Training and Test Machine Learning Models on Encrypted Data: Initial Results and Challenges

Rodrigo Kruger, Jean Paul Barddal, and Vinicius M. A. Souza

Programa de Pós-Graduação em Informática (PPGIa),
Pontifícia Universidade Católica do Paraná (PUCPR)
{rodkruger, jean.barddal, vinicius}@ppgia.pucpr.br

Abstract. Privacy is critical when using Machine Learning (ML) models over sensitive data, like healthcare, finance, and legal systems. Many of these models are trained or executed on cloud services, meaning sensitive data is transmitted over the network, or third-party services operate directly on unprotected data during training and inference, increasing exposure to potential leaks. Data encryption is a promising solution that guarantees high privacy levels. An adequate cryptography solution for ML is Homomorphic Encryption, a cryptographic method that allows mathematical operations on ciphertexts, i.e., encrypted data, producing encrypted models and outputs that only authorized parties can decrypt. However, the protection offered by Homomorphic Encryption comes at a significant computational overhead. Additionally, only specific mathematical operations (typically additions and multiplications) are allowed, and encrypted computations accumulate noise that reduces the result's precision. This paper discusses the challenges of using encrypted data in training and test steps of ML models. It experimentally analyzes the impact on error rates and processing times when traditional classifiers, such as Artificial Neural Network and Logistic Regression, are adapted to process encrypted data. We adopt the CKKS scheme, a Homomorphic Encryption method that supports approximate computations over real numbers and adapted the activation functions of the classifiers using three approximation methods in an experimental evaluation with five medical datasets.

Keywords: Cryptography · Homomorphic Encryption · Classification · Artificial Neural Networks · Logistic Regression · Privacy Preserving.

1 Introduction

The increasing deployment of Machine Learning (ML) models in domains involving sensitive data, such as healthcare, finance, and legal systems, raises fundamental concerns about privacy, security, and regulatory compliance. Many of these models are trained or executed using cloud services, which means that sensitive data is transmitted over the network, or third-party services operate directly on unprotected data during training and inference, increasing exposure to potential leaks.

The emergence of *Privacy-Preserving Machine Learning* (PPML) [2] addresses the need for ML models capable of preserving data privacy in their different steps, from training to inference. Among the various approaches, *Homomorphic Encryption* (HE) [14] provides a particularly strong privacy guarantee. It allows computations (e.g., weight updates or gradient calculations) to be performed directly over encrypted inputs, producing encrypted outputs (e.g., predicted labels) that can be decrypted only by authorized parties. In contrast to other paradigms such as *Differential Privacy* (DP) [13] and *Federated Learning* (FL) [23], which rely on trust in local devices or aggregation mechanisms, HE maintains end-to-end cryptographic protection, rendering the data unintelligible throughout the computation process in the model training and inference.

The transition from traditional ML to PPML introduces non-trivial challenges. While conventional models have unrestricted access to training data and their algorithm performs a wide range of arithmetic and logical operations, an encrypted ML model must operate under strict constraints: (i) no access to plaintext data at any point during the computation in training and inference steps; (ii) restricted support to arithmetic operations, usually limited to additions and multiplications over a ciphertext space; and (iii) noise growth during the computations that can render ciphertexts undecipherable if it exceeds a certain depth. These constraints require algorithmic reformulations, such as controlling noise growth, polynomial approximations for non-linear functions and careful parameter selection to balance precision, security, and efficiency.

This paper discusses the challenges of implementing privacy preserving models using encrypted data in training and testing steps. Our discussions regarding the current capabilities and bottlenecks of PPML are grounded in theoretical and experimental analysis considering different healthcare problems requiring sensitive data protection. The main contributions of this study include:

- A discussion of the constraints imposed by Homomorphic Encryption on the implementation of ML algorithms, including the lack of native Boolean operations, limited supported arithmetic, and computational overhead.
- An analysis of the limitations of current HE schemes (e.g., BFV, BGV, and CKKS), such as noise accumulation, circuit depth, and restricted operation support, and their implications for model complexity and training scalability.
- Results obtained by adapting classical supervised learning models, such as Artificial Neural Networks (ANN) and Logistic Regression, for training and test phases using encrypted data for different healthcare problems.
- An evaluation of different approximation methods, including Chebyshev polynomials, Taylor expansion, and the Least Squares method, for implementing activation functions of classical ML models under encryption constraints.

This paper is organized as follows: Section 2 overviews key PPML paradigms and their trade-offs. Section 3 reviews previous work on encrypted ML. Section 4 outlines the experimental setup, including the adjustments conducted for fully HE adoption in the learning and inference phases. Section 5 reports the results in problems with sensitive data. Section 6 discusses the challenges of training and testing ML models on encrypted data, and suggests future research directions.

2 Privacy-Preserving Machine Learning (PPML)

Privacy-Preserving Machine Learning (PPML) encompasses a broad range of methods that enable statistical analysis and model training while preserving the data privacy. The main approaches include *Differential Privacy* (DP), *Federated Learning* (FL) and *Homomorphic Encryption* (HE), as discussed in the following.

2.1 Differential Privacy (DP)

Differential Privacy (DP) is a framework that offers provable privacy guarantees by introducing calibrated random noise into the computation process, whether at the input, intermediate, or output level, to ensure that the inclusion or exclusion of any single individual's data does not significantly affect the result [13]. This ensures that individual data points cannot be easily inferred from the model's behavior, thereby protecting the privacy of the data.

A randomized algorithm $\mathcal{A}$, i.e., an algorithm that introduces randomness in its computations, satisfies (ε, δ)-differential privacy if, for all pairs of datasets D and D' differing in at most one record, and for all measurable subsets S of the output space, Equation 1 holds.

$$\Pr[\mathcal{A}(D) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathcal{A}(D') \in S] + \delta \tag{1}$$

The parameter $\varepsilon > 0$ controls the privacy loss, with smaller values indicating stronger privacy. The parameter δ allows for a negligible probability of failure, typically required when using approximate mechanisms. DP is typically enforced by injecting calibrated noise into query outputs, gradients, or model parameters.

DP can be implemented under centralized or local models, each with distinct trust and responsibility assumptions [9]. In the centralized model, a trusted server manages privacy, security, and utility, requiring a secure data processor. In contrast, the local model shifts this responsibility to users, who perturb their data before sharing, enhancing privacy but often reducing accuracy. Despite its strong guarantees, DP entails a privacy-utility trade-off: lower ε increases noise, potentially degrading performance, especially in data-limited scenarios.

2.2 Federated Learning

Federated Learning (FL) is a decentralized learning paradigm in which the model training occurs on a federation of client devices that retain their local data. The central idea, introduced by McMahan et al. [20], is to preserve data locality while aggregating learned model updates in a privacy-conscious manner.

Formally, let K denote the number of participants, i.e., client devices. Training proceeds in *communication rounds*, where in each round t, a central server broadcasts the current global model parameters $w^{(t)}$ to all clients. These parameters $w^{(t)}$ represent the state of the model at round t, such as the vector of weights in a linear model or the parameters of a neural network. Each client k then updates $w^{(t)}$ using its own local dataset D_k, often by applying a few steps of a local optimization algorithm, yielding an updated model $w_k^{(t+1)}$. The server aggregates the received updates, e.g., via weighted averaging, to produce the

next global model $w^{(t+1)}$. This iterative process continues over multiple rounds, progressively improving the model without access to the clients' data.

FL removes the need for centralized data storage and reduces privacy risks. However, it does not inherently provide formal privacy guarantees because model updates can encode sensitive information about the local training data. Besides, its effectiveness depends on trusted client devices, secure communication, and proper protocol implementation. Multiple studies have shown that private features can be extracted from shared updates using attacks such as gradient inversion, member inference, and property inference [10].

2.3 Homomorphic Encryption

Homomorphic Encryption (HE) is a form of encryption that allows arithmetic operations be performed directly on encrypted data, also known as ciphertexts. The result of such operations, when decrypted, yields the same value as if the operations had been performed on plaintexts.

HE schemes are classified into categories based on the types and number of operations supported: (i) *Partially Homomorphic Encryption* (PHE), which supports only a single operation, the addition or the multiplication, but not both; (ii) *Somewhat Homomorphic Encryption* (SHE), which supports a limited number of both additions and multiplications before the noise in the ciphertext becomes too large to allow correct decryption; and (iii) *Fully Homomorphic Encryption* (FHE), which supports an unlimited number of operations by periodically applying a process called *bootstrapping*. Bootstrapping refreshes ciphertexts by reducing the accumulated noise, effectively allowing further computation without compromising decryptability. FHE is therefore the most suitable choice for encrypted ML, where models often perform successive operations.

Encryption schemes such as RSA [22] lack full homomorphism, being unsuitable for ML. Examples of FHE schemes include BFV [6], BGV [7], and CKKS [12], each with distinct trade-offs in arithmetic precision and performance. While BFV and BGV support exact integer arithmetic, CKKS enables approximate real-number computations, making it particularly well-suited for adapting ML models such as ANN and Logistic Regression. CKSS is detailed as follows.

Cheon-Kim-Kim-Song (CKKS): CKKS is a FHE scheme that enables approximate arithmetic over real and complex numbers. Its security relies on the Ring Learning With Errors (RLWE) [8], a mathematical problem considered hard to solve even with quantum computers. RLWE involves hiding information by adding small errors to polynomial equations in a way that makes it extremely difficult to recover the original data without a secret key. In our setting, the feature vectors used to represent the instances are encoded as polynomials.

Formally, let $\mathcal{R} = \mathbb{Z}[x]/(x^N + 1)$ be the ring of polynomials modulo the cyclotomic polynomial $x^N + 1$, where N is a power of two, and let $\mathcal{R}_q = \mathcal{R} \bmod q = \mathbb{Z}_q[x]/(x^N + 1)$ for a ciphertext modulus $q \in \mathbb{Z}$. The CKKS scheme supports approximate encryption of plaintext vectors $\mathbf{m} \in \mathbb{C}^k$ by first mapping them to a polynomial $m'(x) \in \mathcal{R}$ via a canonical encoding procedure, which is multiplied by

a scaling factor $\Delta \in \mathbb{R}$. Since the security of the scheme is based on RLWE, real or complex numbers cannot be directly represented in this structure, requiring a scaling to transform the numbers into large integers that approximate their values with high precision. After decryption, a division by the same Δ recovers an approximated version of the original values.

CKKS is an asymmetric scheme employing a *secret-public* key pair. The *secret key* is a small polynomial $s(x) \in \mathcal{R}_q$, sampled from a binary or ternary distribution and kept secret on the client side. Auxiliary keys derived from it enable homomorphic operations, such as multiplication, rotation, and bootstrapping, through *key switching* [1], but do not allow decryption. The *public key*, $\text{pk} = (b(x), a(x)) \in \mathcal{R}_q^2$, is constructed by sampling $a(x)$ uniformly from $\mathcal{R}_q$ and computing $b(x) = -a(x) \cdot s(x) + e(x)$ with noise $e(x)$ drawn from a discrete Gaussian distribution. This public key is used for encryption and is typically shared with the server, for example, to encrypt model weights of ANNs.

Noise is essential for CKKS security, as the hardness of RLWE relies on small random errors in ciphertexts. However, noise increases with each homomorphic operation, especially multiplications, and must be kept below a threshold to ensure correct decryption. CKKS mitigates this issue via rescaling, which controls noise while limiting multiplicative depth. For deeper computations, bootstrapping [11] refreshes ciphertexts by reducing accumulated noise, enabling continued operations at the cost of significant computational overhead.

The CKKS parameters are essential for balancing security and efficiency. The ring dimension N affects both security and computational cost. Larger values lead to higher resource usage but also make it significantly harder for attacks (i.e., decryption without the secret key). The ciphertext modulus q must be large enough to support noise growth during computations, especially multiplications, yet appropriately scaled with N to maintain security. Parameter choices are guided by security estimates, like the LWE estimator [5] and standardized recommendations, such as Homomorphic Encryption Standard [4].

3 Related Work

Homomorphic Encryption has been applied to a range of classical ML models to enable secure training and inference on encrypted data. This section reviews key contributions involving Logistic Regression and ANNs.

Cheon et al. [16] proposed a scalable approach for encrypted Logistic Regression using CKKS. They evaluated their method on the MNIST dataset, achieving 96.4% classification accuracy using encrypted inference, and showed that their solution scales to datasets with over 10,000 records with reasonable execution time (under one minute per inference). In the biomedical domain, Kim et al. [17] applied FHE to train Logistic Regression models on encrypted genomic data (SNP datasets), achieving similar classification performance to plaintext models ($AUC \approx 0.85$), with encryption increasing computation time by 15–20×.

CryptoNets [15] demonstrated that a convolutional neural network could perform encrypted inference using HE over the MNIST dataset, achieving 99%

accuracy. Each encrypted prediction required approximately 250 sec., highlighting the significant computational overhead introduced by HE. More recently, Nocker et al. [21] introduced HE-MAN, a framework for performing encrypted inference using HE. Their work focuses on evaluating pre-trained quantized neural networks on encrypted inputs using the Concrete and TenSEAL libraries. Experimental results on the MNIST dataset show that encrypted inference using a CryptoNets-style architecture requires approximately 7.1 sec. per sample, while encrypted face recognition using a MobileFaceNets architecture takes around 69 sec. per sample. However, all these framework do not support encrypted training, and computational performance remains a bottleneck for larger datasets.

Despite significant progress, current research on encrypted ML has largely focused on inference or simplified training tasks, often under constrained settings such as shallow models or small datasets. While several works have demonstrated the feasibility of encrypted inference and even partial training using HE, fully supporting the end-to-end lifecycle of machine learning, especially training deep or complex models on encrypted data, remains an open challenge. This is primarily due to the computational complexity, noise management, and parameter tuning required by HE schemes. The high cost of bootstrapping, limited multiplicative depth, and difficulties in approximating non-linear functions under encryption all contribute to the gap between theory and practical deployment.

4 Experimental Methodology

This section outlines the methodology for training and testing ML models on encrypted data, algorithm adaptations, cryptographic parameters, and datasets.

4.1 Privacy-preserving machine learning pipeline

Fig. 1 illustrates our privacy-preserving ML pipeline using homomorphic encryption. In this setting, only the client can access plaintext data, while a server accesses encrypted data for model training and inference. The client begins by generating a CKKS encryption key set, which includes a secret key (S), a public key (P), and a set of evaluation keys (E), such as relinearization and rotation keys, required for performing specific homomorphic operations on encrypted data. The public and evaluation keys are securely transmitted to the server, while the secret key remains exclusively with the client for data decryption.

The client encrypts the training and testing data using the public key, then sends the ciphertext and public/evaluation keys to the server. The server then trains a model directly on the encrypted training data. All computations, including weight updates and intermediate operations, are performed homomorphically using the evaluation keys and public key, without accessing the data in plaintext.

After training, the encrypted model performs inference on the encrypted testing dataset. The resulting encrypted predictions are returned to the client, who then decrypts them using the secret key to recover the plaintext outputs. The server never accesses unencrypted data throughout the process, ensuring end-to-end data confidentiality.

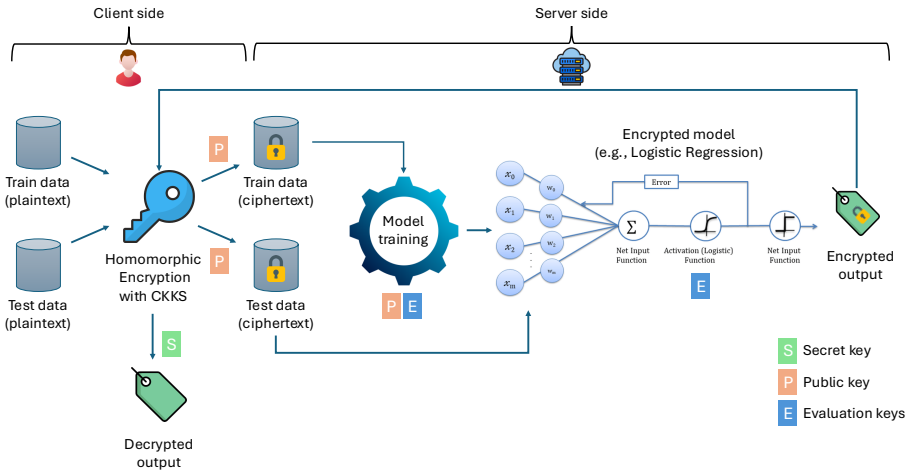


Fig. 1. Privacy-preserving ML pipeline. The client generates the keys before encrypting the data. The secret key remains on the client side and is never shared with the server, which performs model training and inference on encrypted data.

4.2 Adapting machine learning models for encrypted data

In the classical setting, Logistic Regression is a generalized linear model for binary classification that computes a score $z = w^T x$, where $w \in \mathbb{R}^n$ denotes the weight vector and $x \in \mathbb{R}^n$ the input feature vector. The score z is then mapped to a probability value in the interval $[0, 1]$ using the sigmoid function, as shown in Eq. 2. A binary decision is typically made by applying a threshold of 0.5 to the predicted probability. In this work, we also explore the use of the hyperbolic tangent function (tanh), defined in Eq. 3, which maps the score z to the interval $[-1, 1]$. The prediction is made by choosing one class if the output is positive, and another if it is zero or negative.

$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}} \qquad (2) \qquad\qquad \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \qquad (3)$$

In addition to Logistic Regression, we also consider ANNs, which extend the linear model by introducing one or more hidden layers composed of neurons that apply nonlinear activation functions to weighted sums of their inputs. This architecture allows ANNs to model complex nonlinear relationships between input features and target outputs. We explore sigmoid and tanh at the neuron level.

Smooth activation functions, such as the sigmoid and tanh, are fundamental to gradient-based optimization and probabilistic modeling. However, their exact evaluation is incompatible with the CKKS scheme, which does not support conditional branching or discontinuous operations. To overcome this limitation, we employ 5th-degree polynomial approximations, specifically Taylor expansion, Least Squares method, and Chebyshev polynomials, constructed over a common interval [-6, 6] to ensure numerical stability, bounded approximation error, and consistent comparison across methods.

To support the depth of arithmetic operations required during training, the multiplicative depth was configured to its maximum value of 30. Beyond this bound, noise growth was managed through the use of bootstrapping, which was applied periodically to refresh ciphertexts and restore their noise budget. In the following, we introduce each approximation method employed [24]. These approximations are composed entirely of additions, multiplications, and integer powers, making them well-suited for HE schemes like CKKS.

Taylor series. It approximates a smooth function $f(x)$ by a polynomial constructed from its derivatives at a single point, typically around $x = 0$. In this work, we adopt the 5th-degree Taylor expansions of the sigmoid and tanh functions centered at the origin, using only additions and multiplications:

$$\text{sigmoid}(x) \approx 0.5 + 0.25 \cdot x - 0.020833 \cdot x^3 + 0.002083 \cdot x^5 \tag{4}$$

$$\tanh(x) \approx x - 0.333333 \cdot x^3 + 0.133333 \cdot x^5 \tag{5}$$

Their low polynomial degree ensures shallow multiplicative depth, enabling efficient and precise evaluation on encrypted data. However, as Taylor series are accurate only near the expansion point, their approximation error increases substantially for inputs far from zero.

Chebyshev polynomials. Given a continuous function $f(x)$, its Chebyshev approximation of degree n is defined according to Eq. 6.

$$P_n(x) = \sum_{k=0}^{n} a_k T_k(x), \tag{6}$$

where $T_k(x)$ denotes the k-th Chebyshev polynomial of the first kind, and the coefficients $a_k \in \mathbb{R}$ are selected to minimize the maximum approximation error in the uniform norm. Chebyshev polynomials form an orthogonal basis with optimal convergence properties, making them well-suited for uniformly approximating smooth nonlinear functions. Chebyshev approximations are particularly attractive because they require only additions and multiplications and allow explicit control over the multiplicative depth, enabling a balance between approximation accuracy and ciphertext noise growth.

Least Squares. The method approximates a target function $f(x)$ by finding a polynomial $P_n(x) = \sum_{k=0}^{n} a_k x^k$ that minimizes the mean squared error over a chosen interval. Unlike Taylor expansion and Chebyshev, which rely on local derivatives or orthogonal bases, Least Squares fits the polynomial globally by minimizing the integral of the squared difference between $f(x)$ and $P_n(x)$. The polynomial fitted for sigmoid and tanh in the interval [-6, 6] are denoted below:

$$\text{sigmoid}(x) \approx 0.5 + 0.217033 \cdot x - 0.007811 \cdot x^3 + 0.0001179 \cdot x^5 \tag{7}$$

$$\tanh(x) \approx 0.590585 \cdot x - 0.028944 \cdot x^3 + 0.000502 \cdot x^5. \tag{8}$$

A key challenge in implementing ANNs using CKKS stems from ciphertext packing, which encodes multiple values into a single encrypted vector. Although

this enables parallel computations, it complicates access to individual positions in the weight matrix during the gradient estimation of backpropagation. To address this, our implementation employs rotation operations to shift encrypted elements into positions where the required computations can be carried out.

4.3 Cryptographic parameters

The CKKS security is based on the RLWE problem, which depends on carefully selecting parameter values, including the ring dimension, modulus size, and error distribution. To define these parameters, we follow the guidelines provided by the HE standardization [4], which suggests a ring dimension of 2048 with modulus $log_q = 56$, providing 128-bit security.

4.4 Datasets and validation procedure

Five medical datasets from the UCI Repository were used in our evaluation. These datasets were selected for their clinical relevance, diversity in feature types, and manageable size for encrypted computation. Table 1 summarizes the main characteristics. We highlight the substantial increase in sizes resulting from encryption, which can grow from KB to GB. The CKKS scheme causes this overhead, as even real-valued inputs are transformed into complex-valued vectors and encoded as high-degree polynomials to support encrypted computation.

Table 1. Datasets characteristics. The number of features is indicated as the sum of discrete (D) and continuous (C) variables. Dataset sizes are presented in MB.

Dataset	Samples Train | Test	Features D | C	Classes 0 | 1	Size of Plain	Size of Encr.
Breast Cancer	569 (398 | 171)	30 (0 | 30)	212 | 357	0.334	1,112
Cirrhosis Patient	418 (292 | 126)	17 (17 | 7)	161 | 257	0.101	816
Diabetes	768 (537 | 231)	8 (0 | 8)	500 | 268	0.110	1,048
Differentiated Thyroid	383 (268 | 115)	16 (13 | 3)	275 | 108	0.047	750
Glioma Grading	839 (587 | 252)	23 (20 | 3)	487 | 352	0.107	1,638

We consider holdout validation with 70% for training and 30% for testing. Categorical features were encoded using one-hot encoding, in which each value is represented as a binary vector. This transformation was necessary to ensure numerical compatibility with the HE scheme, which operates over fixed-precision vector spaces. Continuous features were normalized between [-1, 1] using min-max [19], reducing the approximation error during encrypted computations.

5 Analysis of results

All implementations are in C++ and available on our support website with additional results [18]. The ANN architecture comprises four layers: an input layer with one neuron per feature, two hidden layers with 10 and 5 neurons, respectively, and an output layer with a single neuron. Training and inference were performed on plaintext and ciphertext using CKKS, with sigmoid and tanh activation functions approximated by three polynomial methods. Both algorithms were trained with 10 epochs and a learning rate of 0.01. For encryption, we use OpenFHE [3] library and execute the experiments on an Ubuntu 24.04.2 LTS server with an Intel(R) Xeon(R) W-1290P CPU @ 3.70GHz and 64 GB of RAM.

Table 2 presents the Logistic Regression results. Overall, models employing the sigmoid activation function achieved higher performance, both in plaintext and under CKKS encryption. Notably, despite the inherent approximation and error accumulation introduced by HE, it was possible to train encrypted models with predictive performance equivalent to – or even surpassing – that of plaintext models. Focusing on the sigmoid activation and F1-score, we observe that among the five datasets, encrypted models achieved identical results in two cases (Breast Cancer and Glioma), outperformed plaintext models in two (Diabetes and Thyroid), and underperformed in only one (Cirrhosis).

Table 2. Results of Logistic Regression using plain and encrypted data.

Dataset	Encryption scheme	Activation function	Approximation method	Accur.	Prec.	Recall	F1
Breast Cancer	CKKS	sigmoid	Chebyshev	0.94	0.91	0.99	0.95
			Least Squares	0.94	0.91	0.99	0.95
			Taylor	0.94	0.91	1.00	0.95
		tanh	Chebyshev	0.70	0.68	1.00	0.81
			Least Squares	0.70	0.68	1.00	0.81
			Taylor	0.70	0.67	1.00	0.80
	Plain	sigmoid	Native	0.93	0.91	0.99	0.95
		tanh	Native	0.71	0.68	1.00	0.81
Cirrhosis Patient	CKKS	sigmoid	Chebyshev	0.75	0.72	0.99	0.83
			Least Squares	0.75	0.72	0.99	0.83
			Taylor	0.75	0.71	0.99	0.83
		tanh	Chebyshev	0.63	0.62	1.00	0.77
			Least Squares	0.63	0.62	1.00	0.77
			Taylor	0.63	0.62	1.00	0.77
	Plain	sigmoid	Native	0.67	0.65	1.00	0.79
		tanh	Native	0.62	0.62	1.00	0.76
Diabetes	CKKS	sigmoid	Chebyshev	0.74	0.77	0.37	0.50
			Least Squares	0.73	0.75	0.33	0.46
			Taylor	0.74	0.76	0.40	0.52
		tanh	Chebyshev	0.39	0.37	0.99	0.53
			Least Squares	0.39	0.36	0.99	0.53
			Taylor	0.40	0.37	0.99	0.54
	Plain	sigmoid	Native	0.74	0.76	0.38	0.51
		tanh	Native	0.41	0.37	0.99	0.54
Differentiated Thyroid	CKKS	sigmoid	Chebyshev	0.90	0.86	0.78	0.82
			Least Squares	0.90	0.86	0.78	0.82
			Taylor	0.90	0.86	0.78	0.82
		tanh	Chebyshev	0.71	0.49	1.00	0.66
			Least Squares	0.69	0.47	0.97	0.63
			Taylor	0.72	0.50	1.00	0.67
	Plain	sigmoid	Native	0.96	1.00	0.84	0.92
		tanh	Native	0.69	0.47	0.97	0.63
Glioma Grading	CKKS	sigmoid	Chebyshev	0.87	0.81	0.90	0.85
			Least Squares	0.87	0.81	0.90	0.85
			Taylor	0.87	0.81	0.91	0.85
		tanh	Chebyshev	0.65	0.55	0.95	0.69
			Least Squares	0.65	0.55	0.95	0.70
			Taylor	0.67	0.57	0.95	0.71
	Plain	sigmoid	Native	0.87	0.81	0.89	0.85
		tanh	Native	0.69	0.58	0.96	0.72

Fig. 2 shows the F1-scores obtained by the ANN using the Chebyshev approximation for the encrypted data. Similar to the Logistic Regression, other approximation methods produced comparable outcomes. We note that the tanh yielded the best results, while using sigmoid led to near-zero F1-scores in 3 out of 5 datasets. This may be due to the use of the same activation function in all network layers, which can be suboptimal when using sigmoid. More importantly, the performance on encrypted data remains close to that of the plaintext version.

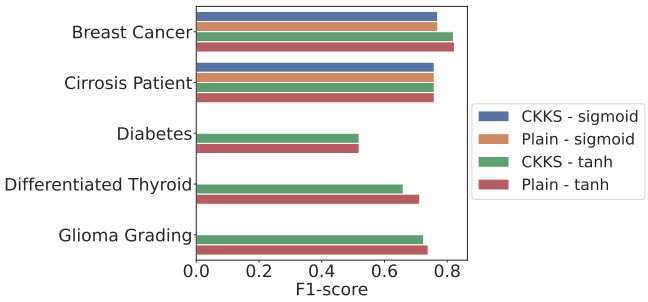
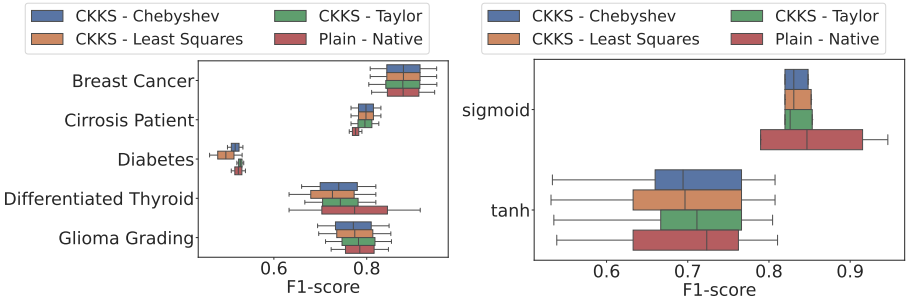


Fig. 2. Results of ANN using plain and encrypted data. The activation functions were approximated with Chebyshev. Sigmoid led to near-zero F1-scores in 3 out of 5 datasets.

5.1 Impact of approximation methods

One of the goals of this study is to evaluate the impact of modifications in the models to work under encryption constraints. Among these modifications, we highlight approximation methods for activation functions. As illustrated in Fig. 3-(a), when considering the F1-score across both sigmoid and tanh activations, the Taylor expansion was the method that most closely matched the performance of plaintext models, where no approximation is required.

In Fig. 3-(b), we compute the F1-score across the datasets considering each activation function. In addition to the superiority of the sigmoid, we can note the impact of the approximations with the reduction of the median F1-score when compared with the plaintext. Interestingly, the encrypted models using sigmoid exhibit lower variability, as indicated by the smaller interquartile ranges in the box plots, suggesting more consistent results despite the accuracy decrease.



(a) Median across activation functions (b) Median across datasets

Fig. 3. Impact in the predictive performance of Logistic Regression when using approximation methods (Chebyshev, Taylor, and Least Squares) for activation functions.

5.2 Computational performance

One of the main challenges of adapting ML models to encrypted data is the significant computational overhead due to the complexity of encrypted arithmetic operations. This overhead can impact both the training and inference phases.

Fig. 4 presents the training and testing times of Logistic Regression when employing the sigmoid function across the approximation functions. While we note no difference in the times of approximation methods, the encrypted models exhibit significantly higher computational costs for the training and testing phases compared to their plaintext counterparts. The similar results across approximation methods are expected, since the polynomial approximations used produce expressions of comparable degree and arithmetic complexity. Once compiled into homomorphic operations, these approximations result in a similar number of ciphertext multiplications and additions, which dominate the overall runtime. Thus, the choice of approximation method has a greater impact on predictive performance than on computational cost under encryption.

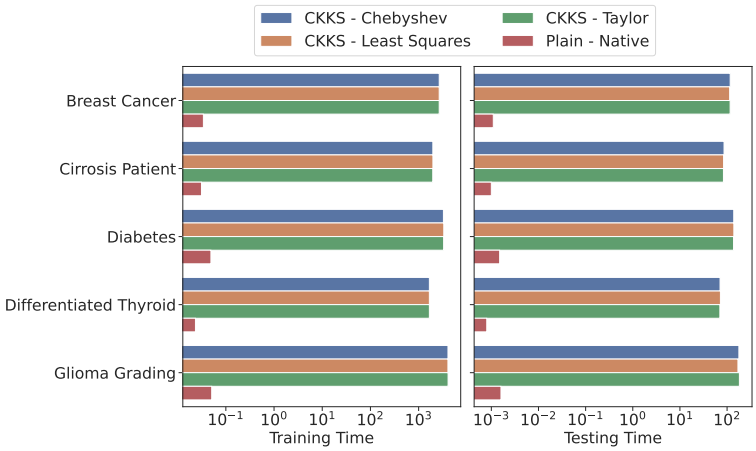
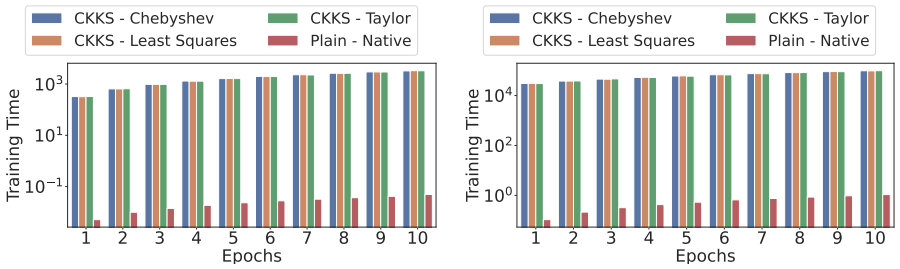


Fig. 4. Training and testing times (sec.) across the approximation methods for sigmoid on Logistic Regression.

While training and inference with plaintext data remain below 100 milliseconds and 1 millisecond, respectively, the encrypted versions of Logistic Regression require approximately 45 to 70 minutes for training and 2 to 3 minutes for inference. On the other hand, ANN spends between 12 and 24 hours on training and between 15 and 20 minutes on inference. For comparison, considering the plaintext, the training time of ANN is between 2 and 12 sec., and inference between 15 and 250 milisseconds.

This computational overhead is intrinsic to HE schemes, where even basic operations involve expensive polynomial arithmetic over ciphertexts. Fig. 5 shows the accumulated training time of both models over 10 epochs for Diabetes dataset. The main overhead occurs predominantly in the first epoch due to the initialization of encrypted data structures and the associated bootstrapping and rescaling operations.

In addition, we note a considerable difference between the times per epoch of Logistic Regression and ANN. While the time of Logistic Regression ranges from 5 to 60 min., ANN spends between 2 hours and 4 hours. Such a difference is due to the higher number of mathematical operations performed by ANN, which is required by the backpropagation process.



(a) Logistic Regression (sigmoid) (b) Artificial Neural Network (tanh)

Fig. 5. Accumulated training time (sec.) per epoch on Diabetes dataset. For each model, we consider the most adequate activation function, i.e., sigmoid for Logistic Regression and tanh for ANN.

6 Challenges and Conclusions

Homomorphic Encryption offers strong privacy guarantees for ML by enabling training and inference directly over encrypted data while keeping both the model and data encrypted throughout the entire pipeline. This setting prevents information leakage from the inputs, outputs, and model itself, even when executed in untrusted environments. However, implementing such models introduces significant challenges due to the mathematical and computational constraints imposed by HE schemes, such as CKKS. These limitations affect the range of algorithms that can be adapted and computational efficiency, as shown in our results.

A key limitation is the difficulty of implementing boolean logic under CKKS, requiring approximations through polynomial functions. This constraint makes the adaptation of popular algorithms such as k-Nearest Neighbors (kNN) and Decision Trees challenging. For instance, kNN relies heavily on comparisons to identify the nearest neighbors, and Decision Trees require branching based on logical conditions, operations that are not natively supported and must be approximated, affecting the computational cost and accuracy.

For the algorithms investigated in this article, the activation functions were approximated using low-degree polynomials. However, only smooth and bounded functions, such as sigmoid and tanh, can be efficiently approximated with this solution. Functions like ReLU, which rely on conditional logic, are incompatible with the approximation methods evaluated. Although other approximation

methods to ReLU exist, they tend to introduce significant errors or require higher polynomial degrees, increasing computational cost and noise growth. In addition, while the derivatives of sigmoid and tanh functions can be expressed using only subtractions and multiplications, the derivative of functions such as softplus involves a division operation, which is not natively supported by CKKS and therefore complicates gradient computation during backpropagation in ANNs.

A fundamental challenge is balancing numerical precision, computational cost, and security. Approximation noise accumulates with each operation, requiring high-resolution encoding and deep modulus chains to preserve accuracy—at the expense of runtime and memory. When depth limits are reached, bootstrapping adds further overhead. Ensuring accurate, efficient learning under 128-bit security remains a key open problem, especially for iterative algorithms.

Challenges related to the restricted set of arithmetic operations can be surpassed by replacing the HE scheme CKKS with alternatives such as TFHE (Fast Fully Homomorphic Encryption over the Torus). However, we noted a superior computational time of TFHE in preliminary analysis, which made us choose CKKS. Hardware acceleration could alleviate the performance bottlenecks, but current libraries offer limited support. Advancing encrypted ML requires continued research into improved encryption schemes and hardware accelerations.

The contribution of this work lies in its demonstration of how HE can be effectively employed to enable privacy-preserving machine learning in realistic scenarios, particularly in the healthcare domain, where data sensitivity is a critical concern. We successfully implemented the entire pipeline using encrypted data, achieving predictive performance equivalent to plaintext models. However, the implementation of different algorithms and the increased computational overhead are challenges for future work. The costs can be reduced through hardware acceleration strategies, such as GPU-based implementations, parallelization techniques, or tailored optimizations for polynomial arithmetic over ciphertexts.

Acknowledgments. J.P. Barddal would like to thank Fundação Araucária for partially funding this research under grant number 188/2025.

References

1. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys **51**(4), 1–35 (2018)
2. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In: Proceedings of the ACM SIGMOD/PODS. pp. 439–450 (2000)
3. Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., et al.: Openfhe: Open-source fully homomorphic encryption library. In: Proceedings of the workshop on encrypted computing & applied homomorphic cryptography. pp. 53–63 (2022)
4. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., et al.: Homomorphic encryption standard. Protecting privacy through homomorphic encryption pp. 31–62 (2021)
5. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)

6. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: Annual cryptology conference. pp. 868–886. Springer (2012)
7. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory **6**(3), 1–36 (2014)
8. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Annual cryptology conference. pp. 505–524. Springer (2011)
9. Chang, J.M., Zhuang, D., Samaraweera, G., Samaraweera, G.D.: Privacy-Preserving Machine Learning. Simon and Schuster (2023)
10. Chen, Y., Gui, Y., Lin, H., Gan, W., Wu, Y.: Federated learning attacks and defenses: A survey. In: Proceedings of IEEE Big Data. pp. 4256–4265. IEEE (2022)
11. Cheon, J.H., Han, K., Kim, A., Kim, M., Song, Y.: Bootstrapping for approximate homomorphic encryption. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 360–384. Springer (2018)
12. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International conference on the theory and applications of cryptology and information security. pp. 409–437. Springer (2017)
13. Dwork, C.: Differential privacy: A survey of results. In: International conference on theory and applications of models of computation. pp. 1–19. Springer (2008)
14. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the ACM STOC. pp. 169–178 (2009)
15. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: International Conference on Machine Learning. pp. 201–210 (2016)
16. Han, K., Hong, S., Cheon, J.H., Park, D.: Logistic regression on homomorphic encrypted data at scale. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 9466–9471 (2019)
17. Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.H.: Logistic regression model training based on the approximate homomorphic encryption. BMC medical genomics **11**, 23–31 (2018)
18. Kruger, R., Barddal, J.P., Souza, V.M.A.: Support website (2025), `https://rodkruger.github.io/BRACIS2025/`
19. Lima, F.T., Souza, V.M.A.: A large comparison of normalization methods on time series. Big Data Research **34**, 100407 (2023)
20. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics. pp. 1273–1282. PMLR (2017)
21. Nocker, M., Drexel, D., Rader, M., Montuoro, A., Schöttle, P.: He-man–homomorphically encrypted machine learning with onnx models. In: Proceedings of ICMLT. pp. 35–45 (2023)
22. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
23. Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., Gao, Y.: A survey on federated learning. Knowledge-Based Systems **216**, 106775 (2021)
24. Zhao, Y., Li, X.: Better approximation of sigmoid function for privacy-preserving neural networks. In: Journal of Physics: Conference Series. vol. 2852, p. 012007. IOP Publishing (2024)