ReBoot: Encrypted Training of Deep Neural Networks with CKKS Bootstrapping

Alberto Pirillo*, **Luca Colombo***

Department of Electronics, Information and Bioengineering
Politecnico di Milano, Via Ponzio 34/5, Milan, 20133, Italy
alberto.pirillo@mail.polimi.it, luca2.colombo@polimi.it

Abstract

Growing concerns over data privacy underscore the need for deep learning methods capable of processing sensitive information without compromising confidentiality. Among privacy-enhancing technologies, Homomorphic Encryption (HE) stands out by offering post-quantum cryptographic security and end-to-end data protection, safeguarding data even during computation. Prior research on encrypted training has primarily focused on logistic regression, model fine-tuning, or relied on multi-party computation. This is largely due to the substantial computational overhead and algorithmic complexity involved in training deep Neural Networks (NNs) under HE. In this paper, we present ReBoot, the first framework to enable fully encrypted and non-interactive training of Multi-Layer Perceptrons (MLPs) using CKKS bootstrapping. ReBoot introduces a novel HE-compliant NN architecture based on local error signals, specifically designed to minimize multiplicative depth and reduce noise accumulation during training. It employs a tailored packing strategy that leverages real-number arithmetic through CKKS *SIMD* operations, significantly lowering both computational and memory overhead. We evaluate ReBoot on both image and tabular benchmarks, demonstrating up to $+6.83\%$ improvement in test accuracy over existing solutions, while reducing training latency by up to $8.83\times$. ReBoot is made available to the scientific community as a public repository.

Code — https://github.com/AI-Tech-Research-Lab/ReBoot
Extended version — https://arxiv.org/abs/2506.19693

1 Introduction

The increase concerns over data protection have sparked widespread interest in the integration of privacy-enhancing technologies into Machine Learning (ML) and Deep Learning (DL) applications. Among these, Homomorphic Encryption (HE) stands out for its unique ability to support computations directly on encrypted data (Acar et al. 2018). By offering post-quantum cryptographic security, HE enables secure data processing in untrusted environments, such as cloud services, without ever exposing sensitive information (Brakerski et al. 2013). Despite its potential, HE poses significant challenges due its limited set of supported arithmetic operations and the bounded number of consecutive operations that can be performed on encrypted data, primarily caused by noise accumulation. *Bootstrapping* is a ciphertext-refreshing operation introduced in Fully Homomorphic Encryption (FHE) schemes, such as BFV (Fan and Vercauteren 2012) and TFHE (Chillotti et al. 2018), that enables unlimited encrypted computation by resetting noise level (Gentry 2009). An *approximate* variant has been introduced for Leveled Homomorphic Encryption (LHE) schemes like CKKS (Cheon et al. 2017; Drucker et al. 2022). While this form of bootstrapping does not reduce noise, it restores ciphertext structure, thereby extending the number of consecutive operations that can be performed before corruption (Al Badawi and Polyakov 2023). However, training ML and DL models over encrypted data remains an open research challenge due to three major factors: *(i)* the inherent noise growth within LHE ciphertexts, *(ii)* the high computational cost of homomorphic operations, and *(iii)* the functional limitations imposed by HE schemes. As a result, most existing research has focused either on encrypted training of simpler ML models, fine-tuning the final layer of Deep Neural Networks (DNNs) within transfer learning frameworks, or on *re-encryption*-based multi-party computation settings.

In this regard, the aim of this paper is to address the following research question: *how can CKKS approximate bootstrapping be effectively leveraged to enable fully encrypted interaction-free training of DNNs?* To the best of our knowledge, we propose ReBoot, the first framework in the literature to enable end-to-end non-interactive encrypted training of real-valued Multi-Layer Perceptrons (MLPs) on encrypted data using CKKS. ReBoot components are specifically designed to manage noise accumulation while minimizing both computational and memory overhead associated with HE. It introduces a novel HE-compliant Neural Network (NN) architecture, along with an advanced packing technique that exploits CKKS ability to perform *Single Instruction, Multiple Data (SIMD)* operations. By integrating approximate bootstrapping, ReBoot learning algorithm supports deep encrypted computation pipelines, allowing fully non-interactive training of MLPs and making it well-suited for practical and scalable ML as-a-service scenarios. ReBoot is implemented as an open-source Python library based on OpenFHE (Badawi et al. 2022).

*These authors contributed equally.

In summary, our contributions are:

1. A packing strategy that reduces computational and memory overhead by exploiting CKKS *SIMD* capabilities.
2. A HE-compliant MLP architecture in which all operations are optimized to manage noise accumulation.
3. A non-interactive encrypted learning algorithm for training HE-compliant MLPs on encrypted data.

The paper is organized as follows. Section 2 surveys related literature. Section 3 outlines the CKKS scheme. Section 4 details ReBoot, while experimental results are presented in Section 5. Lastly, Section 6 concludes the work.

2 Related Literature

Logistic regression was among the first models to be trained under encryption due to its architectural simplicity, with early implementations using the leveled CKKS scheme (Kim et al. 2018; Bergamaschi et al. 2019; Crockett 2020). Subsequent work integrated approximate bootstrapping (Han et al. 2018; Carpov et al. 2019), enabling more practical applications and paving the way for encrypted fine-tuning methods that update only the final layer of pre-trained models (Al Badawi et al. 2020; Walch et al. 2022; Zhang et al. 2022; Lee et al. 2023; Panzade, Takabi, and Cai 2024).

Recent research has shifted toward training full DNNs using FHE. Leveraging TFHE Boolean gates, Montero et al. (2024) introduced interaction-based training, in which ciphertexts are *re-encrypted* after each mini-batch to refresh noise levels, while Yoo and Yoon (2021) and Colombo, Falcetta, and Roveri (2024) implemented end-to-end encrypted training with bootstrapping support for deeper architectures. However, TFHE lack of *batching* capabilities forces homomorphic operations to be executed sequentially, resulting in prohibitive latency. Nandakumar et al. (2019) explored the use of BGV (Brakerski, Gentry, and Vaikuntanathan 2012) to train small MLPs, while Lou et al. (2020) proposed a BFV-based training approach that leverages *scheme-switching* to TFHE in order to exploit FHE bootstrapping. Nonetheless, both BGV and BFV operate on integers, requiring quantization of data and model parameters, a constraint that increases design complexity and degrades model accuracy (Pirillo, Colombo, and Roveri 2024).

To retain real-valued precision during training, Mihara et al. (2020) proposed an interactive CKKS-based protocol that relies on a *re-encryption* mechanism to both refresh noise in the model weights and reorder values within ciphertexts. Although effective, its reliance on client-server communication, introducing latency in model transmission and increasing bandwidth requirements, limits its applicability in *as-a-service* scenarios. In contrast, ReBoot stands out as the first framework to enable fully encrypted non-interactive training of real-valued MLPs with CKKS bootstrapping.

3 The CKKS Scheme

HE is a class of encryption schemes that enables computations directly on encrypted data (Acar et al. 2018). Formally, an encryption function $E(\mathsf{pk}, \cdot)$ and its corresponding decryption function $D(\mathsf{sk}, \cdot)$ are homomorphic w.r.t. a set of functions $\mathcal{F}$ if, for any function $f \in \mathcal{F}$, there exists a function g such that $f(x) = D\left(\mathsf{sk}, g\left(E(\mathsf{pk}, x)\right)\right)$ for any input x, where pk and sk denote the public and secret keys (Boemer et al. 2019). Among HE schemes, our work employs CKKS, which supports approximate arithmetic over encrypted complex numbers, making it well suited for ML and DL applications. Based on the RLWE problem (Chase et al. 2017), CKKS defines ciphertexts through encryption parameters $\Theta = \{N, \mathbf{q}, \Delta\}$, where N is a power of two defining the *polynomial degree*, $\mathbf{q}$ is a list of $l+2$ prime numbers forming the *coefficient moduli*, and Δ is the *scaling factor*. These parameters determine the *security level* λ, the encoding precision, and the number of available consecutive operations l. Each multiplication consumes one level of the modulus chain $\mathbf{q}$. Once level 0 is reached, further multiplications are prohibited due to excessive noise accumulation.

CKKS supports *batching* (Smart and Vercauteren 2014), a technique enabling parallel processing of multiple values via *SIMD*-style operations. Formally, a vector $\mathbf{z} \in \mathbb{C}^{N/2}$ is mapped to a polynomial $z(X) \in \mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ using a variant of the *complex canonical embedding*. The resulting plaintext is then encrypted with the public key pk to obtain a ciphertext $\underline{\mathbf{z}} \in \mathcal{R}_{\mathbf{q}} = \mathbb{Z}_{q_0}[X]/(X^N + 1)$.

CKKS natively supports element-wise operations on encrypted vectors. Given two ciphertexts $\underline{\mathbf{a}}$ and $\underline{\mathbf{b}}$, homomorphic addition, subtraction, and multiplication are defined as

$$\underline{\mathbf{a}} +_\bullet \underline{\mathbf{b}} = [a_1 + b_1, a_2 + b_2, \ldots, a_{N/2} + b_{N/2}]\,,$$
$$\underline{\mathbf{a}} -_\bullet \underline{\mathbf{b}} = [a_1 - b_1, a_2 - b_2, \ldots, a_{N/2} - b_{N/2}]\,,$$
$$\underline{\mathbf{a}} \odot_\bullet \underline{\mathbf{b}} = [a_1 \cdot b_1, a_2 \cdot b_2, \ldots, a_{N/2} \cdot b_{N/2}]\,.$$

Analogous operations can be performed between ciphertexts and plaintext vectors, denoted by $+_\circ$, $-_\circ$, and $\odot_\circ$, respectively. Additionally, CKKS supports homomorphic vector rotation, which cyclically left-shift elements by k positions.

By leveraging batching, matrices can be encoded in their flattened form into single ciphertexts, enabling efficient homomorphic summation across specific dimensions (Han et al. 2018). Consider a flattened encrypted matrix $\underline{\mathbf{W}}$ of dimension $r \times c = N/2$, where both r and c are powers of two. Summation across rows is defined as

$$\underline{\mathbf{S}} = \sum\nolimits_\bullet^{\text{rows}} \underline{\mathbf{W}} = \left[\sum_{i=1}^{r} W_{i,j}\right]_{j=1}^{c}, \tag{1}$$

where $\underline{\mathbf{S}}$ is a flattened matrix of size $r \times c$, and each j-th column of $\underline{\mathbf{S}}$ contains the replicated sum of the j-th column of $\underline{\mathbf{W}}$. Similarly, summation across columns is defined as

$$\underline{\mathbf{S}} = \sum\nolimits_\bullet^{\text{cols}} \underline{\mathbf{W}} = \left[\sum_{j=1}^{c} W_{i,j}\right]_{i=1}^{r}, \tag{2}$$

where each i-th row of $\underline{\mathbf{S}}$ contains the replicated sum of the i-th row of $\underline{\mathbf{W}}$. A more detailed description of these operations is provided in Appendix A (Pirillo and Colombo 2025).

To overcome the limit on consecutive multiplications, CKKS employs *approximate bootstrapping*, a procedure that restores the ciphertext modulus level. Given a ciphertext $\underline{\mathbf{c}}$ at level 0, bootstrapping produces a refreshed ciphertext $\underline{\mathbf{c}}' = \textit{Bootstrap}\left(\underline{\mathbf{c}}\right)$ at level $l - d > 0$, where d is the multiplicative depth of the bootstrapping circuit and $\underline{\mathbf{c}}' \approx \underline{\mathbf{c}}$.

4 The Proposed Solution

This section introduces ReBoot, a framework specifically designed for training encrypted MLPs on multi-class classification tasks defined over a dataset $\mathbf{X} = \{\mathbf{x}^\mu, y^\mu\}_{\mu=1}^{Q}$, where $\mathbf{x}^\mu \in \mathbb{R}^{k_0}$ are the input samples of dimension k_0, $y^\mu \in \{1, \ldots, o\}$ are the class labels (with o denoting the number of classes), and Q is the size of the training set.

4.1 ReBoot Packing

ReBoot encrypted processing is designed to maximize computational efficiency through a structured packing strategy that organizes data within the ciphertext space. It uses two packing schemes to encode vectors and matrices into ciphertexts. Specifically, inputs, weights, activations, and gradients are packed into matrices $\tilde{\mathbf{M}} \in \mathbb{R}^{r \times c}$, with $r \times c = N/2$, and encrypted into single ciphertexts $\underline{\tilde{\mathbf{M}}}$. The encoding dimensions r and c are determined by the DNN architecture, as explained in Section 4.2, and remain fixed for all matrices. This uniform structure enables the *a priori* definition of a fixed set of CKKS rotation keys and efficient *SIMD* execution, eliminating the need for *repacking*, a common procedure used to reorder values within ciphertexts. As a result, ReBoot reduces the number of ciphertexts and homomorphic operations, thereby decreasing the required multiplicative depth and the frequency of costly bootstrapping.

Let $\mathbf{a} \in \mathbb{R}^d$, with $d \leq N/2$, be a one-dimensional vector. Reboot encodes $\mathbf{a}$ into a matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{r \times c}$, structured either in repeated format

$$\tilde{\mathbf{A}}_{\text{rep}} = \begin{bmatrix} a_1 & a_2 & \ldots & a_d & 0 & \ldots & 0 \\ \vdots & & & & & & \\ a_1 & a_2 & \ldots & a_d & 0 & \ldots & 0 \end{bmatrix},$$

where each row contains a zero-padded copy of $\mathbf{a}$ to reach row length c, or in *expanded* format

$$\tilde{\mathbf{A}}_{\text{exp}} = \begin{bmatrix} a_1 & a_1 & \ldots & a_1 \\ \vdots & & & \\ a_d & a_d & \ldots & a_d \\ 0 & 0 & \ldots & 0 \\ \vdots & & & \\ 0 & 0 & \ldots & 0 \end{bmatrix},$$

where each column contains a zero-padded copy of $\mathbf{a}$ to reach column length r. Similarly, let $\mathbf{W} \in \mathbb{R}^{d \times k}$, with $d \times k \leq N/2$, be a two-dimensional matrix. Reboot encodes $\mathbf{W}$ into a matrix $\tilde{\mathbf{W}} \in \mathbb{R}^{r \times c}$, using either a *row-wise* format

$$\tilde{\mathbf{W}}_{\text{row}} = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1k} & 0 & \ldots & 0 \\ \vdots & & & & & & \\ W_{d1} & W_{d2} & \cdots & W_{dk} & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & & & & & & \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \end{bmatrix},$$

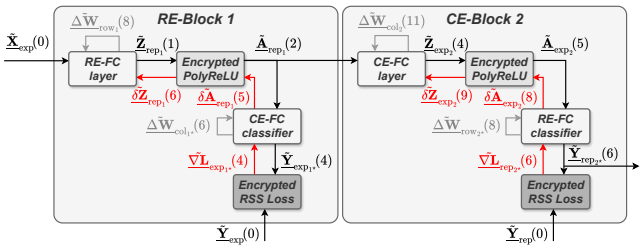


Figure 1: Overview of ReBoot MLP with two hidden layers. Forward, backward, and weight updates are shown in black, red, and gray. Multiplicative depths appear in parentheses.

where $\mathbf{W}$ is padded with $r - d$ rows and $c - k$ columns of zeros, or a *column-wise* format

$$\tilde{\mathbf{W}}_{\text{col}} = \begin{bmatrix} W_{11} & W_{21} & \cdots & W_{d1} & 0 & \ldots & 0 \\ \vdots & & & & & & \\ W_{1k} & W_{2k} & \cdots & W_{dk} & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & & & & & & \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \end{bmatrix},$$

where the transposed matrix $\mathbf{W}^\intercal$ is padded with $r - k$ rows and $c - d$ columns of zeros.

4.2 ReBoot Encrypted Architecture

This section presents ReBoot encrypted NN architecture and its main components, as shown in Figure 1.

Encrypted Fully-Connected Layer. ReBoot encrypted Fully-Connected (FC) layer performs a vector-matrix multiplication between an encrypted activation vector $\underline{\tilde{\mathbf{A}}}$ and an encrypted weight matrix $\underline{\tilde{\mathbf{W}}}$, producing an encrypted pre-activation vector $\underline{\tilde{\mathbf{Z}}}$. This multiplication can take two forms, depending on the dimension along which the summation is computed. Row Encrypted Matrix Multiplication (*RE-Matmul*) performs summation across rows $\sum_{\bullet}^{\text{rows}}$, as defined in Equation 1, and is expressed as

$$\underline{\tilde{\mathbf{Z}}}_{\text{rep}} = \textit{RE-Matmul}\left(\underline{\tilde{\mathbf{A}}}_{\text{exp}}, \underline{\tilde{\mathbf{W}}}\right) = \sum_{\bullet}^{\text{rows}} \tilde{\mathbf{A}}_{\text{exp}} \odot_{\bullet} \underline{\tilde{\mathbf{W}}}.$$

Conversely, Column Encrypted Matrix Multiplication (*CE-Matmul*) performs summation across columns $\sum_{\bullet}^{\text{cols}}$, as defined in Equation 2, and is expressed as

$$\underline{\tilde{\mathbf{Z}}}_{\text{exp}} = \textit{CE-Matmul}\left(\underline{\tilde{\mathbf{A}}}_{\text{rep}}, \underline{\tilde{\mathbf{W}}}\right) = \sum_{\bullet}^{\text{cols}} \tilde{\mathbf{A}}_{\text{rep}} \odot_{\bullet} \underline{\tilde{\mathbf{W}}}.$$

From a computational perspective, as detailed in Appendix A (Pirillo and Colombo 2025), *RE-Matmul* requires a multiplicative depth of 1, incurred by the element-wise multiplication, along with $\log_2(r)$ additions and rotations. Conversely, *CE-Matmul* requires a multiplicative depth of 2, due to both the element-wise multiplication and a masking step, along with $2\ \log_2(c)$ additions and rotations.

Let $\mathcal{H} = \{1, \ldots, H\}$ denote the set of ReBoot encrypted FC layers, where H is the total number of hidden layers. Let

k_h represent the dimensionality of layer $h \in \mathcal{H}$, with k_0 denoting the input size. The set $\mathcal{H}$ is partitioned into Row Encrypted FC (*RE-FC*) layers $\mathcal{R} = \{h \in \mathcal{H} \mid h \bmod 2 = 1\}$, which use *row-wise* encoded weights, and Column Encrypted FC (*CE-FC*) layers $\mathcal{C} = \{h \in \mathcal{H} \mid h \bmod 2 = 0\}$, which use *column-wise* encoded weights. Alternating between *RE-FC* and *CE-FC* layers ensures that the output format of one layer matches the input format of the next, eliminating the need for explicit ciphertext repacking between layers. ReBoot packing dimensions r and c, introduced in Section 4.1, are then defined as follows:

$$\begin{cases} r = 2^{\lceil \log_2\left(\max_{h \in \mathcal{R} \cup \{0\}} k_h\right) \rceil} \\ c = 2^{\lceil \log_2(\max_{h \in \mathcal{C}} k_h) \rceil} \end{cases}.$$

Encrypted Activation Function. ReBoot applies an encrypted non-linearity to the pre-activation vector $\tilde{\mathbf{Z}}$, producing the activation vector $\tilde{\mathbf{A}}$. Since CKKS does not support non-polynomial functions like *ReLU* and *tanh*, ReBoot uses *EncryptedPolyReLU*, a second-degree polynomial approximation of *ReLU* (Ali, So, and Avestimehr 2024), defined as

$$\underline{\tilde{\mathbf{A}}} = \left(\underline{\tilde{\mathbf{Z}}} \odot_\bullet \underline{\tilde{\mathbf{Z}}}\right) +_\bullet \underline{\tilde{\mathbf{Z}}}.$$

This operation has a multiplicative depth of 1, contributed by the element-wise multiplication, which preserves the encoding structure between input and output encrypted vectors.

Encrypted Local-Loss Block. To reduce multiplicative depth and limit costly bootstrapping, Reboot segments the MLP into multiple encrypted local-loss blocks, aligning with a stream of literature that employs local layer-wise loss functions (Nøkland and Eidnes 2019; Patel, Eickenberg, and Belilovsky 2023; Colombo, Pittorino, and Roveri 2025). Each block confines gradient propagation within itself, restricting HE noise accumulation and enabling independent and parallel training across blocks, significantly reducing the computational overhead. As shown in Figure 1, each ReBoot FC layer $h \in \mathcal{H}$, with its associated *EncryptedPolyReLU* activation, is paired with a local classifier $h^\star$ of size $k_h \times o$, forming a block B_h. The local classifier $h^\star$, itself an encrypted FC layer, projects the activation $\underline{\tilde{\mathbf{A}}}_h$ to the output dimension o. During the backward pass, the local output $\underline{\tilde{\mathbf{Y}}}^\star_h$ and the true label $\underline{\tilde{\mathbf{Y}}}$ are used by the local loss function L_h to compute the local gradient $\underline{\tilde{\nabla}\mathbf{L}}_h$, which is then used by ReBoot training algorithm to update the weights of both the layer h and its local classifier $h^\star$. In the final block B_H, the local classifier $H^\star$ serves as the NN output layer, producing the final prediction $\underline{\tilde{\mathbf{Y}}}^\star_H$. Blocks B_h fall into two categories: Row Encrypted Blocks (*RE-Block*s), when $h \in \mathcal{R}$, and Column Encrypted Blocks (*CE-Block*s), when $h \in \mathcal{C}$. Notably, a *RE-Block* has a local classifier $h^\star \in \mathcal{C}$, while a *CE-Block* has a local classifier $h^\star \in \mathcal{R}$.

4.3 ReBoot Encrypted Learning Algorithm

This section outlines the key components of ReBoot encrypted learning algorithm. Since each local-loss block is trained independently, and the extension to *CE-Block*s is straightforward, we focus on a representative *RE-Block* B_h, with $h \in \mathcal{R}$. Full algorithms are provided in Appendix B (Pirillo and Colombo 2025).

Forward Pass. At each training iteration $t \in \{1, \ldots, T\}$, the forward pass begins with the activations from the previous *CE-Block* B_{h-1}, or from the input samples if $h = 1$. The *RE-Block* B_h computes the pre-activations as:

$$\underline{\tilde{\mathbf{Z}}}^t_{\mathrm{rep}_h} = \textit{RE-Matmul}\left(\underline{\tilde{\mathbf{A}}}^t_{\mathrm{exp}_{h-1}}, \underline{\tilde{\mathbf{W}}}^t_{\mathrm{row}_h}\right),$$

which are then passed through *EncryptedPolyReLU* to produce the activations $\underline{\tilde{\mathbf{A}}}^t_{\mathrm{rep}_h}$. These activations are simultaneously propagated to the next *CE-Block* B_{h+1} and to the local classifier $h^\star$, which computes the local predictions as:

$$\underline{\tilde{\mathbf{Y}}}^t_{\mathrm{exp}_{h^\star}} = \textit{CE-Matmul}\left(\underline{\tilde{\mathbf{A}}}^t_{\mathrm{rep}_h}, \underline{\tilde{\mathbf{W}}}^t_{\mathrm{col}_{h^\star}}\right).$$

The predictions are then fed to the encrypted loss function to compute the gradients required to update *RE-FC* layer h and *CE-FC* local classifier $h^\star$.

Encrypted Loss Function. The objective of ReBoot learning algorithm is to minimize the encrypted local loss function L_h at each block B_h. Specifically, ReBoot employs the residual sum of squares loss, whose gradient is given by:

$$\underline{\tilde{\nabla}\mathbf{L}}^t_{\mathrm{exp}_{h^\star}} = \underline{\tilde{\mathbf{Y}}}^t_{\mathrm{exp}_{h^\star}} -_\bullet \underline{\tilde{\mathbf{Y}}}_{\mathrm{exp}}.$$

This gradient offers two key advantages in the encrypted domain: it does not increase the multiplicative depth, thereby limiting noise accumulation, and it matches the *cross-entropy* gradient when coupled with a *sigmoid*, retaining effective learning dynamics under CKKS constraints.

Backward Pass. Once the encrypted loss gradient has been computed, the corresponding encrypted gradients w.r.t. weights, activations, and pre-activations are derived. First, the gradient w.r.t. the *CE-FC* local classifier weights is computed via element-wise multiplication between the *repeated*-format activations and the *expanded*-format loss gradient:

$$\underline{\delta\tilde{\mathbf{W}}}^t_{\mathrm{col}_{h^\star}} = \underline{\tilde{\mathbf{A}}}^t_{\mathrm{rep}_h} \odot_\bullet \underline{\tilde{\nabla}\mathbf{L}}^t_{\mathrm{exp}_{h^\star}}.$$

Second, the gradient w.r.t. the activations is computed using *RE-Matmul* between the loss gradient and the *column-wise* encoded weights of the local classifier, which implicitly represents the transposed weight matrix:

$$\underline{\delta\tilde{\mathbf{A}}}^t_{\mathrm{rep}_h} = \textit{RE-Matmul}\left(\underline{\tilde{\nabla}\mathbf{L}}^t_{\mathrm{exp}_{h^\star}}, \underline{\tilde{\mathbf{W}}}^t_{\mathrm{col}_{h^\star}}\right).$$

Third, the gradient w.r.t. the pre-activations is obtained by evaluating the derivative of the activation function *EncryptedPolyReLU′*, computed as:

$$\underline{\delta\tilde{\mathbf{Z}}}^t_{\mathrm{rep}_h} = \left(\mathbf{2}_{r \times c} \odot_\circ \underline{\tilde{\mathbf{Z}}}^t_{\mathrm{rep}_h} +_\circ \mathbf{1}_{r \times c}\right) \odot_\bullet \underline{\delta\tilde{\mathbf{A}}}^t_{\mathrm{rep}_h},$$

where $\{\mathbf{1}, \mathbf{2}\} \in \mathbb{R}^{r \times c}$ are plaintext matrices filled with constants 1 and 2, respectively. Finally, the gradient w.r.t. the *RE-FC* layer weights is obtained as:

$$\underline{\delta\tilde{\mathbf{W}}}^t_{\mathrm{row}_h} = \underline{\tilde{\mathbf{A}}}^t_{\mathrm{exp}_{h-1}} \odot_\bullet \underline{\delta\tilde{\mathbf{Z}}}^t_{\mathrm{rep}_h}.$$

These gradients are then used by ReBoot encrypted weight update algorithm to update the weights of both the *CE-FC* local classifier $h^\star$ and the *RE-FC* layer h.

Name	Architecture	N	l	Δ
eMLP-1	*RE-Block* (32)	2^{16}	24	2^{49}
eMLP-2	*RE-Block* (64) $\rightarrow$ *CE-Block* (32)	2^{17}	27	2^{59}
eMLP-3	*RE-Block* (128) $\rightarrow$ *CE-Block* (64) $\rightarrow$ *RE-Block* (32)	2^{17}	29	2^{59}

Table 1: MLP architectures and encryption parameters. N is the polynomial degree, l the scheme level, and Δ the scaling factor. Hidden layer sizes k_h are shown in parentheses.

Encrypted Weight Update. Once the encrypted gradients for the *CE-FC* local classifier $h^\star$ and the *RE-FC* layer h have been computed, ReBoot updates the corresponding weights using a HE-compatible version of the *Nesterov Accelerated Gradient (NAG)* (Sutskever et al. 2013), which supports both weight decay and momentum. This choice is motivated by the crucial role of these techniques in controlling convergence speed in HE-based training, where each operation introduces additional noise and computational overhead. Let η be the weight decay rate, μ the momentum, and γ the learning rate. The decay-regularized gradient is computed as:

$$\underline{\delta\tilde{\mathbf{W}}}^t \leftarrow \underline{\delta\tilde{\mathbf{W}}}^t +_\bullet \left(\boldsymbol{\eta}_{r\times c} \odot_\circ \underline{\tilde{\mathbf{W}}}^t\right).$$

The encrypted weight update is then computed as:

$$\underline{\boldsymbol{\Delta}\tilde{\mathbf{W}}}^t = \left(\boldsymbol{\gamma}_{r\times c} \odot_\circ \underline{\delta\tilde{\mathbf{W}}}^t\right) +_\bullet \left(\boldsymbol{\gamma\mu}_{r\times c} \odot_\circ \underline{\delta\tilde{\mathbf{W}}}^t\right),$$

for the first iteration $t = 1$, and as:

$$\underline{\boldsymbol{\Delta}\tilde{\mathbf{W}}}^t = \left(\boldsymbol{\gamma}_{r\times c} \odot_\circ \underline{\delta\tilde{\mathbf{W}}}^t\right) +_\bullet \left(\boldsymbol{\gamma\mu}_{r\times c} \odot_\circ \underline{\delta\tilde{\mathbf{W}}}^t\right) +_\bullet \left(\boldsymbol{\gamma\mu^2}_{r\times c} \odot_\circ \underline{\tilde{\mathbf{V}}}^t\right)$$

for subsequent iterations, where $\underline{\tilde{\mathbf{V}}}^t$ denotes the encrypted velocity. The velocity is initialized to the weight gradient at the first iteration and updated at each subsequent step as:

$$\underline{\tilde{\mathbf{V}}}^{t+1} \leftarrow \left(\boldsymbol{\mu}_{r\times c} \odot_\circ \underline{\tilde{\mathbf{V}}}^t\right) +_\bullet \underline{\delta\tilde{\mathbf{W}}}^t,$$

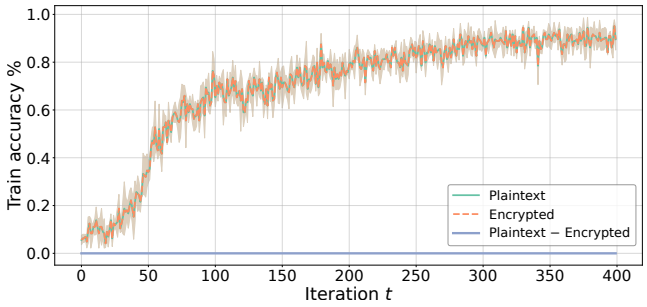
where $\{\boldsymbol{\eta}, \boldsymbol{\mu}, \boldsymbol{\gamma}, \boldsymbol{\gamma\mu}, \boldsymbol{\gamma\mu^2}\} \in \mathbb{R}^{r\times c}$ are plaintext matrices filled with the constants η, μ, γ, $\gamma\mu$, and $\gamma\mu^2$, respectively. These constants are grouped and precomputed in plaintext to avoid additional multiplicative depth during encrypted computation. Finally, the encrypted weights are updated as:

$$\underline{\tilde{\mathbf{W}}}^{t+1} \leftarrow \underline{\tilde{\mathbf{W}}}^t -_\bullet \underline{\boldsymbol{\Delta}\tilde{\mathbf{W}}}^t.$$

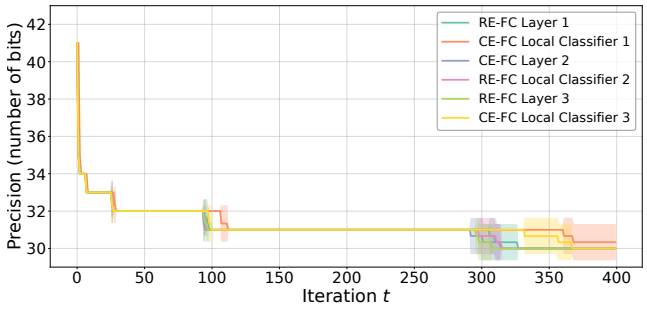
To support further encrypted computations beyond scheme level l, two iterative bootstrapping steps are applied to refresh the ciphertext modulus $\mathbf{q}$ of the updated weights and velocities, as explained in Section 3.

4.4 ReBoot Multiplicative Depth

Properly configuring CKKS encryption parameters requires determining the maximum number of consecutive homomorphic multiplications per training iteration, known as the multiplicative depth τ, which depends on both the MLP architecture and the learning algorithm. Minimizing τ is crucial, as it directly impacts both memory usage and computational overhead. To highlight the efficiency of ReBoot



(a) Plain and encrypted accuracy.



(b) Weights precision per layer.

Figure 2: Precision analysis of *eMLP-3* on MNIST dataset.

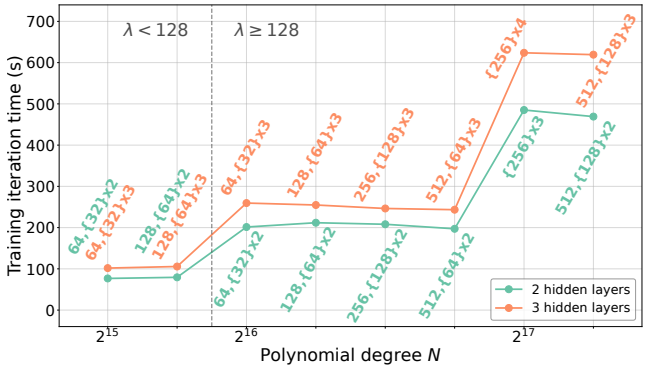


Figure 3: ReBoot training iteration time as a function of polynomial degree N, layer width, and NN depth H.

learning algorithm, we compare its depth to that of standard Back-Propagation (BP). In BP, gradients propagate from the output to the first layer, causing the maximum depth τ_{BP} to occur when updating the first layer $h = 1$. In contrast, ReBoot limits gradient propagation within local-loss blocks. As a result, its maximum depth τ_{ReBoot} is reached at the last layer $h = H$. More specifically, both methods share the same forward pass depth $\tau^{\text{fw}} = \lfloor 2.5H \rfloor + \lfloor 1.5 + (H \bmod 2) \rfloor$, but differ significantly in the backward pass. BP requires a backward pass depth $\tau^{\text{bw}} = \lfloor 2.5H \rfloor + 2$, while ReBoot requires only a depth $\tau^{\text{bw}} = \lfloor 5.5 - (H \bmod 2) \rfloor$. The resulting reduction in maximum multiplicative depth compared to BP can thus be quantified as:

$$\Delta\tau = \tau_{\text{BP}} - \tau_{\text{ReBoot}} = \lfloor 2.5H \rfloor - 3 - (H \bmod 2).$$

Dataset	*eMLP-1*		*eMLP-2*		*eMLP-3*	
	ReBoot	FP32 BP	ReBoot	FP32 BP	ReBoot	FP32 BP
MNIST	94.61 ± 0.27	94.58 ± 0.46	96.39 ± 0.16	94.18 ± 0.32	96.77 ± 0.13	95.03 ± 0.36
Fashion-MNIST	85.12 ± 0.20	85.09 ± 0.22	86.32 ± 0.23	82.27 ± 0.72	86.64 ± 0.28	86.25 ± 5.81
Kuzushiji-MNIST	76.60 ± 0.66	79.41 ± 0.51	82.31 ± 0.53	79.57 ± 0.56	83.83 ± 0.37	82.07 ± 0.51
Breast Cancer	99.03 ± 1.78	98.95 ± 1.70	98.96 ± 1.78	98.25 ± 2.74	99.03 ± 1.88	98.74 ± 2.49
Heart Disease	87.06 ± 5.12	87.62 ± 5.72	93.44 ± 2.64	94.87 ± 3.00	93.25 ± 2.64	94.13 ± 2.06
Letter Recognition	73.08 ± 1.27	75.06 ± 1.77	85.84 ± 0.72	85.66 ± 1.20	90.66 ± 0.90	89.27 ± 0.81

Table 2: Test accuracy comparison across 10 independent runs between ReBoot and FP32-precision plaintext BP training.

As the number of hidden layers H increases, the depth gap $\Delta\tau$ widens, significantly reducing ciphertext noise accumulation and underscoring ReBoot scalability advantage.

5 Experimental Results

In this section, we evaluate both the effectiveness and efficiency of ReBoot compared to existing solutions. The MLP architectures and CKKS encryption parameters Θ are summarized in Table 1. The scheme level is set to $l = \tau_{\text{ReBoot}} + \tau_{\text{bs}}$, where τ_{bs} denotes the *iterative bootstrapping* depth (Bae et al. 2022). Parameters are chosen to ensure 128-bit security, in accordance with the HE Standard (HES) (Albrecht et al. 2019). All experiments were conducted on an Ubuntu 20.04 LTS workstation equipped with two Intel Xeon Gold 5318S CPUs and 384 GB of RAM. Full set of hyperparameters is provided in Appendix D (Pirillo and Colombo 2025).

5.1 Precision Analysis

The first experiment analyzes the impact of the approximate nature of CKKS computations on encrypted training. Each CKKS operation introduces noise that accumulates over time, degrading ciphertext precision. The severity of this effect depends on the scaling factor Δ, which defines the precision used to represent values, and the scheme level l, which sets the number of available operations. Unlike in FHE, CKKS bootstrapping does not eliminates noise, making precision management crucial for encrypted training.

Figure 2a compares the training accuracy (averaged over 3 runs) of ReBoot plaintext and encrypted training on the MNIST dataset using *eMLP-3*. Corresponding results for the other *eMLP* architectures are provided in Appendix C (Pirillo and Colombo 2025). The exact match between the accuracies demonstrates that ReBoot encrypted training faithfully replicates the behavior of its Floating-Point (FP) plaintext counterpart, effectively mitigating CKKS approximation errors and managing noise accumulation. Similarly, Figure 2b analyzes encrypted-weight precision by quantifying the number of matching bits with their plaintext equivalents, following the methodology of Li et al. (2022). The results show a logarithmic decay in precision across layers, with a sharp drop after the first bootstrapping step. However, precision stabilizes in subsequent iterations, with the approximation error remaining consistently on the order of 10^{-11}, enabling ReBoot to sustain effective training.

These findings confirm the functional equivalence between ReBoot encrypted and plaintext training, both in terms of learning dynamics and numerical precision. Given this demonstrated alignment, and considering the significant computational overhead of HE, subsequent ReBoot accuracy experiments are performed using plaintext training to enable a broader experimental campaign.

5.2 ReBoot Test Accuracy

The second experiment compares ReBoot generalization capability to that of standard FP32-precision plaintext BP. We train both *eMLP* models using ReBoot and equivalent plaintext MLPs using BP. The BP-trained models use the NAG optimizer and the *categorical cross-entropy* loss. Table 2 reports average test accuracy over 10 runs. Despite HE constraints, ReBoot consistently achieves accuracy comparable to those of HE-incompatible models trained with BP. Moreover, ReBoot test accuracy improves with increasing model complexity, indicating that its learning algorithm scales effectively with NN width and depth.

5.3 Comparison with SotA Solutions

This experiment evaluates ReBoot effectiveness against SotA encrypted training methods, using the same datasets as prior work (detailed in Appendix D (Pirillo and Colombo 2025)). To ensure a fair comparison, ReBoot was tested using both the original MLP configurations from previous studies and the *eMLP* models, which maintain the same NN depth H but feature increased layer widths k_h. Notably, ReBoot supports the use of wider FC layers without increasing overall computational complexity, as analyzed in Section 5.4, enabling the training of more expressive models. As show in Table 3, ReBoot consistently outperforms prior encrypted DNN approaches across all datasets, achieving improvements of up to $+6.83\%$. Unlike most existing methods, which rely on integer-based training, ReBoot real-valued training enables a higher-precision learning process.

5.4 ReBoot Computational Demand

The last experiment compares the computational demand of ReBoot in terms of training latency against related solutions. Nandakumar et al. (2019) and Lou et al. (2020) adopt an insecure 80-bit security level, which permits the use of a smaller polynomial degree N, thereby reducing computational cost but falling short of HES recommendations. Similarly, Mihara et al. (2020) and Montero et al. (2024) avoid bootstrapping, which accounts for up to 86% of total runtime, by introducing a *re-encryption* mechanism that

Dataset	Work	HE Scheme	λ	Architecture	Test accuracy
MNIST	(Lou et al. 2020)	BFV/TFHE	80	MLP[784-128-32-10]	96.60
	ReBoot	**CKKS (bootstrapping)**	**128**	**MLP[784-128-32-10]**	**97.67 ± 0.13**
C-MNIST	(Nandakumar et al. 2019)	BGV	$\ll 80$	MLP[64-32-16-10]	96.00
	ReBoot	CKKS (bootstrapping)	128	MLP[64-32-16-10]	95.08 ± 0.13
	ReBoot	**CKKS (bootstrapping)**	**128**	***eMLP-2***	**96.71 ± 0.12**
T-MNIST	(Colombo, Falcetta, and Roveri 2024)	TFHE	128	MLP[16-4-2-3]	91.90
	ReBoot	CKKS (bootstrapping)	128	MLP[16-4-2-3]	91.23 ± 0.53
	ReBoot	**CKKS (bootstrapping)**	**128**	***eMLP-2***	**93.39 ± 0.64**
Fashion-MNIST	(Colombo, Falcetta, and Roveri 2024)	TFHE	128	MLP[784-200-10]	86.00
	ReBoot	**CKKS (bootstrapping)**	**128**	**MLP[784-200-10]**	**89.08 ± 0.20**
Iris	(Mihara et al. 2020)	CKKS (leveled)	128	MLP[4-10-3]	98.05
	ReBoot	CKKS (bootstrapping)	128	MLP[4-10-3]	98.44 ± 6.40
	ReBoot	**CKKS (bootstrapping)**	**128**	***eMLP-1***	**99.69 ± 1.88**
Penguins	(Colombo, Falcetta, and Roveri 2024)	TFHE	128	MLP[4-2-3]	92.20
	ReBoot	CKKS (bootstrapping)	128	MLP[4-2-3]	81.18 ± 2.35
	ReBoot	**CKKS (bootstrapping)**	**128**	***eMLP-1***	**99.03 ± 0.68**
Breast Cancer	(Montero et al. 2024)	TFHE (leveled)	128	MLP[30-29-1]	98.25
	ReBoot	CKKS (bootstrapping)	128	MLP[30-29-1]	98.95 ± 1.99
	ReBoot	**CKKS (bootstrapping)**	**128**	***eMLP-1***	**99.03 ± 1.78**

Table 3: Test accuracy comparison between ReBoot and related encrypted training frameworks over 10 runs.

Work	HE Scheme	Architecture	Training iteration time (s)
(Yoo and Yoon 2021)	TFHE	MLP[1-1-1]	1681.20
ReBoot	CKKS	MLP[1-1-1]	198.37
ReBoot	CKKS	*eMLP-1*	190.32
(Colombo, Falcetta, and Roveri 2024)	TFHE	MLP[4-2-3]	646.72
ReBoot	CKKS	MLP[4-2-3]	193.77
ReBoot	CKKS	*eMLP-1*	190.32
(Colombo, Falcetta, and Roveri 2024)	TFHE	MLP[16-4-2-3]	5034.00
ReBoot	CKKS	MLP[16-4-2-3]	621.15
ReBoot	CKKS	*eMLP-2*	629.65

Table 4: Latency comparison between ReBoot and related encrypted training frameworks.

requires client-server interaction, making it unsuitable for *as-a-service* scenarios. Therefore, we focus our comparison on the works of Yoo and Yoon (2021) and Colombo, Falcetta, and Roveri (2024). As show in Table 4, which reports training time per iteration, ReBoot demonstrates a significant performance advantage, achieving speedups ranging from $3.40\times$ to $8.83\times$. This is primarily due to its packing strategy, which fully exploits CKKS *SIMD* capabilities. In contrast, TFHE lack of *batching* results in sequential execution, with $\mathcal{O}(r \times c)$ complexity for a $r \times c$ matrix.

To assess ReBoot scalability, we further analyzed how training latency is influenced by the polynomial degree N, layer width k_h, and NN depth H. Figure 3 shows the latency per encrypted iteration for ReBoot architectures with $H = 2$ and $H = 3$ hidden layers. Experiments with $N = 2^{15}$ and $N = 2^{16}$ adopt a scaling factor $\Delta = 49$, while those with $N = 2^{17}$ adopt $\Delta = 59$. Two key observations emerge. First, doubling N more than doubles latency. Second, for a fixed N, latency remains nearly constant across different layer widths. However, if the layer width exceeds the number of ciphertext slots, a larger N is required, leading to a substantial increase in iteration time. In contrast, increasing the NN depth has only a moderate impact, as it introduces relatively few additional sequential operations.

6 Conclusions

This paper introduced ReBoot, the first framework to enable fully encrypted and non-interactive training of MLPs using CKKS with bootstrapping. ReBoot features an advanced packing technique that significantly reduces both computational and memory overhead by leveraging CKKS *SIMD* capabilities. In addition, it introduces a dedicated HE-compliant architecture and an encrypted learning algorithm that minimize the required multiplicative depth. Future work will focus on extending ReBoot to support the training of more complex DNN architectures, such as convolutional and recurrent NNs, on real-world use cases (Colombo et al. 2024; Tosevski and Gulak 2025).

Acknowledgments

The authors would like to thank Aurora A.F. Colombo for valuable suggestions and support throughout the writing process of the ReBoot paper. This paper is supported by Dhiria s.r.l. and by PNRR-PE-AI FAIR project funded by the NextGeneration EU program.

References

Acar, A.; Aksu, H.; Uluagac, A. S.; and Conti, M. 2018. A Survey on Homomorphic Encryption Schemes: Theory and Implementation. *ACM Comput. Surv.*, 51(4).

Al Badawi, A.; Hoang, L.; Mun, C. F.; Laine, K.; and Aung, K. M. M. 2020. Privft: Private and fast text classification with homomorphic encryption. *IEEE Access*, 8: 226544–226556.

Al Badawi, A.; and Polyakov, Y. 2023. Demystifying bootstrapping in fully homomorphic encryption. *Cryptology ePrint Archive*.

Albrecht, M.; Chase, M.; Chen, H.; Ding, J.; Goldwasser, S.; Gorbunov, S.; Halevi, S.; Hoffstein, J.; Laine, K.; Lauter, K.; Lokam, S.; Micciancio, D.; Moody, D.; Morrison, T.; Sahai, A.; and Vaikuntanathan, V. 2019. Homomorphic Encryption Standard. Cryptology ePrint Archive, Paper 2019/939.

Ali, R. E.; So, J.; and Avestimehr, A. S. 2024. On Polynomial Approximations for Privacy-Preserving and Verifiable ReLU Networks. arXiv:2011.05530.

Badawi, A. A.; Alexandru, A.; Bates, J.; Bergamaschi, F.; Cousins, D. B.; Erabelli, S.; Genise, N.; Halevi, S.; Hunt, H.; Kim, A.; Lee, Y.; Liu, Z.; Micciancio, D.; Pascoe, C.; Polyakov, Y.; Quah, I.; R.V., S.; Rohloff, K.; Saylor, J.; Suponitsky, D.; Triplett, M.; Vaikuntanathan, V.; and Zucca, V. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Paper 2022/915. https://eprint.iacr.org/2022/915.

Bae, Y.; Cheon, J. H.; Cho, W.; Kim, J.; and Kim, T. 2022. META-BTS: Bootstrapping Precision Beyond the Limit. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, 223–234. New York, NY, USA: Association for Computing Machinery. ISBN 9781450394505.

Bergamaschi, F.; Halevi, S.; Halevi, T. T.; and Hunt, H. 2019. Homomorphic training of 30,000 logistic regression models. In *Applied Cryptography and Network Security: 17th International Conference, ACNS 2019, Bogota, Colombia, June 5–7, 2019, Proceedings 17*, 592–611. Springer.

Boemer, F.; Costache, A.; Cammarota, R.; and Wierzynski, C. 2019. nGraph-HE2: A High-Throughput Framework for Neural Network Inference on Encrypted Data. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'19, 45–56. New York, NY, USA: Association for Computing Machinery. ISBN 9781450368292.

Brakerski, Z.; Gentry, C.; and Vaikuntanathan, V. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, 309–325. New York, NY, USA: Association for Computing Machinery. ISBN 9781450311151.

Brakerski, Z.; Langlois, A.; Peikert, C.; Regev, O.; and Stehlé, D. 2013. Classical hardness of learning with errors. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 575–584.

Carpov, S.; Gama, N.; Georgieva, M.; and Troncoso-Pastoriza, J. R. 2019. Privacy-preserving semi-parallel logistic regression training with Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2019/101.

Chase, M.; Chen, H.; Ding, J.; Goldwasser, S.; Gorbunov, S.; Hoffstein, J.; Lauter, K.; Lokam, S.; Moody, D.; Morrison, T.; et al. 2017. Security of homomorphic encryption. *HomomorphicEncryption. org, Redmond WA, Tech. Rep*.

Cheon, J. H.; Kim, A.; Kim, M.; and Song, Y. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*.

Chillotti, I.; Gama, N.; Georgieva, M.; and Izabachène, M. 2018. TFHE: Fast Fully Homomorphic Encryption over the Torus. Cryptology ePrint Archive, Paper 2018/421.

Colombo, A. A.; Colombo, L.; Falcetta, A.; and Roveri, M. 2024. Enhancing Privacy-Preserving Cancer Classification with Convolutional Neural Networks. In *Biocomputing 2025: Proceedings of the Pacific Symposium*, 565–579. World Scientific.

Colombo, L.; Falcetta, A.; and Roveri, M. 2024. Training Encrypted Neural Networks on Encrypted Data with Fully Homomorphic Encryption. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '24, 64–75. New York, NY, USA: Association for Computing Machinery. ISBN 9798400712418.

Colombo, L.; Pittorino, F.; and Roveri, M. 2025. Training multi-layer binary neural networks with random local binary error signals. *Machine Learning: Science and Technology*, 6(3): 035015.

Crockett, E. 2020. A low-depth homomorphic circuit for logistic regression model training. *Cryptology ePrint Archive*.

Drucker, N.; Moshkowich, G.; Pelleg, T.; and Shaul, H. 2022. BLEACH: Cleaning Errors in Discrete Computations over CKKS. Cryptology ePrint Archive, Paper 2022/1298.

Fan, J.; and Vercauteren, F. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144.

Gentry, C. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, 169–178. New York, NY, USA: Association for Computing Machinery. ISBN 9781605585062.

Han, K.; Hong, S.; Cheon, J. H.; and Park, D. 2018. Efficient logistic regression on large encrypted data. *Cryptology ePrint Archive*.

Kim, A.; Song, Y.; Kim, M.; Lee, K.; and Cheon, J. H. 2018. Logistic regression model training based on the approximate homomorphic encryption. *BMC medical genomics*, 11: 23–31.

Lee, S.; Lee, G.; Kim, J. W.; Shin, J.; and Lee, M.-K. 2023. HETAL: efficient privacy-preserving transfer learning with homomorphic encryption. In *International Conference on Machine Learning*, 19010–19035. PMLR.

Li, B.; Micciancio, D.; Schultz-Wu, M.; and Sorrell, J. 2022. Securing Approximate Homomorphic Encryption Using Differential Privacy. In Dodis, Y.; and Shrimpton, T., eds., *Advances in Cryptology – CRYPTO 2022*, 560–589. Cham: Springer Nature Switzerland. ISBN 978-3-031-15802-5.

Lou, Q.; Feng, B.; Charles Fox, G.; and Jiang, L. 2020. Glyph: Fast and Accurately Training Deep Neural Networks on Encrypted Data. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 9193–9202. Curran Associates, Inc.

Mihara, K.; Yamaguchi, R.; Mitsuishi, M.; and Maruyama, Y. 2020. Neural Network Training With Homomorphic Encryption. arXiv:2012.13552.

Montero, L.; Frery, J.; Kherfallah, C.; Bredehoft, R.; and Stoian, A. 2024. Neural Network Training on Encrypted Data with TFHE. *arXiv preprint arXiv:2401.16136*.

Nandakumar, K.; Ratha, N.; Pankanti, S.; and Halevi, S. 2019. Towards Deep Neural Network Training on Encrypted Data. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 40–48.

Nøkland, A.; and Eidnes, L. H. 2019. Training Neural Networks with Local Error Signals. arXiv:1901.06656.

Panzade, P.; Takabi, D.; and Cai, Z. 2024. I can't see it but I can Fine-tune it: On Encrypted Fine-tuning of Transformers using Fully Homomorphic Encryption. arXiv:2402.09059.

Patel, A.; Eickenberg, M.; and Belilovsky, E. 2023. Local learning with neuron groups. *arXiv preprint arXiv:2301.07635*.

Pirillo, A.; and Colombo, L. 2025. ReBoot: Encrypted Training of Deep Neural Networks with CKKS Bootstrapping. *arXiv preprint arXiv:2506.19693*.

Pirillo, A.; Colombo, L.; and Roveri, M. 2024. NITRO-D: Native Integer-only Training of Deep Convolutional Neural Networks. arXiv:2407.11698.

Smart, N. P.; and Vercauteren, F. 2014. Fully homomorphic SIMD operations. *Des. Codes Cryptography*, 71(1): 57–81.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, III–1139–III–1147. JMLR.org.

Tosevski, V.; and Gulak, G. 2025. Large-Scale Recurrent Neural Networks with Fully Homomorphic Encryption for Privacy-Enhanced Speaker Identification. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1–5. IEEE.

Walch, R.; Sousa, S.; Helminger, L.; Lindstaedt, S.; Rechberger, C.; and Trügler, A. 2022. Cryptotl: Private, efficient and secure transfer learning. *arXiv preprint arXiv:2205.11935*.

Yoo, J. S.; and Yoon, J. W. 2021. t-BMPNet: Trainable Bitwise Multilayer Perceptron Neural Network over Fully Homomorphic Encryption Scheme. *Security and Communication Networks*, 2021(1): 7621260.

Zhang, L.; Saito, H.; Yang, L.; and Wu, J. 2022. Privacy-preserving federated transfer learning for driver drowsiness detection. *IEEE Access*, 10: 80565–80574.