Privacy-preserving deep learning with homomorphic encryption: An introduction

Alessandro Falcetta and Manuel Roveri

***Abstract*—Privacy-preserving deep learning with homomorphic encryption (HE) is a novel and promising research area aimed at designing deep learning solutions that operate while guaranteeing the privacy of user data. Designing privacy-preserving deep learning solutions requires one to completely rethink and redesign deep learning models and algorithms to match the severe technological and algorithmic constraints of HE. This paper provides an introduction to this complex research area as well as a methodology for designing privacy-preserving convolutional neural networks (CNNs). This methodology was applied to the design of a privacy-preserving version of the well-known LeNet-1 CNN, which was successfully operated on two benchmark datasets for image classification. Furthermore, this paper details and comments on the research challenges and software resources available for privacy-preserving deep learning with HE.**

***Index Terms*—Deep learning, homomorphic encryption, privacy-preserving computation, convolutional neural network (CNN).**

I. Introduction

Today's world is characterized by information abundance [1]. Thousands of exabytes of data are generated every day [2] by Internet-of-Things systems, mobile devices, social media, and industrial machinery. To extract value from these data, intelligent "data-processing" services have increased in number in recent years, which are based on machine and deep learning and operate on the cloud or in mobile apps [3]. Unfortunately, the processing of data acquired by users, companies, or stakeholders by third-party software services may severely impact privacy when sensitive data are involved (e.g., medical diagnoses, political or personal opinions, and confidential information) [4]. The need to combine privacy with intelligent services sheds light on one of the most relevant scientific and technological challenges of the coming years: *How can software services and mobile apps that provide intelligent functionalities (through machine and deep learning solutions) be designed while guaranteeing the privacy of user data*? This is a crucial question that research has begun to address from several perspectives, including scientific [5], technological [6], [7], and legislative [8]. Table I presents a comparison of the main approaches provided in the literature for integrating privacy constraints with intelligent processing abilities.

Interestingly, among these families of solutions, *homomorphic encryption* (HE) is the only one that guarantees both the ability to process encrypted data as well as to operate without requiring multiple rounds of client-server computation/communication. HE schemes represent a special type of encryption that allows (a set of) operations to be performed on encrypted data. Specifically [9], an encryption function E and its decryption function D are homomorphic w.r.t. a class of functions $\mathcal{F}$ if, for any function $f \in \mathcal{F}$, one can construct a function g such that $f(m) = D(g(E(m)))$ for a set of input m.

TABLE I
Comparison of methodologies for privacy-preserving machine learning.

	Ability to process encrypted data	Processing without the need for multiple rounds of communication
Homomorphic encryption	Yes	Yes
Multi-party computation	Yes	No
Group-based anonymity	No	Yes
Differential privacy	No	Yes

Due to HE's ability to perform operations on encrypted data without multiple rounds of client-server communications, it is particularly suitable for consideration in the "as-a-service" computing paradigm, which requires high standards of privacy and data confidentiality. Indeed, integrating HE with machine and deep learning solutions could lead, for example, to the design of a cloud-based diagnosis system that is able to process X-ray images previously encrypted by a patient. The encrypted results (e.g., an index measuring the presence of potentially critical health threats) would be sent back to the patient, who would be the only one able to decrypt them.

Unfortunately, this ability comes at the expense of three drawbacks: *First*, only a subset of operations (mainly addition and multiplication) is allowed in most of HE-based processing systems; *second*, the length of the processing pipeline (i.e., the amount and type of operations to be executed) is restricted; and *third*, the memory and computational demand of HE-based processing systems are much higher than those of traditional systems.

These three drawbacks are particularly relevant in a scenario where deep learning solutions are considered, since deep learning models are typically characterized by a long pipeline of processing layers that comprises various types of nonlinear operations. For this reason, deep learning models and solutions must be completely redesigned and redeveloped to consider the constraints of HE schemes. Only a few studies have proposed addressing this issue with effective solutions in highly specific fields [10], and a general approach to HE for deep learning is still missing. Therefore, the aim of this study was twofold:

A. Falcetta and M. Roveri are associated with Politecnico di Milano, Italy. Email alessandro.falcetta@polimi.it, manuel.roveri@polimi.it.

TABLE II
SUMMARY OF THE NOTATIONS USED IN THIS PAPER.

Notation	Meaning
m	Raw message
n	Polynomial modulus degree
p	Plaintext coefficient modulus
q	Ciphertext coefficient modulus
Θ	Encryption parameters (n, p, q)
Φ_n	Cyclotomic polynomial $(x^n + 1)$
$R_p = \mathbb{Z}_p[x]/\Phi_n(x)$	Plaintext polynomial ring
$R_q = \mathbb{Z}_q[x]/\Phi_n(x)$	Ciphertext polynomial ring
$\underline{m}$	Encoded message (plaintext)
$\widetilde{m}$	Encrypted message (ciphertext)
$\underline{m} = \Gamma_\Theta(m)$	Encoding
$m = \Gamma_\Theta^{-1}(\underline{m})$	Decoding
k_s, k_p	Secret and public keys
$\widetilde{m} = E_\Theta(\underline{m}, k_p)$	Encryption
$\underline{m} = D_\Theta(\widetilde{m}, k_s)$	Decryption
$\lfloor \dots \rfloor$	Floor operator
$\lceil \dots \rfloor$	Round operator
$[\dots]_{\Phi_n, p}$	Modulo $x^n + 1$, modulo p

- To introduce HE for machine and deep learning by describing HE encoding/encryption mechanisms and the operations on plaintexts/ciphertexts;
- To provide a methodology for the step-by-step design of privacy-preserving convolutional neural networks (CNNs) based on HE. The goal of this methodology is to trace the path in the design of HE-based machine and deep learning solutions for supporting privacy-preserving intelligent processing in cloud-based services or mobile apps.

To achieve these aims, this study complemented theory with examples and code by applying the proposed methodology to the design of a privacy-preserving version of the well-known LeNet-1 CNN [11]. Experimental results are presented regarding the effectiveness and efficiency of the privacy-preserving LeNet-1 on two benchmark datasets for image classification. Furthermore, the research challenges and the software resources available for the design of privacy-preserving deep learning solutions are detailed and commented on. In addition, all of the codes used in this study have been made available to the scientific community as a public repository.[1]

The remainder of this paper is organized as follows. Section II introduces the Brakerski–Fan–Vercauteren (BFV) scheme for HE together with the encoding/decoding mechanisms and privacy-preserving operations. Section III introduces the proposed methodology for the design of privacy-preserving CNNs with HE, and then Section IV details the application of this methodology to the well-known LeNet-1 CNN. Finally, the research challenges and software resources available for privacy-preserving deep learning are presented in Sections V and VI, respectively, before the conclusions of the study are drawn in Section VII.

II. HOMOMORPHIC ENCRYPTION: THE BFV SCHEME

This section illustrates the main characteristics of HE schemes and provides concrete examples of its algebraic peculiarities. An HE scheme is an encryption scheme that supports the computation of a set of operations directly on encrypted data. This goal is achieved thanks to the HE scheme's ability to maintain the algebraic structure of the data during the encrypted processing. In other words, HE allows a party to compute operations between ciphertexts, guaranteeing that the obtained result, when decrypted, will be equal (under certain assumptions, which are detailed as follows) to that obtained by computing the same operations between the corresponding plaintexts.

HE schemes can generally be classified into four categories, which are characterized by increasing complexity in terms of the number and type of operations. First, *partially HE schemes* are HE schemes that can support the homomorphic computation of only one class of operations. An example of a partially HE scheme is Rivest–Shamir–Adleman (RSA) [12]. Second, *somewhat HE schemes* support the homomorphic computation of an unbounded number of additions and a single multiplication. A notable example in this category is the Boneh–Goh–Nissim (BGN) scheme [13]. Third, *leveled HE schemes* are schemes that support the homomorphic computation of a predetermined number of additions and multiplications. The BFV scheme [14] (which is the reference HE scheme considered in this paper) and the Cheon–Kim–Kim–Song (CKKS) scheme [15] belong to this category. Finally, *fully HE schemes* support the homomorphic computation of an unbounded number of operations, often binary ones (AND, NOT...). While the HE schemes in this category provide the greatest flexibility in terms of processing abilities, configuring and managing them is very difficult. An example of a fully HE scheme is Torus–Fully–HE (TFHE) [16]. As previously mentioned, this paper focuses on the BFV scheme [14], which is a popular leveled HE scheme based on the ring learning with errors (RLWE) problem [17].

A deep learning solution that implements the BFV scheme is summarized in Figure 1, where the raw message m (i.e., the message to be processed in an encrypted manner) is a vector of numbers representing, for instance, an image or an audio clip according to the considered deep learning task. The raw message m is initially transformed into an encoded message $\underline{m}$ (called *plaintext*) by means of the *encoding* step $\Gamma_\Theta(\cdot)$ which transforms every number in m into a BFV polynomial. The plaintext $\underline{m}$ is then encrypted into the encrypted message $\widetilde{m}$ (called *ciphertext*) by means of the *encryption* step $E_\Theta(\cdot)$. Encoding and encryption, detailed in Sections II-B and II-C, respectively, depend on the BFV parameters Θ, which are detailed in Section II-A.

The core of the BFV scheme is its ability to support the computation of additions and multiplications between ciphertexts and ciphertexts as well as between ciphertexts and plaintexts. Additions and multiplications in the BFV scheme are detailed in Section II-D. Then, in Section II-E, the "noise budget" is described, which is an integer value that measures the number and type of operations that can be executed while guaranteeing that input data are correctly processed. Table II summarizes notations that appear throughout this paper.

[1] https://github.com/AlexMV12/Introduction-to-BFV-HE-ML

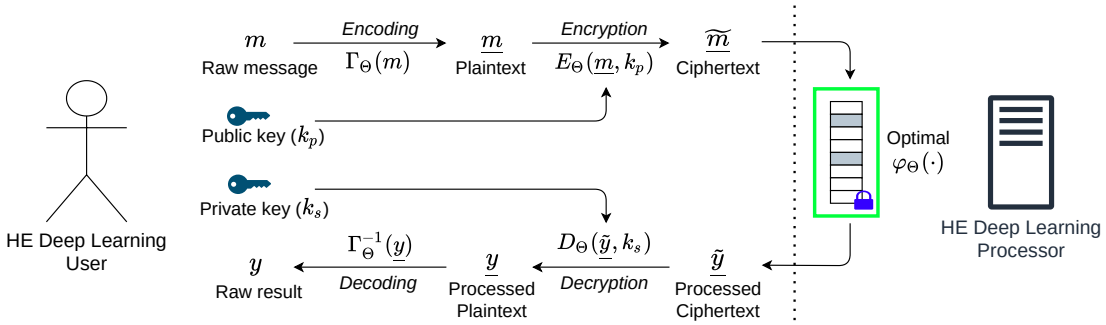


Fig. 1. Machine and deep learning processing chain based on the BFV homomorphic encryption scheme.

A. Encryption parameters of the BFV scheme

The BFV scheme relies on the following set $\Theta = [n, p, q]$ of encryption parameters:

- n: Polynomial modulus degree;
- p: Plaintext coefficient modulus;
- q: Ciphertext coefficient modulus.

As detailed in Section II-B, ciphertexts and plaintexts are represented by polynomials in the BFV scheme; these parameters define the order of the BFV polynomials and the range of values for their coefficients. Specifically, the parameter n must be a positive power of 2 and represents the degree of the cyclotomic polynomial $\Phi_n(x)$. In particular, the polynomial $\Phi_n(x) = x^n + 1$ represents the polynomial modulus. The plaintext modulus p is a positive integer that represents the module of the coefficients of the polynomial ring $R_p = \mathbb{Z}_p[x]/\Phi_n(x)$ (on which the RLWE problem is based). Finally, the parameter q is a large positive integer (larger than p) that results from the product of distinct prime numbers. It represents the modulo of the coefficients of the polynomial ring in the ciphertext space.

The setting of these parameters as well as their effect on the *noise budget* are described and commented on in the rest of the section.

B. Encoding and decoding

Throughout the remainder of this section, raw messages ms are considered unsigned integers. Each number can be transformed into a BFV polynomial by means of the encoding step. Formally, in the BFV scheme the encoding step

$$\underline{m} = \Gamma_\Theta(m)$$

aims to transform m into an n-degree polynomial defined as follows:

$$\underline{m} = c_{n-1}x^{n-1} + \ldots + c_1x^1 + c_0 \tag{1}$$

whose coefficients c_is are modulus p, that is, $c_i \in \mathcal{N} \setminus p$, $i = 0, \ldots, n-1$, where n and p are the polynomial modulus degree and the plaintext coefficient modulus, respectively, as defined in Section II-A. Several methods for encoding numbers into polynomials are available in the literature (e.g., integer encoding and fractional encoding [18]); however, this section focuses on encoding based on *binary representation* [19], which is a widely used encoding mechanism for natural numbers. The basis of the method is the ability to initially encode m into an n-bit binary representation. These n bits are considered the n coefficients $[c_{n-1}, \ldots, c_0]$ of the n-degree polynomial defined in Eq. (1). Hence, $c_i = \{0, 1\}$ with $i = 0, \ldots, n-1$.

Noteworthily, the corresponding *decoding* step $\Gamma_\Theta^{-1}(\underline{m})$ based on binary representation simply refers to the evaluation of the polynomial $\underline{m}$ for $x = 2$:

$$m = \Gamma_\Theta^{-1}(\underline{m}) = \underline{m}(2).$$

For example, consider a BFV scheme with the parameters $n = 16$, $p = 7$ and $q = 874$ and assume that one wishes to encode the following two raw messages: $m_1 = 7$ and $m_2 = 2$. The binary representation of $m_1 = 7$ over $n = 16$ bits is $[0000000000000111]$; hence the corresponding plaintext becomes

$$\underline{m_1} = \Gamma_\Theta(m_1) = x^2 + x + 1.$$

Similarly, when considering $m_2 = 2$, the corresponding plaintext is

$$\underline{m_2} = \Gamma_\Theta(m_2) = x.$$

C. Encryption and decryption

In the BFV scheme, a plaintext $\underline{m}$ is transformed into a ciphertext $\widetilde{\underline{m}}$ by means of the *encryption* step $E_\Theta(\underline{m}, k_p)$, where k_p is the public key. The corresponding *decryption* step $D_\Theta(\widetilde{\underline{m}}, k_s)$ transforms the ciphertext $\widetilde{\underline{m}}$ into a plaintext $\underline{m}$ with the private key k_s.

The public and private keys, which are crucial for the encryption and decryption steps of the BVF scheme, are generated by the user before the encryption phase. Specifically, the secret key k_s corresponds to an n-degree polynomial whose coefficients are randomly selected in the set $\{-1, 0, 1\}$. Given k_s, the public key k_p is a couple (k_{p_0}, k_{p_1}) of n-degree polynomials, which are defined as follows:

$$k_p = (k_{p_0}, k_{p_1}) = \left([-(ak_s + e)]_{\Phi_n, q}, a\right) \tag{2}$$

where a is an n-degree polynomial whose coefficients are randomly selected in the set $\{0, \ldots, q-1\}$ and e is an n-degree polynomial whose coefficients are randomly selected from a discrete and bounded Gaussian distribution $\mathcal{N}(\mu = 0, \sigma = 3.2)$ over the integers [20].

The *encryption* step

$$\widetilde{\underline{m}} = E_{\Theta}(\underline{m}, k_p)$$

aims to transform the plaintext $\underline{m}$ into the ciphertext $\widetilde{\underline{m}}$, which is a couple $(\widetilde{\underline{m_0}}, \widetilde{\underline{m_1}})$ of n-degree polynomials defined as follows:

$$\begin{aligned}\widetilde{\underline{m}} &= (\widetilde{\underline{m_0}}, \widetilde{\underline{m_1}}) = \\ &= \left(\left[k_{p_0} u + e_1 + \left\lfloor \frac{q}{p} \right\rfloor \cdot \underline{m} \right]_{\Phi_n, q}, [k_{p_1} u + e_2]_{\Phi_n, q} \right) \end{aligned} \quad (3)$$

where e_1, e_2 are n-degree polynomials computed as e, while u is an n-degree polynomial computed as k_s. Notably, in Eq. (3), the sum operator "$+$" refers to the sum between polynomials, whereas the operator "$\cdot$" represents the pointwise multiplication between a scalar value and a polynomial (i.e., all of the coefficients of $\underline{m}$ are multiplied by the factor $\left\lfloor \frac{q}{p} \right\rfloor$, which represents the floor of the division between q and p). Note that this factor is larger than one since q is larger than p. The two polynomials representing $\widetilde{\underline{m}}$ are of order n and with coefficients that are modulus q.

Some noise is injected into the ciphertext by e_1, e_2, and u. In particular, u is a "mask" that actually hides the message in the ciphertext. Notably, these noise terms are randomly selected every time the encryption step is activated, and thus, they are responsible for guaranteeing the probabilistic encryption ability of the encryption scheme, which is a relevant property from a security point of view [21].

The decryption step of a ciphertext $\widetilde{\underline{m}}$ operates as follows [14], [19]:

$$D_{\Theta}\left(\widetilde{\underline{m}}, k_s\right) = \underline{m} = \left[\left\lfloor \frac{p}{q} [\widetilde{\underline{m_0}} + \widetilde{\underline{m_1}} k_s]_{\Phi_n, q} \right\rceil \right]_p \quad (4)$$

where $\lfloor \bullet \rceil$ is the round operation. In the decryption phase, $\left[\widetilde{\underline{m_0}} + \widetilde{\underline{m_1}} k_s\right]_{\Phi_n, q}$ is scaled by the factor $\frac{p}{q}$. All of the coefficients are modulus p (after being rounded).

For the sake of clarity, the role of the secret key in the decryption step requires further elaboration. Expanding Eq. (3) w.r.t. the public key k_s leads to:

$$\widetilde{\underline{m}} = \left(\left[-a k_s u - e u + e_1 + \left\lfloor \frac{q}{p} \right\rfloor \cdot \underline{m} \right]_{\Phi_n, q}, [a u + e_2]_{\Phi_n, q} \right).$$

The encoded message $\underline{m}$, multiplied by $\left\lfloor \frac{q}{p} \right\rfloor$, is included in the first term of the ciphertext, suitably hidden by a mask ($-a k_s u$), and perturbed by noise ($-e u + e_1$). The mask is also present in the second term (i.e., $a u$) of the ciphertext together with the noise (e_2). The core of the decryption phase resides in the ability of the user to encrypt $\underline{m}$ by means of k_s and by the fact that k_s can be multiplied for the second term $\widetilde{\underline{m_1}}$ of the ciphertext. Intuitively, multiplying $\widetilde{\underline{m_1}}$ with k_s and summing with $\widetilde{\underline{m_0}}$ removes the mask u from the raw message $\underline{\mathrm{m}}$, as long as the error terms—accumulated during the pipeline of homomorphic operations—are not too big. An example of this encryption/decryption phase is presented in Section II-F.

D. Homomorphic operations in the BFV scheme

The BFV scheme allows the computation of additions and multiplications between ciphertexts, between ciphertexts and plaintexts, and between plaintexts. The addition and multiplications in the BFV scheme are as follows:

- "modulus $x^n + 1$", where the polynomial that is the outcome of the operation is modulus the cyclotomic polynomial $\Phi_n(x) = x^n + 1$;
- "coefficient modulus" p;

where the operands are plaintexts;

- "modulus $x^n + 1$", where the polynomial that is the the outcome of the operation is modulus the cyclotomic polynomial $\Phi_n(x) = x^n + 1$;
- "coefficient modulus" q;

where at least one of the two operands is a ciphertext.

Now, this example continues with $m_1 = 7$ and $m_2 = 2$ considering the addition and multiplication of these two plaintexts in the BFV scheme. In particular, the addition of $m_1 + m_2$ becomes, in polynomial form,

$$\left[\underline{m_1} + \underline{m_2} \right]_{\Phi_{16}, 7} = x^2 + 2x + 1,$$

while the corresponding decoding of $\underline{m_1} + \underline{m_2}$ becomes $\Gamma_{\Theta}^{-1}(\underline{m_1} + \underline{m_2}) = 9$. Similarly, the multiplication of m_1 and m_2 becomes

$$\left[\underline{m_1} * \underline{m_2} \right]_{\Phi_{16}, 7} = x^3 + x^2 + x,$$

while the decoding of $\underline{m_1} * \underline{m_2}$ leads to $\Gamma_{\Theta}^{-1}(\underline{m_1} * \underline{m_2}) = 14$.

Note that, with modulus $x^n + 1$ and p, the following two operations

$$\sum_{i=1}^{7} \underline{m_1}, \quad (5)$$

$$\prod_{i=1}^{16} \underline{m_2} \quad (6)$$

introduce an "overflow" in the processing, thereby leading to incorrect results. In fact, when one of the coefficients of the polynomial becomes equal to or larger than p at the end of the processing, an incorrect result will be obtained due to the modulo p operation. In particular, in Eq. (5), all of the coefficients of the polynomial become 7 at the end of the processing. Hence, when the modulo 7 operation is applied, the coefficients become 0 leading to the following incorrect result:

$$\left[7x^2 + 7x + 7\right]_7 = 0x^2 + 0x + 0.$$

By contrast, in Eq. (6), the problem is related to the modulo Φ_n operation performed on the polynomial. Indeed, the degree of the polynomial becomes 16 during the last multiplication, leading to the following loss of information:

$$\left[x^{16}\right]_{\Phi_{16}} = -1.$$

When an overflow occurs in the processing, the final decrypted value will differ from the correct one. The overflow issue, described here on additions and multiplications between

two plaintexts, also affects the operations that comprise ciphertexts, which are described as follows.

The addition between two ciphertexts (e.g., $\widetilde{m_1}$ and $\widetilde{m_2}$) is as simple as computing their element-wise sum (recall that a ciphertext is a couple of polynomials):

$$\widetilde{m_1} + \widetilde{m_2} = \left(\left[\widetilde{m_{1_0}} + \widetilde{m_{2_0}} \right]_{\Phi_n, q}, \left[\widetilde{m_{1_1}} + \widetilde{m_{2_1}} \right]_{\Phi_n, q} \right). \quad (7)$$

By contrast, the multiplication between two ciphertexts produces a three-term ciphertext:

$$\widetilde{m_1} * \widetilde{m_2} = \left(\left[\left\lfloor \frac{p}{q} \ \widetilde{m_{1_0}} \widetilde{m_{2_0}} \right\rceil \right]_{\Phi_n, q}, \left[\left\lfloor \frac{p}{q} (\widetilde{m_{1_0}} \widetilde{m_{2_1}} + \widetilde{m_{1_1}} \widetilde{m_{2_0}}) \right\rceil \right]_{\Phi_n, q}, \left[\left\lfloor \frac{p}{q} \widetilde{m_{1_1}} \widetilde{m_{2_1}} \right\rceil \right]_{\Phi_n, q} \right). \quad (8)$$

Having a three-term ciphertext is not a problem, given that all of the procedures (decryption, addition, and multiplication) can be modified to work on ciphertexts of arbitrary dimensions. However, the larger the ciphertexts, the larger their memory and computational footprint. To address these problems, the BFV scheme supports an operation called *relinearization* [14] which receives as input a ciphertext with size k and returns a ciphertext with size $k-1$, 2 the minimum dimension allowed. Although relinearization slightly increases the noise in the ciphertext and requires some additional keys, relinearizing the ciphertext after every ciphertext–ciphertext multiplication is generally recommended.

E. Noise budget

As detailed in Eq. (3), the BFV scheme (similar to other HE schemes presented in the literature) injects noise into ciphertexts during the encryption step. This is necessary to guarantee the probabilistic encryption property of the BFV scheme since encrypting the same plaintext through two different activations of the encryption step would lead to two different ciphertexts. The drawback is that, during homomorphic addition and multiplication on the ciphertext, noise is added as well as multiplied. This might lead to a critical scenario where, during processing, one of the coefficients of the ciphertext is rounded to an incorrect value during decryption (as in Eq. (4)), hence failing the decryption phase.

Noise handling is a crucial point in the BFV HE scheme. A correct evaluation of the number (and type) of operations allowed on the ciphertexts is crucial in the design of HE-based processing systems. This is exactly where the noise budget (NB) comes into play. While providing a formal definition of the NB is outside of the scope of this paper (see [19] for details), one can intuitively define it as an indicator of the number of operations that can be performed on a ciphertext before its decryption will fail.

Interestingly, the NB is a property of a ciphertext that varies during the processing pipeline. It is measured as a positive integer and depends on the parameters Θ of the BFV scheme.

TABLE III
NOISE BUDGET (NB) CONSUMPTION, ON THE BASIS OF THE COMPUTED HOMOMORPHIC OPERATION.

Operation	NB consumption
Ciphertext-ciphertext multiplication	High
Ciphertext-ciphertext addition	Medium
Ciphertext-plaintext multiplication	Low
Ciphertext-plaintext addition	Very low

TABLE IV
EXAMPLE OF NB CONSUMPTION, USING TWO DIFFERENT CONFIGURATIONS OF $\Theta = (n, p)$. THE VALUE FOR q IS SET AUTOMATICALLY ACCORDING TO THE SEAL LIBRARY [19].

Ciphertext	Noise Budget $\Theta = (2048, 15162)$	Noise Budget $\Theta = (4096, 151262)$
Freshly encrypted $\widetilde{m_1}$	30	81
Freshly encrypted $\widetilde{m_2}$	30	81
$\widetilde{m_1} + \widetilde{m_2}$	29	80
$\widetilde{m_1} * \widetilde{m_2}$	5	52
$\widetilde{m_1} + m_2$	30	81
$\widetilde{m_1} * m_2$	29	81

The NB is initially allocated to the ciphertext immediately after the encryption step. In general, increasing n will increase the amount of NB available in a freshly encrypted ciphertext. By contrast, increasing p and q will increase NB consumption during homomorphic operations. Identifying the values of Θ that guarantee the correct processing of the ciphertexts while reducing computation and memory complexity is crucial for HE-based systems, particularly for those that implement deep learning solutions. This aspect is addressed in Section III.

HE operations (additions and multiplications) applied on the ciphertext decrease the NB. As presented in Table III, the types of operations and operands significantly affect the amount of reduction to the NB. It is crucial to highlight that the decryption step on ciphertexts must be conducted before the NB reduces to 0; otherwise the decryption will fail. Computing the NB is highly complex, but specific HE tools are available for its estimation (see [19] for details).

Table IV depicts the effects of Θ, operations, and operands on NB consumption. As previously mentioned, the initial NB increases with n, thus increasing the number of subsequent operations that can be computed on the ciphertexts. On the other hand, increasing p will increase the NB consumption of the HE operations.

F. Example

This section presents an example application of the BFV scheme. The basic operations of the scheme are implemented in Python through the scientific computation library NumPy [22]. The code used in this example has been made available to the scientific community in the public repository specified in the Section I.

First, this study defines the encryption parameters Θ of the BFV as follows:

```
n = 16
p = 7
q = 124112
theta = (n, p, q)
```

To create polynomials the function POLY is used; it accepts a dictionary of coefficients, and returns the corresponding NumPy polynomial. NumPy polynomials implement the basic operations of polynomial additions, multiplications, and evaluation among others.

For instance, the cyclotomic polynomial Φ_{16}, which is used in the following operations, is generated as follows:

```
cyclotomic_poly = Poly({0: 1, 16: 1})
print(cyclotomic_poly)
------------------------------------------------

X**16 + 1
```

The function GENERATE_KEYS generates a pair of public and secret keys (k_s, k_p) according to Eq. (2), given Θ.

```
ks, kp = generate_keys(theta)
print("Secret key: " + ks)
print("Public key: " + kp)
------------------------------------------------

Secret key: X**15 - X**14 - X**13 - X**12 + X
    **11 + X**10 - X**9 - X**8 - X**7 + X**6 -
     X**5 + X**4 - X**3 - X**2 + 1

Public key: 61199X**15 + 48133X**14 + ... +
    62895X**2 + 113586X + 92746, 80534X**15 +
    36864X**14 + ... + 27318X**2 + 35006X +
    72552
```

It is now possible to encode and encrypt the two values of $m_1 = 7$ and $m_2 = 2$. An example of addition and multiplication is provided as follows.

First, the two values must be encoded as specified in Section II-B. To achieve this, the function ENCODE_BINARY takes as input an unsigned integer and returns the corresponding polynomial encoded with binary encoding. The DECODE_BINARY function is defined accordingly.

```
m1 = encode_binary(7)
m2 = encode_binary(2)
print("m1: " + m1)
print("m2: " + m2)
------------------------------------------------

m1: X**2 + X + 1
m2: X
```

After the values have been encoded in polynomial form, it is possible to proceed with the encryption. First, the ENCRYPT_POLY function creates a ciphertext $\widetilde{m}$ that starts from an encoded polynomial $\underline{m}$. To this end, random polynomials e_1, e_2, u are generated and the ciphertext is created following Eq. (3).

```
enc_m1 = encrypt_poly(m1, kp, theta)
enc_m2 = encrypt_poly(m2, kp, theta)
print("enc_m1: " + enc_m1)
print("enc_m2: " + enc_m2)
------------------------------------------------

enc_m1: 74404X**15 + 9539X**14 + ... + 27250X
    **2 + 64856X + 51090, 121602X**15 + 100582
    X**14 + ... + 60289X**2 + 80948X + 13677

enc_m2: 14068X**15 + 6425X**14 + ... + 23691X
    **2 + 53327X + 52978, 100091X**15 + 85920X
    **14 + ... + 16044X**2 + 51146X + 117168
```

Notably, after the encryption, the coefficients of the involved polynomials are now modulo q, that is in the range $[0, q)$. The maximum degree of the polynomials is 15, as this work is being conducted in the modulo Φ_{16} space.

It is now possible to sum up the two ciphertexts. The function ADD_CIPHERTEXTS simply computes the element-wise sum of the two ciphertexts while applying the modulus operations, as specified in Eq. (7).

```
enc_sum = add_ciphertexts(enc_m1, enc_m2)
print("Sum ciphertext: " + enc_sum)
------------------------------------------------

Sum ciphertext: 88472X**15 + 15964X**14 + ...
    + 50941X**2 + 118183X + 104068, 97581X**15
     + 62390X**14 + ... + 76333X**2 + 7982X +
    6733
```

Finally, the result of the addition can be decrypted. For this purpose, the function DECRYPT_POLY, implemented according to Eq. (4), can be used as follows:

```
decrypted = decrypt_poly(enc_sum, ks, theta)
decoded = decode_binary(decrypted)

print("Decrypted encoded result: " +
    decrypted)
print("Decoded result: " + decoded))
------------------------------------------------

Decrypted encoded result: X**2 + 2X + 1
Decoded result: 9
```

The result is the same as the result that would have be obtained on plain data. The same holds for multiplications. To this end, the function MUL_CIPHERTEXTS is defined. Note that the function will return a three-terms ciphertext, as depicted in Eq. (8).

```
enc_prod = mul_ciphertexts(enc_m1, enc_m2)
print("Product ciphertext: " + enc_prod)
------------------------------------------------

Product ciphertext: 4616X**15 + 69851X**14 +
    ... + 70424X**2 + 108145X + 97186, 41476X
    **15 + 15425X**14 + ... + 13366X**2 +
    108332X + 94209, 77259X**15 + 32461X**14 +
     ... + 49450X**2 + 5811X + 35650
```

Decryption can now be used as follows:

```
decrypted = decrypt_poly(enc_prod, ks, theta)
decoded = decode_binary(decrypted)

print("Decrypted encoded result: " +
    decrypted)
print("Decoded result: " + decoded))
------------------------------------------------

Decrypted encoded result: X**3 + X**2 + X
Decoded result: 14
```

In this case, the homomorphic multiplication between the two ciphertexts also produces a correct result. This ends the first part of this study, which was the introduction to HE with examples. In the next section, HE will be used for designing privacy-preserving CNNs.

III. Designing privacy-preserving CNNs with homomorphic encryption: a methodology

The aim of this section is to detail the methodology for designing privacy-preserving CNNs using HE, which is the second of the two main contributions of this study. Designing privacy-preserving deep learning solutions based on HE requires one to rethink and redesign deep learning solutions that consider the constraints on the type and number of operations that characterize the BFV scheme. Among the wide range of deep learning solutions, the present paper focuses on CNNs [11], which are the state-of-the-art solution in several application scenarios, such as object detection and sound recognition [23].

A CNN $F(\cdot)$ is a deep neural network with L processing layers $\eta_{\xi_l}^{(l)}$, each of which is characterized by the parameters ξ_l with $l = 1, \ldots, L$. In a privacy-preserving CNN based on HE, the processing layers must comprise only addition and multiplication, and the length L of the processing pipeline must guarantee that the NB is not exhausted during processing.

Figure 2 presents an overview of the proposed methodology for the design of privacy-preserving CNNs based on HE. Let $F(\cdot)$ be the CNN to be encoded with HE. The goal of the methodology is to design a privacy-preserving version $\varphi_\Theta(\cdot)$ of $F(\cdot)$ as well as to identify the configuration of the encryption parameters Θ which will guarantee that the NB does not reach 0 during processing. To achieve these goals, the methodology has three different steps: model approximation, model encoding, and model validation. These three steps are detailed in the following sections.

A. Model approximation

The aim of the *model approximation* is to replace the processing layers of $F(\cdot)$ that do not comply with the BFV scheme with those that rely only on additions and multiplications. Doing so is crucial in a deep learning scenario where processing layers typically comprise division, square root as well as nonlinear activation functions. The output of this step is an approximated model $\varphi(\cdot)$ that comprises processing layers that are HE-compliant.

In more detail, the model approximation step receives as input a CNN $F(\cdot)$ and provides the corresponding approximated model $\varphi(\cdot)$ as output. In such a model each processing layer $\eta_{\hat{\xi}_l}^{(l)}$ comprises only additions and multiplications, and hence, is compatible with the BFV scheme. After this approximation phase, the approximated CNN model must be retrained.

A summary of the possible approximations is presented in Fig. 3 and detailed in the following paragraphs.

1) Pooling layers: Maximum pooling layers use the comparison operator, which is not available in the BFV scheme. To address this problem, various pooling algorithms can be used to replace the maximum pooling. The present authors suggest replacing it with the average pooling available in the BVF scheme since it only requires multiplication between the sum of ciphertexts and a fixed value, which is known a priori (i.e., $\frac{1}{k_w \times k_h}$, with k_w, k_h being the width and height of the pooling kernel, respectively).

2) Normalization layers: Normalization layers cannot be considered in the BFV scheme since it is impossible to compute the mean and standard deviation of encrypted data. By contrast, batch normalization layers are available given that they depend on the values of the data used for training. Such values are computed during the training and can be used during the processing of ciphertexts.

3) Activation functions: The activation functions used in CNNs typically comprise nonlinear functions. The *ReLU* activation function, for instance, cannot be computed because it requires the use of the comparison operator. The same holds for the hyperbolic tangent *tanh* which involves division. This study suggests replacing these nonlinear activation functions with the square activation function $f(x) = x^2$. Such an approximation can be further refined using Taylor polynomial expansions. However, increasing the accuracy of the expansion (and thus using a larger number of polynomials) entails an increase in the number of operations (and thus an increased NB consumption).

B. Model encoding

Once the model has been approximated, it can be encoded through the *model encoding* step, where an encoded approximated model $\varphi_\Theta(\cdot)$ is obtained, whose weights have been encoded according to the BFV scheme and the parameters Θ. Encrypted data can now be processed by $\varphi_\Theta(\cdot)$ to obtain the result of CNN processing. Notably, this result is still encrypted and only the owner of the secret key k_s will be able to decrypt it. The proper setting parameters Θ in HE processing remains an open research. Generally, the selection of the value of these parameters follows a "trial-and-error" approach (see the model validation step in Section III-C). Nevertheless, some guidelines for the setting of Θ are provided as follows.

The most critical encryption parameter is n as it is a relevant parameter for the setting of the initial NB and the computational overhead of the encrypted processing. Generally, values of n smaller than 4096 are able to guarantee a NB sufficient only for very simple machine learning models (typically comprising two or three processing layers at most). From a methodological point of view, n is typically initially set to 4096 and then increased as described in Section III-C.

The parameter p affects NB consumption as well as the precision of the homomorphic operations (i.e., p affects the possibility that some coefficients of the decrypted polynomials are rounded to the incorrect value). Tuning p requires a trial-and-error process; typically, a value of p between 2^{16} and 2^{18} represents a good starting point for exploring the parameters described in Section III-C. The value of q is critical for the security of the scheme; it is suggested to rely on the helper function provided by SEAL [19] (see Section VI for details) to set q according to n and p.

The obtained $\Theta = (n, p, q)$ can be used to encode $\varphi(\cdot)$, thus obtaining the encoded deep learning model $\varphi_\Theta(\cdot)$.

C. Model validation

Once the encoding step is completed, the *model validation* step is activated to evaluate the encoded model $\varphi_\Theta(\cdot)$ from

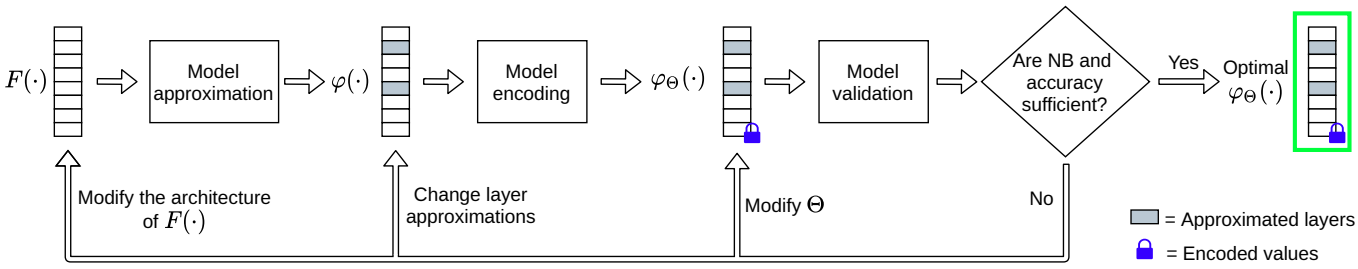


Fig. 2. A methodology for the design of privacy-preserving CNNs based on HE.

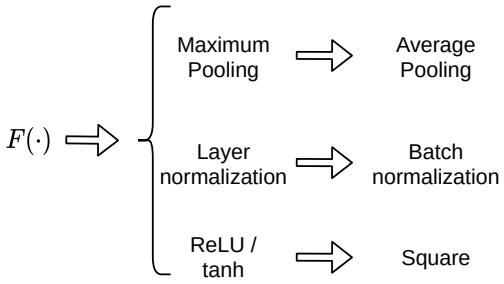


Fig. 3. Possible approximations for typical CNN layers.

two different perspectives. First, $\varphi_\Theta(\cdot)$ is evaluated to check whether the selected configuration Θ provides a sufficient NB in the processing of ciphertexts. Second, the loss in accuracy of $\varphi_\Theta(\cdot)$ w.r.t. $F(\cdot)$ is evaluated. To achieve both goals, a (possibly large) set of raw messages ms is processed by $\varphi_\Theta(\cdot)$ with the aim of measuring the NB of the final ciphertexts and evaluating the discrepancy between the accuracy of the encoded model $\varphi_\Theta(\cdot)$ and that of the plain model $F(\cdot)$. Typically, the problems associated with a loss of NB depend on an incorrect setting of Θ, whereas the discrepancies in the output between $\varphi_\Theta(\cdot)$ and $F(\cdot)$ could be associated with either the approximations of the processing layers introduced in the model approximation step or the fact that p and q are not large enough for the processing pipeline defined in the encoded approximated model $\varphi_\Theta(\cdot)$.

If the constraint on the NB is satisfied and the loss of accuracy is below a user-defined threshold (e.g., 1% or 5%), $\varphi_\Theta(\cdot)$ becomes the privacy-preserving version of $F(\cdot)$ to be considered. Conversely, when either the constraint on the NB is not satisfied (i.e., the NB of the ciphertexts decreases to 0 during the processing of $\varphi_\Theta(\cdot)$) or the loss in accuracy is larger than the threshold, the methodology suggests three different actions: update Θ, modify how layers in $F(\cdot)$ are approximated, or change the processing pipeline of $F(\cdot)$. These three actions are described in detail as follows.

First, the NB and loss of accuracy strictly depend on Θ. In particular, increasing the parameter n increases the initial NB but at the expense of a (potentially large) increase in computational overhead and the memory demand of $\varphi_\Theta(\cdot)$. Conversely, increasing p and q would reduce the loss in accuracy (by increasing the precision of the processing) but at the expense of increased NB consumption by the HE operations.

Second, different model approximations could be considered for the processing layers that are not HE-compliant in $F(\cdot)$. Here, a trade-off must be carefully explored. Indeed, to reduce the loss of accuracy, a coarse-grain layer approximation could be replaced by a finer one (e.g., by using a higher degree of polynomial approximation); however, this would be at the expense of an increased number of operations to be performed for that layer (hence further reducing the NB). On the other hand, moving from a fine-grain layer approximation to a coarse-grain one could reduce NB consumption but possibly increase the loss of accuracy.

Third, if the previous two actions do not succeed in satisfying the constraints on NB and accuracy, a modified version $F'(\cdot)$ of $F(\cdot)$ can be designed. The aim is to reduce the number of operations to be conducted, such as by reducing the number of processing layers or simplifying the operations to be considered. Once $F'(\cdot)$ has been redesigned, the model approximation, encoding, and validation steps are newly activated to identify $\varphi_\Theta(\cdot)$.

Having detailed the proposed methodology, the next section will use it for the design of a privacy-preserving version of the well-known LeNet-1 CNN.

IV. Application to a real-world CNN: privacy-preserving LeNet-1 with BFV

The aim of this section is to detail the application of the methodology proposed in this study to the LeNet-1 CNN, as well as provide numerical results to demonstrate its effectiveness and efficiency.

LeNet-1 is a simple yet effective CNN that was introduced by LeCun et al. [11]. Its processing pipeline $F(\cdot)$ is depicted in Fig. 4(a). LeNet-1 comprises $L = 5$ different processing layers: a convolutional layer with four kernels of size 5 x 5 and tanh activation; an average pooling layer with a kernel of size 2; a convolutional layer with 16 kernels of size 5 x 5 and tanh activation; an average pooling layer with a kernel of size 2; and a fully connected layer of size 192 x 10.

This study applied the methodology described in the previous section to LeNet-1 to design a privacy-preserving version $\varphi_\Theta(\cdot)$ compliant with the BFV scheme. The use of the methodology and the designed model $\varphi_\Theta(\cdot)$ are described in Section IV-A, and then the performance and accuracy of $\varphi_\Theta(\cdot)$ computed on the MNIST [24] and Fashion-MNIST [25] datasets are detailed in Section IV-B. Both datasets are composed of 70,000 28 x 28 gray-scale images (60,000 for training and 10,000 for testing), representing a 10-class classification

problem, that is, 10 digits in MNIST and 10 fashion products in Fashion-MNIST.

A. Applying the methodology to design the privacy-preserving LeNet-1

This section presents the application of the methodology from Section III to the design of the privacy-preserving version $\varphi_\Theta(\cdot)$ of the LeNet-1 CNN $F(\cdot)$. During the model approximation step, this study replaced the tanh activation function of LeNet-1 (involving non-polynomial operations) with the square activation function. Moreover, during the model validation step, this study determined that the NB given by $n = 4096$ was not sufficient for conducting the processing of the encoded model on encrypted data. As mentioned in Section III-B, increasing n would have led to a higher NB but at the expense of a relevant increase in the computational overhead. Thus, this study explored the action of redesigning $F(\cdot)$ by removing the second tanh activation: this change in $F(\cdot)$ led us to consider $F'(\cdot)$, that is, a simplified version of the LeNet-1 CNN without the second tanh activation. Interestingly, $F'(\cdot)$ guaranteed a negligible loss in accuracy w.r.t. $F(\cdot)$ and led to a more effective definition of $\varphi_\Theta(\cdot)$ through the methodology. This aspect will be elaborated on in Section IV-B.

The final configuration of the parameters Θ was as follows:

$$\Theta = \begin{bmatrix} n = 4096 \\ p = 953983721 \\ q = 6.490334708969677435863641547079 68 * 10^{33} \end{bmatrix}.$$

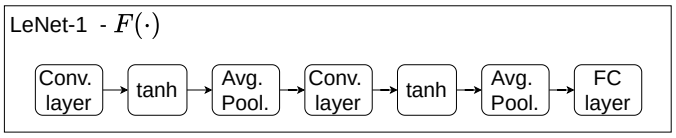
In particular, the value for p was obtained with the trial-and-error procedure described in Section III-B, whereas the value of q was automatically computed by means of the specific SEAL function mentioned in Section III-B.

In summary, the outcome $\varphi_\Theta(\cdot)$ of the methodology representing the privacy-preserving version of LeNet-1 is depicted in Fig. 4(b). The processing pipeline of $\varphi_\Theta(\cdot)$ comprises five different layers: a convolutional layer with four kernels of size 5 x 5 and square activation; an average pooling layer with a kernel of size 2; a convolutional layer with 16 kernels of size 5 x 5; an average pooling layer with a kernel of size 2; and a fully connected layer of size 192x10.

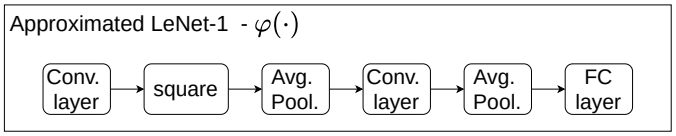
The Python code of the privacy-preserving LeNet-1 $\varphi_\Theta(\cdot)$ is available in the same repository cited in Section I.

B. Performance and accuracy of the privacy-preserving LeNet-1

To demonstrate the effectiveness and efficiency of the designed privacy-preserving version of LeNet-1, we designed an experimental campaign aimed at measuring and comparing the accuracy, memory occupation (in MB) and computation time (in s) of the original LeNet-1 $F(\cdot)$, the simplified LeNet-1 $F'(\cdot)$ (without the second tanh activation), the approximated version $\varphi(\cdot)$ of $F'(\cdot)$, and the outcome $\varphi_\Theta(\cdot)$ of the methodology. The experiments were conducted on a machine with an Intel i7-4770K 64-bit CPU and 16 GB of RAM. The experimental results, computed on the testing images from the MNIST and Fashion-MNIST datasets, are detailed in Table V.



(a)
The original LeNet-1 convolutional neural network



(b)
The privacy-preserving version of the LeNet-1 convolutional neural network

Fig. 4. (a) The original LeNet-1 CNN and (b) its privacy-preserving version based on HE.

TABLE V
EXPERIMENTAL RESULTS ON THE MNIST AND FASHION-MNIST DATASETS. MEMORY OCCUPATION (MB) AND COMPUTATION TIME (S) REFER TO THE PROCESSING OF A SINGLE IMAGE. NO PARALLELIZATION OF THE CODE IS CONSIDERED.

Symbol	Model	Accuracy MNIST	Accuracy Fashion MNIST	Memory Occup.	Comp. Time
$F(\cdot)$	LeNet-1	98.76%	86.88%	7.6MB	0.001s
$F'(\cdot)$	LeNet-1 (single tanh)	98.22%	86.23%	6.5MB	0.0009s
$\varphi(\cdot)$	Approximated $F'(\cdot)$	98.18%	85.29%	6.5MB	0.0009s
$\varphi_\Theta(\cdot)$	$\varphi(\cdot)$ on encrypted data	98.18%	85.29%	780MB	138s

Figure 5 graphically compares the accuracy, execution time, and memory occupation on the Fashion-MNIST dataset. The following three main comments can be made.

First, the simplified version $F'(\cdot)$ of the original LeNet-1 $F(\cdot)$ provided accuracies of 98.22% on MNIST and 86.23% on Fashion-MNIST. The drop in accuracy between $F(\cdot)$ and $F'(\cdot)$ was approximately 0.6% in both datasets. It was therefore crucial to verify whether the drop in accuracy induced by the use of $F'(\cdot)$ (instead of $F(\cdot)$) was acceptable.

Second, as expected, the model approximation step of the procedure, leading from $F'(\cdot)$ to $\varphi(\cdot)$ caused a loss of accuracy. This loss was negligible in the case of the MNIST dataset (0.04%), whereas it was more relevant in the case of the Fashion-MNIST dataset (0.94%). This study speculated that the removal of non-linearities from LeNet-1 has a higher impact when more complex tasks are considered (e.g., the recognition of fashion products instead of simple digits). In addition, the accuracy of $\varphi(\cdot)$ was equal to that of $\varphi_\Theta(\cdot)$ meaning that the privacy-preserving model applied on encrypted images provided the same accuracy as that obtained on plain images. These results confirmed that, thanks to a correct choice of Θ and an accurate approximation of nonlinear layers, the discrepancy between the plain and encrypted results was negligible for the considered scenario.

Third, as expected, encrypted processing introduced a crucial overhead in terms of memory usage and computation time

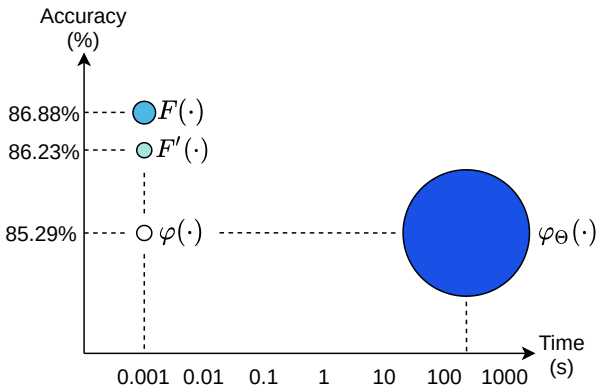


Fig. 5. Time, accuracy, and memory occupation of the different models for the Fashion-MNIST dataset. The dimensions of the circles are representative of the memory occupation (in MB).

for the classification of a single image. Currently, HE libraries do not support parallelization; hence, GPU support for HE is only partially available with few examples present in the literature (e.g., [26]).

This ends the second main contribution of this study, which was the introduction of a methodology for designing privacy-preserving CNNs. The next section will discuss the main challenges to be addressed in this research field.

V. Homomorphic Encryption and Deep Learning: The Challenges

Designing privacy-preserving deep learning solutions is a novel research area with several open research challenges to address. This section, without aiming to be exhaustive, discussed three main challenges in this research area that will be relevant over the next few years:

- **Automatic parameter configuration**: The optimal parameter configuration Θ is currently selected using a trial-and-error approach. The challenge here lies in the study of optimization algorithms, theoretical solutions, and meta-learning algorithms (e.g., AutoML) to provide the optimal configuration of Θ given a processing pipeline $F(\cdot)$ and a reference dataset describing the problem to be addressed.
- **Privacy-preserving recurrent neural networks and transformers**: Currently, the privacy-preserving deep learning solutions in the literature have mostly focused on CNNs, whose transformation in HE-compliant models is easier than other deep learning solutions. One major challenge in this field is the design of privacy-preserving recurrent neural networks and transformers that are able to deal with data sequences. The major issue to be addressed here is the ability to manage the NB in processing pipelines where data are sequentially processed over time.
- **Training privacy-preserving models**: The literature regarding privacy-preserving machine and deep learning models with HE focuses on the inference of privacy-preserving models. The next challenge to address is the training of machine and deep learning models directly on encrypted data. The main issue to be addressed here is to manage the NB consumption not only during the inference, but also during the training of the privacy-preserving model.

Having outlined the main challenges to be addressed in this research area, the next section presents a collection of useful resources for privacy-preserving deep learning with HE.

VI. Available Resources for Privacy-Preserving Deep Learning with Homomorphic Encryption

Two main frameworks for HE, which can be used to facilitate the design of privacy-preserving deep learning solutions, are available in the literature: SEAL and HElib. *SEAL* [19] is a Microsoft C++ library that implements the BFV and CKKS schemes. It offers helper functions for selecting the encryption parameters as well as provides support for basic HE operations (e.g., encrypting and decrypting values). Python users may refer to Pyfhel [27] and TenSEAL [28], which are Python wrappers for SEAL. The code used in the present study relies on SEAL and Pyfhel, while Torch [29] was used for the training of the plain-version of the CNNs. *HElib* [30] is a C++ library that implements the CKKS scheme, among others. HElib also includes optimization mechanisms for efficient homomorphic evaluation, focusing on the effective use of ciphertext packing techniques and Gentry–Halevi–Smart optimizations.

Concrete [31] is a Rust implementation of the TFHE scheme, while a few examples of software libraries specifically intended for HE-based machine and deep learning are available, such as PyCrCNN [32], nGraph-HE [9], and CHET [33].

VII. Conclusions

The aim of this study was to explore the promising but highly challenging research area of privacy-preserving deep learning based on HE. Specifically, the BVF scheme and its privacy-preserving operations were introduced both theoretically and algorithmically through Python code examples. A methodology for designing privacy-preserving CNNs was also proposed, which was applied to the design of a privacy-preserving version of the well-known LeNet-1. Experimental results on two datasets highlighted that it is possible to design privacy-preserving CNNs with HE, which are characterized by a negligible loss in accuracy (w.r.t. the original version) and relevant increases in memory and computational demand. Finally, this paper described the research challenges to be addressed in this field as well as the available software resources for privacy-preserving deep learning.

The path toward privacy-preserving deep learning with HE has now been traced. Over the next few years, great advances will be made in this direction.

Acknowledgment

This work was partially funded by the CATCH 4.0 project within the Italian *Programma Operativo Nazionale* (PON) “Imprese e competitività” FESR 2014/2020, and by Dhiria S.r.l., a spin-off of Politecnico di Milano.

REFERENCES

[1] L. Cai and Y. Zhu, "The challenges of data quality and data quality assessment in the big data era," *Data science journal*, vol. 14, 2015.

[2] S. Cass, "The age of the zettabyte cisco: the future of internet traffic is video [dataflow]," *IEEE Spectrum*, vol. 51, no. 3, pp. 68–68, 2014.

[3] Y. Yao, Z. Xiao, B. Wang, B. Viswanath, H. Zheng, and B. Y. Zhao, "Complexity vs. performance: empirical analysis of machine learning as a service," in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 384–397.

[4] E. P. Council of European Union, "Regulation (eu) no 2016/679, article 4(1)," 2016.

[5] B. C. Stahl and D. Wright, "Ethics and privacy in ai and big data: Implementing responsible research and innovation," *IEEE Security & Privacy*, vol. 16, no. 3, pp. 26–33, 2018.

[6] H. C. Tanuwidjaja, R. Choi, S. Baek, and K. Kim, "Privacy-preserving deep learning on machine learning as a service—a comprehensive survey," *IEEE Access*, vol. 8, pp. 167 425–167 447, 2020.

[7] B. Pulido-Gaytan, A. Tchernykh, J. M. Cortés-Mendoza, M. Babenko, G. Radchenko, A. Avetisyan, and A. Y. Drozdov, "Privacy-preserving neural networks with homomorphic encryption: C hallenges and opportunities," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1666–1691, 2021.

[8] E. P. Council of European Union, "Regulation laying down harmonised rules on artificial intelligence (artificial intelligence act), com/2021/206," 2021.

[9] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "ngraph-he: a graph compiler for deep learning on homomorphically encrypted data," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, 2019, pp. 3–13.

[10] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International conference on machine learning*. PMLR, 2016, pp. 201–210.

[11] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[12] L. Morris, "Analysis of partially and fully homomorphic encryption," *Rochester Institute of Technology*, pp. 1–5, 2013.

[13] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," in *Theory of cryptography conference*. Springer, 2005, pp. 325–341.

[14] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[15] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *Advances in Cryptology – ASIACRYPT 2017*, ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 409–437.

[16] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast Fully Homomorphic Encryption Over the Torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, Jan. 2020.

[17] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2010, pp. 1–23.

[18] S. S. Sathya, P. Vepakomma, R. Raskar, R. Ramachandra, and S. Bhattacharya, "A review of homomorphic encryption libraries for secure computation," *arXiv preprint arXiv:1812.02428*, 2018.

[19] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library-seal v2. 1," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 3–18.

[20] M. R. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. E. Lauter *et al.*, "Homomorphic encryption standard." *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 939, 2019.

[21] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of computer and system sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[22] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[23] M. S. Hossain and G. Muhammad, "Cloud-assisted speech and face recognition framework for health monitoring," *Mobile Networks and Applications*, vol. 20, no. 3, pp. 391–399, 2015.

[24] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[25] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[26] T. Morshed, M. M. Al Aziz, and N. Mohammed, "Cpu and gpu accelerated fully homomorphic encryption," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 142–153.

[27] M. O. Alberto Ibarrondo, Laurent Gomez, "Pyfhel: Python for homomorphic encryption libraries," https://github.com/ibarrond/Pyfhel, 2018.

[28] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal, "Tenseal: A library for encrypted tensor operations using homomorphic encryption," 2021.

[29] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, pp. 8026–8037, 2019.

[30] S. Halevi and V. Shoup, "Algorithms in helib," in *Annual Cryptology Conference*. Springer, 2014, pp. 554–571.

[31] I. Chillotti, M. Joye, D. Ligier, J.-B. Orfila, and S. Tap, "Concrete: Concrete operates on ciphertexts rapidly by extending tfhe," in *WAHC 2020–8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, vol. 15, 2020.

[32] S. Disabato, A. Falcetta, A. Mongelluzzo, and M. Roveri, "A privacy-preserving distributed architecture for deep-learning-as-a-service," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[33] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "Chet: an optimizing compiler for fully-homomorphic neural-network inferencing," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 142–156.