamer and Christian Wressnegger (Eds.). ACM, 14:1–14:10. https://doi.org/10.1145/3407023.3407045

[10] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. 2019. nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers, CF 2019, Alghero, Italy, April 30 - May 2, 2019*, Francesca Palumbo, Michela Becchi, Martin Schulz, and Kento Sato (Eds.). ACM, 3–13. https://doi.org/10.1145/3310273.3323047

[11] Florian Bourse, Olivier Sanders, and Jacques Traoré. 2020. Improved Secure Integer Comparison via Homomorphic Encryption. In *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12006)*, Stanislaw Jarecki (Ed.). Springer, 391–416. https://doi.org/10.1007/978-3-030-40186-3_17

[12] Centers for Medicare & Medicaid Services. 1996. The Health Insurance Portability and Accountability Act of 1996 (HIPAA). Online at http://www.cms.hhs.gov/hipaa/.

[13] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. 2017. Privacy-Preserving Classification on Deep Neural Network. *IACR Cryptol. ePrint Arch.* (2017), 35. http://eprint.iacr.org/2017/035

[14] Zhigang Chen. 2022. SEAL-Python (release 4.0). https://github.com/Huelse/SEAL-Python.

[15] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. 2018. A Full RNS Variant of Approximate Homomorphic Encryption. In *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 11349)*, Carlos Cid and Michael J. Jacobson Jr. (Eds.). Springer, 347–368. https://doi.org/10.1007/978-3-030-10970-7_16

[16] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, 409–437. https://doi.org/10.1007/978-3-319-70694-8_15

[17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *J. Cryptol.* 33, 1 (2020), 34–91. https://doi.org/10.1007/s00145-019-09319-x

[18] Ilaria Chillotti, Marc Joye, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2020. CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TfhE. In *WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, Vol. 15.

[19] Ilaria Chillotti, Marc Joye, and Pascal Paillier. 2021. Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. In *Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8-9, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12716)*, Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander A. Schwarzmann (Eds.). Springer, 1–19. https://doi.org/10.1007/978-3-030-78086-9_1

[20] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (1995), 273–297. https://doi.org/10.1007/BF00994018

[21] Jack L. H. Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. 2018. Doing Real Work with FHE: The Case of Logistic Regression. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, WAHC@CCS 2018, Toronto, ON, Canada, October 19, 2018*, Michael Brenner and Kurt Rohloff (Eds.). ACM, 1–12. https://doi.org/10.1145/3267973.3267974

[22] European Commission. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance). https://eur-lex.europa.eu/eli/reg/2016/679/oj

[23] Craig Gentry. 2009. *A fully homomorphic encryption scheme.* Ph. D. Dissertation. Stanford University. crypto.stanford.edu/craig.

[24] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, 201–210. http://proceedings.mlr.press/v48/gilad-bachrach16.html

[25] Oded Goldreich. 2004. *The Foundations of Cryptography - Volume 2: Basic Applications.* Cambridge University Press.

[26] Shai Halevi. 2017. Homomorphic Encryption. In *Tutorials on the Foundations of Cryptography.*, Yehuda Lindell (Ed.). Springer International Publishing, 219–276. https://doi.org/10.1007/978-3-319-57048-8_5

[27] Shai Halevi and Victor Shoup. 2014. Algorithms in HElib. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 8616)*, Juan A. Garay and Rosario Gennaro (Eds.). Springer, 554–571. https://doi.org/10.1007/978-3-662-44371-2_31

[28] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2019. Deep Neural Networks Classification over Encrypted Data. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY 2019, Richardson, TX, USA, March 25-27, 2019*, Gail-Joon Ahn, Bhavani Thuraisingham, Murat Kantarcioglu, and Ram Krishnan (Eds.). ACM, 97–108. https://doi.org/10.1145/3292006.3300044

[29] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N. Wright. 2018. Privacy-preserving Machine Learning as a Service. *Proc. Priv. Enhancing Technol.* 2018, 3 (2018), 123–142. https://doi.org/10.1515/popets-2018-0024

[30] Hai Huang, Yongjian Wang, and Haoran Zong. 2022. Support vector machine classification over encrypted data. *Appl. Intell.* 52, 6 (2022), 5938–5948. https://doi.org/10.1007/s10489-021-02727-2

[31] Lei Jiang and Lei Ju. 2022. FHEBench: Benchmarking Fully Homomorphic Encryption Schemes. *CoRR* abs/2203.00728 (2022). https://doi.org/10.48550/arXiv.2203.00728 arXiv:2203.00728

[32] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha P. Chandrakasan. 2018. 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*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 1651–1669. https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[33] Aviad Kipnis and Eliphaz Hibshoosh. 2012. Efficient Methods for Practical Fully Homomorphic Symmetric-key Encrypton, Randomization and Verification. *IACR Cryptol. ePrint Arch.* (2012), 637. http://eprint.iacr.org/2012/637

[34] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. 1998. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/. (1998).

[35] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. 2022. Privacy-Preserving Machine Learning With Fully Homomorphic Encryption for Deep Neural Network. *IEEE Access* 10 (2022), 30039–30054. https://doi.org/10.1109/ACCESS.2022.3159694

[36] Bo Liu, Ming Ding, Sina Shaham, Wenny Rahayu, Farhad Farokhi, and Zihuai Lin. 2022. When Machine Learning Meets Privacy: A Survey and Outlook. *ACM Comput. Surv.* 54, 2 (2022), 31:1–31:36. https://doi.org/10.1145/3436755

[37] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. 2017. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 619–631. https://doi.org/10.1145/3133956.3134056

[38] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 6110)*. Springer, 1–23.

[39] Mohamad Mansouri, Melek Önen, Wafa Ben Jaballah, and Mauro Conti. 2023. SoK: Secure Aggregation Based on Cryptographic Schemes for Federated Learning. *Proc. Priv. Enhancing Technol.* 2023, 1 (2023), 140–157. https://doi.org/10.56553/popets-2023-0009

[40] Chiara Marcolla, Victor Sucasas, Marc Manzano, Riccardo Bassoli, Frank H. P. Fitzek, and Najwa Aaraj. 2022. Survey on Fully Homomorphic Encryption, Theory, and Applications. *Proc. IEEE* 110, 10 (2022), 1572–1609. https://doi.org/10.1109/JPROC.2022.3205665

[41] Payman Mohassel and Yupeng Zhang. 2017. 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, 19–38. https://doi.org/10.1109/SP.2017.12

[42] Jean-Michel Muller, Nicolas Brunie, Florent de Dinechin, Claude-Pierre Jeannerod, Mioara Joldes, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, and Serge Torres. 2018. *Handbook of Floating-Point Arithmetic (2nd Ed.).* Springer. https://doi.org/10.1007/978-3-319-76526-6

[43] Karthik Nandakumar, Nalini K. Ratha, Sharath Pankanti, and Shai Halevi. 2019. Towards Deep Neural Network Training on Encrypted Data. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019.* Computer Vision Foundation / IEEE, 40–48. https://doi.org/10.1109/CVPRW.2019.00011

[44] Srinath Obla, Xinghan Gong, Asma Aloufi, Peizhao Hu, and Daniel Takabi. 2020. Effective Activation Functions for Homomorphic Evaluation of Deep Neural Networks. *IEEE Access* 8 (2020), 153098–153112. https://doi.org/10.1109/ACCESS.2020.3017436

[45] Claudio Orlandi, Alessandro Piva, and Mauro Barni. 2007. Oblivious Neural Network Computing via Homomorphic Encryption. *EURASIP J. Inf. Secur.* 2007 (2007). https://doi.org/10.1155/2007/37343

[46] Saerom Park, Junyoung Byun, and Joohee Lee. 2022. Privacy-Preserving Fair Learning of Support Vector Machine with Homomorphic Encryption. In *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini (Eds.). ACM, 3572–3583. https://doi.org/10.1145/3485447.3512252

[47] Saerom Park, Junyoung Byun, Joohee Lee, Jung Hee Cheon, and Jaewook Lee. 2020. HE-Friendly Algorithm for Privacy-Preserving SVM Training. *IEEE Access* 8 (2020), 57414–57425. https://doi.org/10.1109/ACCESS.2020.2981818

[48] Adnan Qayyum, Junaid Qadir, Muhammad Bilal, and Ala Al-Fuqaha. 2021. Secure and Robust Machine Learning for Healthcare: A Survey. *IEEE Reviews in Biomedical Engineering* 14 (2021), 156–180. https://doi.org/10.1109/RBME.2020.3013489

[49] Jean Louis Raisaro, Jeffrey G Klann, Kavishwar B Wagholikar, Hossein Estiri, Jean-Pierre Hubaux, and Shawn N Murphy. 2018. Feasibility of Homomorphic Encryption for Sharing I2B2 Aggregate-Level Data in the Cloud. *AMIA Summits on Translational Science Proceedings* 2017 (may 2018), 176–185. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5961814/

[50] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, Harold N. Gabow and Ronald Fagin (Eds.). ACM, 84–93. https://doi.org/10.1145/1060590.1060603

[51] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. 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*, Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim (Eds.). ACM, 707–721. https://doi.org/10.1145/3196494.3196522

[52] Ronald L. Rivest, M. L. Dertouzos, and Leonard M. Adleman. 1978. On data banks and privacy homomorphisms. *Fondations of Secure Computation, Academia Press* (1978), 169–179.

[53] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.

[54] Sinem Sav, Apostolos Pyrgelis, Juan Ramón Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. 2021. POSEIDON: Privacy-Preserving Federated Neural Network Learning. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/poseidon-privacy-preserving-federated-neural-network-learning/

[55] SEAL 2022. Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[56] Harry Chandra Tanuwidjaja, Rakyong Choi, Seunggeun Baek, and Kwangjo Kim. 2020. Privacy-Preserving Deep Learning on Machine Learning as a Service - a Comprehensive Survey. *IEEE Access* 8 (2020), 167425–167447. https://doi.org/10.1109/ACCESS.2020.3023084

[57] Boaz Tsaban and Noam Lifshitz. 2015. Cryptanalysis of the MORE symmetric key fully homomorphic encryption scheme. *J. Math. Cryptol.* 9, 2 (2015), 75–78. https://doi.org/10.1515/jmc-2014-0013

[58] Serge Vaudenay and Damian Vizár. 2015. Cryptanalysis of chosen symmetric homomorphic schemes. *STUDIA SCIENTIARUM MATHEMATICARUM HUNGARICA* 52, 2 (2015), 288–306. https://doi.org/10.1556/012.2015.52.2.1311

[59] Anamaria Vizitiu, Cosmin Ioan Nita, Andrei Puiu, Constantin Suciu, and Lucian Mihai Itu. 2020. Applying Deep Neural Networks over Homomorphic Encrypted Medical Data. *Comput. Math. Methods Medicine* 2020 (2020), 3910250:1–3910250:26. https://doi.org/10.1155/2020/3910250

[60] William Wolberg, Nick Street, and Olvi Mangasarian. 1995. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. http://archive.ics.uci.edu/ml http://archive.ics.uci.edu/ml.

A ALGORITHMS

We used Algorithm 5 to generate the Chebyshev polynomial of degree n.

Algorithm 5: Function `chebyshev` that computes Chebyshev T_n

```
Input  : Degree $n$
Output : Chebyshev polynomial $T_n$ of degree $n$
$T_0 \leftarrow 1$
$T_1 \leftarrow x$
for $i \leftarrow 2$ to $n$ do
    $T_i \leftarrow 2xT_{i-1} - T_{i-2}$
return $T_n$
```

The algorithm for computing $\tilde{h}''$ is given in Algorithm 6.

Algorithm 6: Function `approx_dirac` that approximates the Dirac function

```
Input  : A degree $n$ and $\beta \in \mathbb{R}$
Output : A polynomial $Q(x) \approx h''$ and $s \in \mathbb{R}$
if $n \not\equiv 0 \bmod 2$ then
    Stop: the degree must be even.
$T_{n/2} \leftarrow$ chebyshev($n/2$)
$s \leftarrow$ seek_beta($T_{n/2}, \beta$)
$Q(x) \leftarrow \beta^{-1} T_{n/2}(s - (s + 1.0)x^2)$
return $(Q(x), s)$
```

The solution to the equation $T_{n/2}(s) = \beta$, used in Algorithm 6, is generated using the function $\texttt{seek}_\beta$ defined by Algorithm 7, which is essentially a binary search.

The algorithm for approximating of the hinge loss is given in Algorithm 8.

B SOLUTION ILLUSTRATION

Figure 5 presents a design overview of our solution described in Section 1.2 including the various tasks achieved by the various actors during the training of the PPML model based on FHE, using the terminology of our motivating example in Section 1.1.1.

Algorithm 7: Function `seek_beta`

```
Input  : A polynomial $P(x)$ of degree $n$ and $\beta \in \mathbb{R}$
Output : $s \in \mathbb{R}$ such that $P(s) = \beta$
if $\beta \leq 1$ then
    Stop
$\epsilon \leftarrow 10^{-14}$
$x_{min} \leftarrow 1.0$
$x_{max} \leftarrow 1.1$
while $P(x_{max}) < \beta$ do
    $x_{max} \leftarrow 2x_{max} - 1.0$
while $x_{max} - x_{min} > \epsilon$ do
    $x_{mid} \leftarrow (x_{max} + x_{min}) * 0.5$
    $y_{mid} \leftarrow P(x_{mid})$
    if $y_{mid} > \beta$ then
        $x_{max} \leftarrow x_{mid}$
    else
        $x_{min} \leftarrow x_{mid}$
return $x_{max}$
```

Algorithm 8: Function `approx_hinge` that approximates the hinge loss

```
Input  : A degree $n$ and $\alpha \in \mathbb{R}$
Output : A polynomial approximation $\tilde{h}$ of the hinge loss
$(Q(x), s) \leftarrow$ approx_dirac($n - 2, \alpha^{-1}$)
$\tilde{h} \leftarrow \iint Q(x)$
$m \leftarrow \frac{n-2}{2}$
$k_{min} \leftarrow 0$
$k_{max} \leftarrow m - 1$
$x_{min} \leftarrow \sqrt{s - \frac{\cos\left(\frac{0.5\pi + \pi k_{min}}{m}\right)}{s+1.0}}$
$x_{max} \leftarrow \sqrt{s - \frac{\cos\left(\frac{0.5\pi + \pi k_{max}}{m}\right)}{1+1.0}}$
$y_{min} \leftarrow \tilde{h}(x_{min})$
$y_{max} \leftarrow \tilde{h}(x_{max})$
$a \leftarrow \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$
$\tilde{h} \leftarrow \tilde{h} \times \frac{a}{2}$
$\tilde{h} \leftarrow \tilde{h} + \frac{x}{2}$
return $\tilde{h}$
```

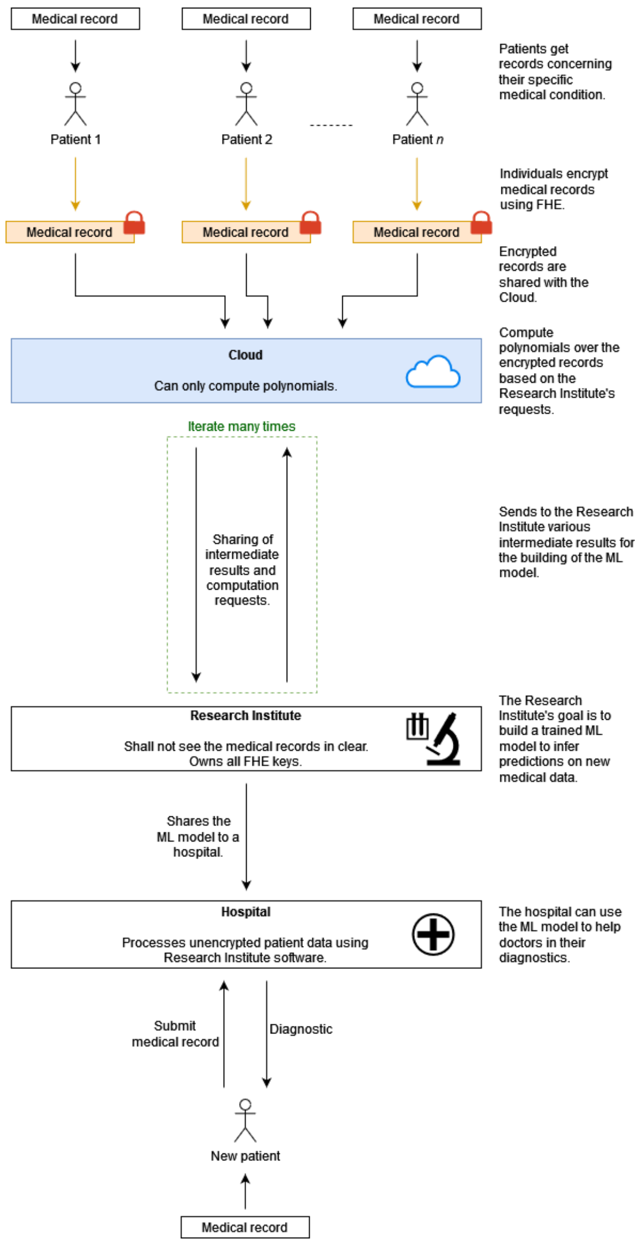

Medical record
Medical record
Medical record
Patients get records concerning their specific medical condition.
Patient 1
Patient 2
Patient *n*
Individuals encrypt medical records using FHE.
Medical record
Medical record
Medical record
Encrypted records are shared with the Cloud.
Cloud
Can only compute polynomials.
Compute polynomials over the encrypted records based on the Research Institute's requests.
Iterate many times
Sharing of intermediate results and computation requests.
Sends to the Research Institute various intermediate results for the building of the ML model.
Research Institute
Shall not see the medical records in clear.
Owns all FHE keys.
The Research Institute's goal is to build a trained ML model to infer predictions on new medical data.
Shares the ML model to a hospital.
Hospital
Processes unencrypted patient data using Research Institute software.
The hospital can use the ML model to help doctors in their diagnostics.
Submit medical record
Diagnostic
New patient
Medical record

Figure 5: Illustration of our solution design.