

Article

A Comparative Study of Secure Outsourced Matrix Multiplication Based on Homomorphic Encryption

Mikhail Babenko ^{1,*}, Elena Golimblevskaia ², Andrei Tchernykh ^{3,4,*}, Egor Shiriaev ¹, Tatiana Ermakova ⁵,
Luis Bernardo Pulido-Gaytan ³, Georgii Valuev ¹, Arutyun Avetisyan ⁴ and Lana A. Gagloeva ⁶

¹ North-Caucasus Center for Mathematical Research, North-Caucasus Federal University, 355017 Stavropol, Russia

² Computer Science Department, University of Potsdam, 14469 Potsdam, Germany

³ Computer Science Department, CICESE Research Center, Ensenada 22800, Mexico

⁴ Control/Management and Applied Mathematics, Ivannikov Institute for System Programming, 109004 Moscow, Russia

⁵ School of Computing, Communication and Business, Hochschule für Technik und Wirtschaft (University of Applied Sciences for Engineering and Economics), 10318 Berlin, Germany

⁶ Informatics and Computer Engineering Department, South Ossetia State University, 100001 Tskhinvali, Russia

* Correspondence: mgbabenko@ncfu.ru (M.B.); tchernykh@cicese.edu.mx (A.T.)

Abstract: Homomorphic encryption (HE) is a promising solution for handling sensitive data in semi-trusted third-party computing environments, as it enables processing of encrypted data. However, applying sophisticated techniques such as machine learning, statistics, and image processing to encrypted data remains a challenge. The computational complexity of some encrypted operations can significantly increase processing time. In this paper, we focus on the analysis of two state-of-the-art HE matrix multiplication algorithms with the best time and space complexities. We show how their performance depends on the libraries and the execution context, considering the standard Cheon–Kim–Kim–Song (CKKS) HE scheme with fixed-point numbers based on the Microsoft SEAL and PALISADE libraries. We show that Windows OS for the SEAL library and Linux OS for the PALISADE library are the best options. In general, PALISADE-Linux outperforms PALISADE-Windows, SEAL-Linux, and SEAL-Windows by 1.28, 1.59, and 1.67 times on average for different matrix sizes, respectively. We derive high-precision extrapolation formulas to estimate the processing time of HE multiplication of larger matrices.

Keywords: CKKS; homomorphic encryption; homomorphic encryption standard; matrix multiplication; PALISADE; SEAL



Citation: Babenko, M.; Golimblevskaia, E.; Tchernykh, A.; Shiriaev, E.; Ermakova, T.; Pulido-Gaytan, L.B.; Valuev, G.; Avetisyan, A.; Gagloeva, L.A. A Comparative Study of Secure Outsourced Matrix Multiplication Based on Homomorphic Encryption. *Big Data Cogn. Comput.* **2023**, *7*, 84. <https://doi.org/10.3390/bdcc7020084>

Academic Editor: Min Chen

Received: 8 April 2023

Revised: 20 April 2023

Accepted: 24 April 2023

Published: 28 April 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Third-party services offer a convenient alternative to build, complement, or extend their infrastructures. They can provide convenient data access, unlimited storage, and processing capacities [1]. However, storing and processing sensitive data (e.g., medical records) requires selecting a third-party provider with a high level of data protection. To provide this type of protection, various techniques are used, including homomorphic encryption (HE), which can process encrypted data [2–31]. In addition, homomorphic encryption is part of post-quantum cryptography (PQC), which aims to avoid the risk of attacks by quantum computers. There are several software implementations of PQC methods with hardware acceleration of cryptographic primitives [32–34]. In this study, we focus on the software implementation of matrix multiplication using Microsoft libraries SEAL and PALISADE under Windows and Linux operating systems.

HE defines a class of encryption techniques that allow performing mathematical operations on encrypted data, generating results that correspond to the results of operations on the plaintext, without information about the secret key and access to the raw data [35].

Initially, Partially Homomorphic Encryption (PHE) and Somewhat Homomorphic Encryption (SHE) techniques offered only a limited number of operations, only one type, or a predetermined set. This limited its applicability to a small range of problems. Fully Homomorphic Encryption (FHE) brought another breakthrough by allowing unrestricted addition and multiplication operations on encrypted data.

The main idea of FHE can be presented as follows. Let m_k be a plaintext and ek the encryption key. Then, $Enc(ek, m_k)$ defines the encryption of m_k via HE with ek and the encryption function Enc . Thus, FHE operations can be represented as

$$Dec(Enc(m_1) \otimes Enc(m_2)) = m_1 * m_2, \quad (1)$$

$$Dec(Enc(m_1) \oplus Enc(m_2)) = m_1 + m_2, \quad (2)$$

where Dec is a decryption function, $\otimes$ defines homomorphic multiplication, and $\oplus$ is homomorphic addition. Thus, these schemes are homomorphic over both operations.

Not all operations can be easily performed in the HE domain. In particular, many non-modular and matrix operations are time consuming [36]. This is especially true for matrix multiplication, a common operation in many algorithms [37–41].

Matrix multiplication is a fundamental operation for data processing. It is used in a wide range of data processing and analysis algorithms, including principal component analysis, linear regression, image processing, and neural networks [42–45].

Multidimensional packing is a technique used in HE to pack multiple values into a single ciphertext, which can then be homomorphically processed [46]. This technique can be used to perform approximate matrix arithmetic by packing each row of a matrix into a single ciphertext.

To pack a matrix using multidimensional packing, we must first choose a packing factor that determines how many values are packed into each ciphertext. Then, each row of the matrix is divided into blocks of the size of the packing factor and each block is packed into a single ciphertext.

Once the matrix is packed, we can perform approximate matrix arithmetic by homomorphically processing the ciphertexts. For example, to perform a matrix addition, we can add the corresponding ciphertexts element by element. Similarly, we can perform matrix multiplication by first packing the second matrix and then performing a series of homomorphic operations to compute the product.

It is important to note that some approximation errors may occur in multidimensional packing, since the packed values may not be exact representations of the original matrix elements. However, these errors can be controlled by choosing an appropriate packing factor and carefully selecting the parameters of the HE scheme [47].

In this paper, we provide for the first time a detailed analysis of the practical applicability of current HE matrix multiplication algorithms, discuss bottlenecks and further directions for their efficient implementation. We examine the key technical and theoretical aspects that distinguish algorithms and libraries. We then present performance benchmarks and the main use cases.

We analyze efficiency as a function of several factors: algorithm, implementation, programming language, operating system (OS), and HE scheme.

The main contributions of the paper can be summarized as follows.

- We provide a detailed analysis of the state of the art in HE matrix multiplication algorithms with fixed-point numbers.
- We compare implementations of algorithms with the best time and space complexity based on the Microsoft SEAL [48] and PALISADE [49] libraries.
- We evaluate the impact of different operating systems and libraries on their performance.
- We apply curve fitting to derive high-precision extrapolation formulas for homomorphic multiplication of larger matrices.

We consider space complexity in the analysis of the size of axillary cipher matrices, time complexity, and multiplicative depth. The implementations are based on C++ to ensure comparability between the analysis and the Cheon–Kim–Kim–Song (CKKS) method [48]. CKKS is a pioneer in enabling approximate computations over fixed-point numbers, which are critical for machine learning and deep learning applications (see Appendix A).

There are three reasons for the selection of the two FHE libraries. First, they have been continuously developed and adapted to the changing needs in the field. Second, they work with the promising scheme of CKKS and allow the implementations to run on multiple operating systems. Third, for comparability reasons, they offer the possibility of implementing algorithms in a single programming language.

Large cloud service providers (CSPs) such as Amazon EC2 and Google Cloud typically offer computing services that run on Linux and Windows operating systems. Therefore, we analyze the performance of our implementations on both operating systems.

The paper is organized as follows. Section 2 reviews related work in this area. Section 3 describes the current HE libraries and their features: programming language, supporting numbers, HE schemes, etc. Section 4 describes state-of-the-art secure matrix multiplication methods and their comparison. Section 5 presents the experimental analysis. Section 6 presents an extrapolation to estimate the execution time of a homomorphic matrix multiplication of arbitrary size. Section 7 discusses the use of HE to implement privacy-preserving matrix operations. Finally, Section 8 summarizes the main results, their implications, and further research directions.

2. Related Work

2.1. Privacy-Preservation in Deep Learning

Privacy preservation in deep learning includes techniques and methods to ensure that individuals' sensitive information, such as personal information, financial information, and medical information, is not exposed or compromised during the training or use of deep learning models.

There are several techniques for maintaining privacy in deep learning, including:

Differential Privacy: This is a mathematical framework that adds noise to data before they are processed by a deep learning model. This ensures that the model cannot learn individual-level information, thus preserving the privacy of the data [50].

Federated Learning: This technique allows training a deep learning model on decentralized data. Instead of collecting all the data in a centralized location, the data are kept locally and only the model parameters are shared between devices or nodes [51]. This approach ensures that the data are not exposed and individual privacy is preserved.

Homomorphic Encryption: This is a technique that allows computations to be performed on encrypted data without decrypting them. This technique ensures that the data remain encrypted throughout the computation process, thus preserving the privacy of the data [2–11].

Secure Multi-Party Computation: This is a technique that allows multiple parties to participate in a computation without revealing their input [52]. This approach ensures that the data remain private during the computation process.

Overall, privacy preservation is critical in deep learning, especially for applications involving sensitive data. The above techniques and methods can be used to ensure that the privacy of individuals is maintained when using deep learning models.

2.2. Matrix Multiplication in Privacy-Preserving Neural Networks

Matrix operations are commonly used in the development of privacy-preserving neural networks (PPNNs) for various tasks such as image recognition, natural language processing, and speech recognition. Some of the key places where matrix operations are used in PPNN are:

Data preparation: Data are typically represented as a matrix in PPNN, where each row of the matrix represents an input example and each column represents a feature of the example.

Weight initialization: In a PPNN, the weights connecting neurons in different layers are typically initialized as random matrices.

Forward propagation: In forward propagation, the input matrix is multiplied by the weight matrix of the first layer, an activation function is applied to the resulting matrix, and the result is passed to the next layer [53].

Backpropagation: In the backpropagation process, the gradients of the loss function are calculated with respect to the weights of the PPNN, which is usually performed using matrix calculus [54].

Gradient descent: The weights of the network are updated using an optimization algorithm such as gradient descent, where the gradients are multiplied by a learning rate and the result is subtracted from the current weights [55].

Convolutional layers: In convolutional neural networks (CNNs), the convolutional operation is performed using matrix multiplication between the input and the filter kernel [56].

Overall, matrix operations are an essential part of neural network development and training, as they enable efficient computation of complex mathematical operations with large amounts of data.

3. Homomorphic Encryption Libraries

Rivest, Adleman, and Dertuzos [57] published the initial efforts to construct a homomorphic cipher in 1978. This work provided an essential theoretical foundation for HE. After several decades, the construction of an efficient homomorphic cipher that can be used in practice is still an open question [58].

The first FHE scheme was proposed by Craig Gentry in 2009 [59]. Several modifications have been made to the existing FHE schemes and new schemes have been proposed. For instance, Martin Van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan [30] developed a simpler FHE approach based on [50]. This FHE uses integer arithmetic instead of ideal lattice calculations.

Some of these schemes are the basis of the Homomorphic Encryption Standard [60]: The Brakerski–Gentry–Vaikuntanathan (BGV) scheme developed in 2011 [6], the scale-invariant Brakerski/Fan-Vercauteren (BFV) and the Nth-degree TRUncated polynomial ring (NTRU)-based López–Tromer–Vaikuntanathan (LTV) schemes, which was developed in 2012 [61–63], the Gentry–Sahai–Waters (GSW) method and the YASHE method, which appeared in 2013 [63,64], and CKKS scheme, which was developed in 2017.

Despite the progress in this field, researchers are still trying to solve performance and memory problems to make HE technique mature for real applications [12–15,65–68]. This situation also affects matrix operations [33,69,70], especially matrix multiplication for privacy-preserving machine and deep learning applications [36,71–74].

Advances in theoretical foundations have been followed by several open-source implementations of FHE methods. These libraries provide support for various HE schemes, operations, and data types. The choice of a particular library depends on several factors, and understanding its properties facilitates the choice.

Table 1 provides an overview of the HE libraries. It contains the library name, the number type, the development language, the supported operating system, and the physical resources used, such as the central processing unit (CPU) and the graphics processing unit (GPU).

The Homomorphic Encryption Library (HElib) [69] is an open-source software library developed by IBM in 2013. It supports the integer BGV method with bootstrapping and the CKKS method for approximate value arithmetic.

Simple Encrypted Arithmetic Library (SEAL) [48] is a library developed by Microsoft in 2015. It supports the BFV scheme for working with integers and the CKKS scheme for working with fixed-point numbers. SEAL implements most of the operations associated with HE, including encoding/decoding with single and vector inputs. Homomorphic arithmetic operations are also available. However, the library is not capable of performing operations with matrices.

PALISADE [49] is an open-source lattice crypto software library developed by the New Jersey Institute of Technology (NJIT) in 2017. It implements schemes such as BGV, BFV, CKKS, FHEW, and the TFHE variant that includes bootstrapping.

Homomorphic Encryption for Approximate Numbers (HEAAN) [58] is an open source HE library developed by Seoul National University (SNU). It was developed to implement the CKKS method, and its first version was released in 2016. The multiplication is accelerated by using Fast Fourier Transform (FFT) and Number Theoretic Transform (NTT).

Fastest Homomorphic Encryption in the West (FHEW) [75] is an open-source library developed by the Defense Advanced Research Projects Agency (DARPA) and the University of California (UCSD). It is based on the Fastest Fourier Transform in the West (FFTW) library, which was released in 2017 [76].

TFHE: Fast Fully Homomorphic Encryption over the Torus [77] is an open source HE library released in 2017. It is based on a ring variant of the GSW method. A parallel implementation of the TFHE method called NuFHE was released in 2019 [78]. NuFHE provides GPU acceleration using Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL).

Lattigo [79] is a library in Go that implements HE based on Ring Learning With Errors (R-LWE). It was developed in 2019 and supports BFV and CKKS methods. It was developed using the residue number system (RNS).

$\Lambda\circ\lambda$ (“LOL”, or Lattice Cryptography Library) [80] is a Haskell library released in 2016. It focuses on functional lattice cryptography and implements the BGV scheme. $\Lambda\circ\lambda$ has the ability to integrate specialized backends (e.g., GPUs).

CUDA Homomorphic Encryption Library (cuHE) [81] is a GPU-accelerated library for HE. It implements the Doröz-Hu-Sunar (DHS) SHE method [82], which is based on the LTV method. This library was released in 2016 and has not been updated.

Concrete [83] is an FHE library created in 2020 that implements the variant of Zama’s TFHE scheme [77]. It was developed using the fast and secure programming language Rust.

cuFHE [84] is a library developed in 2018. It implements the TFHE scheme on CUDA-enabled GPUs. cuFHE reports a $26\times$ speedup in gate-by-gate bootstrapping performance compared to the TFHE library for the CPU version.

cuYASHE [85] is an open-source library developed in 2016 by the University of Campinas. It implements the CUDA-accelerated version of the YASHE process [86]. The authors report a 6- to 35-fold improvement in polynomial multiplication compared to implementations on CPU, GPU, and FPGA.

Node-seal [86] is a version of the Microsoft SEAL library adapted for TypeScript or JavaScript, and was first released in 2019. Node-seal provides the fastest web implementation that works in any server/client configuration.

Python for Homomorphic Encryption Libraries (Pyfhel) [87] provides functionalities of FHE libraries in Python. The current version supports only Microsoft SEAL. The library was developed based on the abstraction for homomorphic encryption libraries (Afhel).

Table 1. Main properties of most common HE libraries.

Library	Numbers with Fixed-Point		Integer Numbers						Language	OS		CPU	GPU	Ref.
	CKKS	TFHE	BFV	BGV	LTV	DHS	FHEW	YASHE		Windows	Linux			
Microsoft SEAL	•		•						C++	•	•	•		[48]
PALISADE	•	•	•	•			•		C++	•	•	•		[49]
HEAAN	•								C++		•	•		[58]
cuYASHE								•	C++		•		•	[85]
HElib	•			•					C++		•	•		[69]
FHEW							•		C++		•	•		[75]

Table 1. Cont.

Library	Numbers with Fixed-Point		Integer Numbers						Language	OS		CPU	GPU	Ref.
	CKKS	TFHE	BFV	BGV	LTV	DHS	FHEW	YASHE		Windows	Linux			
TFHE		•							C++		•	•		[77]
NuFHE		•							Python	•	•		•	[78]
Lattigo	•		•						Go	•	•	•		[79]
$\Lambda\lambda$				•					Haskell	•	•	•	•	[80]
cuHE					•	•			C++	•	•		•	[81]
Concrete		•							Rust		•	•		[83]
cuFHE		•							C++		•		•	[84]
node-seal	•		•						TypeScript	•	•	•		[86]
Pyfhel	•		•						Python	•	•	•		[87]
SEAL-python	•		•						Python	•	•	•		[88]

SEAL-python [88] is a header-only library that allows the use of Microsoft's SEAL library in Python. It was developed in 2020 by the Cryptography Research Group at Microsoft. SEAL-python can include source code in a Docker image.

Not all libraries provide support for Windows and Linux operating systems. This is especially important when performing HE operations on a third-party infrastructure, such as a CSP's infrastructure.

Amazon EC2, one of the leading CSPs, supports six different virtual machine (VM) configurations with Linux operating systems and four configurations when running a Windows operating system. Similarly, Google Cloud offers seven different configurations of VMs running a Linux family operating system and only two configurations running a Windows family operating system.

A library that is compatible with both operating systems increases interoperability and facilitates transfer between CSPs and services (VM types).

Therefore, in our experimental setup, we consider probably the most practical operating systems from the Linux and Windows families.

4. Secure Matrix Multiplication

Homomorphic matrix computation is a fundamental operation for statistical analysis and privacy-preserving machine learning. The algorithms proposed by Halevi and Shoup [69] and Jiang et al. [70] are currently the best state-of-the-art algorithms.

The algorithm of Halevi and Shoup [69] is based on a sequence of matrix-vector multiplications. It encodes each vector of the matrix as plaintext, i.e., a plaintext vector is created. Then, the encrypted matrix is encrypted as a vector of ciphertexts. Finally, the vector of ciphertexts is encrypted into a single ciphertext. The operations are performed with this ciphertext.

In matrix-vector multiplication, the input matrices are encoded in their diagonal representation, i.e., each diagonal is encoded into a ciphertext.

Let a matrix A of size $d \times d$ be given by $a_0, \dots, a_{d-1}$, where $a_i = (A_{0,i}, A_{1,i+1}, \dots, A_{d-1,d+i-1})$. Therefore, $a_i[j] = A_{j,j+i}$. The product $w = vA$, where v is the input vector, can be calculated as $w \leftarrow \sum_{i=0}^{n-1} a_i \times (v \lll i)$.

This method requires d rotations, multiplications, and additions. The multiplicative depth is 1.

The algorithm of Jiang et al. [70] is based on the linear transformation of square matrices.

For a matrix $U \in \mathcal{R}^{n \times n}$, a linear transformation $L : \mathcal{R}^n \rightarrow \mathcal{R}^n$ can be represented as $L : m_e \mapsto U \cdot m_e$. Thus, the multiplication of a matrix by a vector can be represented by combining the operations of rotation and multiplication by a constant.

For $0 \leq \ell < n$, ℓ -th -th diagonal vector U can be determined as

$$\begin{aligned} u_\ell &= (U_{0,\ell}, U_{1,\ell+1}, \dots, U_{n-\ell-1,n-1}, U_{n-\ell,0}, \dots, U_{n-1,\ell-1}) \in \mathcal{R}^n \\ U \cdot m_e &= \sum_{0 \leq \ell < n} (u_\ell \odot \rho(m_e; \ell)), \end{aligned} \quad (3)$$

where $\odot$ is a component-wise multiplication between vectors.

$A = (A_{ij})_{0 \leq i,j < d}$ is a matrix of size $d \times d$. The permutations of σ, τ, φ and ψ on the set $\mathcal{R}^{d \times d}$ are defined as follows.

- $\sigma(A)_{i,j} = A_{i,i+j}$.
- $\tau(A)_{i,j} = A_{i+j,j}$.
- $\varphi(A)_{i,j} = A_{i,j+1}$.
- $\psi(A)_{i,j} = A_{i+1,j}$.

The matrix multiplication can be specified by the following formula:

$$A \cdot B = \sum_{k=0}^{d-1} \left(\varphi^k \circ \sigma(A) \right) \odot \left(\psi^k \circ \tau(B) \right), \quad (4)$$

where $\circ$ denotes the function composition.

The multiplication algorithm requires determining the matrix representations corresponding to permutations: U^σ , U^τ , V^k , and W^k . For $0 \leq i, j < d$, $1 \leq k < d$, and $0 \leq \ell < d^2$:

$$U_{d \cdot i + j, \ell}^\sigma = \begin{cases} 1 & \text{if } \ell = d \cdot i + [i + j]_d, \\ 0 & \text{otherwise} \end{cases}, \quad (5)$$

$$U_{d \cdot i + j, \ell}^\tau = \begin{cases} 1 & \text{if } \ell = d \cdot [i + j]_d + j, \\ 0 & \text{otherwise} \end{cases}, \quad (6)$$

$$V_{d \cdot i + j, \ell}^k = \begin{cases} 1 & \text{if } \ell = d \cdot i + [j + k]_d, \\ 0 & \text{otherwise} \end{cases}, \quad (7)$$

$$W_{d \cdot i + j, \ell}^k = \begin{cases} 1 & \text{if } \ell = d \cdot [i + k]_d + j, \\ 0 & \text{otherwise} \end{cases}. \quad (8)$$

Multiplication of two encrypted matrices $\text{ct}.A$ (ciphertext A) and $\text{ct}.B$ (ciphertext B) is performed as follows.

Step 1.1. Linear transformation U^σ on the input $\text{ct}.A$:

$$U^\sigma \cdot a = \sum_{-d < k < d} \left(u_k^\sigma \odot \rho(a; k) \right), \quad (9)$$

where $a = i^{-1}(A) \in \mathcal{R}^n$ is a vector representation of A .

This expression can be computed in the HE scheme as

$$\sum_{-d < k < d} \text{CMult}(\text{Rot}(\text{ct}.A; k); u_k^\sigma). \quad (10)$$

Step 1.2. Linear transformation U^τ on the input $\text{ct}.B$:

$$U^\tau \cdot a = \sum_{0 \leq k < d} \left(u_{d \cdot k}^\tau \odot \rho(b; d \cdot k) \right), \quad (11)$$

where $b = i^{-1}(B) \in \mathcal{R}^n$ and $u_{d \cdot k}^\tau$ is a diagonal vector of U^τ .

In HE, it is represented as follows:

$$\sum_{0 \leq k < d} \text{CMult}(\text{Rot}(\text{ct}.B; d \cdot k); u_{d \cdot k}^\tau). \quad (12)$$

Step 2. Homomorphic computation of operations $\sigma(A)$ and $\tau(B)$. For $1 \leq k < d$, the column shifting matrix V^k contains two non-zero diagonal vectors v_k and v_{k-d} :

$$v_k[\ell] = \begin{cases} 1 & \text{if } 0 \leq [\ell]_d < (d-k) \\ 0 & \text{otherwise} \end{cases}, \quad (13)$$

$$v_{k-d}[\ell] = \begin{cases} 1 & \text{if } (d-k) \leq [\ell]_d < d \\ 0 & \text{otherwise} \end{cases}. \quad (14)$$

Adding two ciphertexts $\text{CMult}(\text{Rot}(\text{ct}.A^{(0)}; k); v_k)$ and $\text{CMult}(\text{Rot}(\text{ct}.A^{(0)}; k-d); v_{k-d})$ we obtain $\text{ct}.A^{(k)}$ and $\text{ct}.B^k \leftarrow \text{Rot}(\text{ct}.B^{(0)}; d \cdot k)$. This step requires d additions, $2d$ constant multiplications, and $3d$ rotations.

Step 3. At this step, Hadamard multiplication is calculated for the $\text{ct}.A^{(k)}$ and $\text{ct}.B^{(k)}$ ciphertexts for $0 \leq k < d$, and we obtain the resulting ciphertext. This step requires d homomorphic additions and multiplications.

Although the method proposed by Jiang et al. [70] requires more depth compared to the method of Halevi and Shoup [69], the algorithm has the lowest time complexity and space complexity (number of ciphertexts) (see Table 2). These factors are crucial for efficient implementation of privacy-preserving neural networks, especially when processing large amounts of data. Therefore, in our work, we analyze the performance of the method of Jiang et al. [70].

Table 2. Main Properties of matrix multiplication algorithms.

Ref.	Number of Ciphertexts	Complexity	Required Depth	Library				OS		
				HElib	HEAAN	Microsoft SEAL	PALISADE	Windows	Linux	Mac OS
[69]	d	$O(d^2)$	1 Mult	•					•	
[79]	1	$O(d)$	1 Mult + 2 CMult		•					•
new	1	$O(d)$	1 Mult + 2 CMult			•	•	•	•	

5. Experimental Analysis

In this section all steps of HE matrix multiplication are analyzed: Encoding and encoding of an input matrix, encoding of precomputed auxiliary matrices, execution of matrix multiplication, decoding, and decoding of the result.

The creation of U^σ , U^τ , V^k , W^k , and the diagonal functions is not the subject of our experiments, since these operations are not related to HE and are performed identically in both libraries.

The implementation uses the CKKS scheme with Microsoft SEAL 3.5.6 and PALISADE v1.10.6 libraries on Ubuntu 20.04 and Windows 10 Home Edition. The hardware configuration consists of a CPU Intel Core i5-8250U 1.60 GHz, RAM DDR4 8 GB 1196 MHz and SSD 512 GB. The average time was measured by running the algorithms 1000 times on each platform.

We adopt the security settings specified in the HE standard for both libraries [63], see Table 3.

Table 3. Security parameters.

Parameter	Security Level	N	$\log Q$	ω
Value	128	8,192	220	3.2

The results are presented in two-dimensional graphs, with the abscissa indicating the size $d \in \{2, 3, \dots, 19\}$ of the square matrices and the ordinate indicating the time in seconds (sec.).

Each plot shows four curves illustrating the library-operating system combinations labeled SEAL-Linux, SEAL-Windows, PALISADE-Linux, and PALISADE-Windows, see Tables 4 and 5.

Table 4. Notations.

Notation	Meaning
SEAL-Linux	Implementation of the algorithm using the SEAL library compiled in OS Linux Ubuntu
SEAL-Windows	Implementation of the algorithm using the SEAL library compiled in OS Windows 10
PALISADE- Linux	Implementation of the algorithm using the PALISADE Library compiled in OS Linux Ubuntu
PALISADE-Windows	Implementation of the algorithm using the PALISADE Library compiled in OS Windows 10
T, s	Time to complete the operation/algorithm in seconds.
d	The matrix order
α	Degradation over PALISADE-Linux (times)

Table 5. Implementation characteristics with CKKS scheme.

Name	Library		OS	
	SEAL	PALISADE	Windows	Linux
SEAL-Linux	•			•
SEAL-Windows	•		•	
PALISADE- Linux		•		•
PALISADE-Windows		•	•	

The performance degradation is defined as the ratio of the execution time of all combinations to PALISADE-Linux.

5.1. Encoding Time

Figure 1 shows the encoding time of the U^σ matrix. It can be seen that the most efficient implementation is SEAL-Windows and the worst is SEAL-Linux (see Figure 1a). On average, it takes 0.94 times the time required by PALISADE-Linux, with the best result of 0.85 for the 10×10 matrix (Figure 1b).

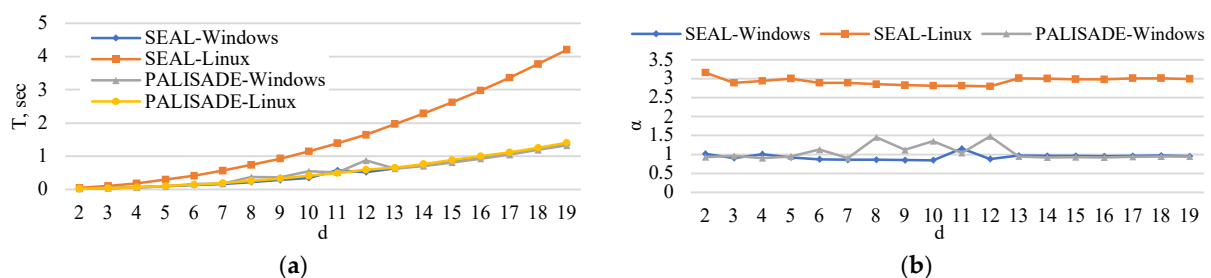


Figure 1. Encoding time of U^σ matrix: (a) encoding time; (b) degradation over PALISADE-Linux (times).

In the worst case, this implementation outperforms PALISADE-Linux by a factor of 1.15 for a matrix size of 11×11 .

The results of encoding the matrix U^τ are similar to those of the matrix U^σ (see Figure 2). The SEAL-Windows implementation takes on average 0.913 times compared to PALISADE-Linux. The best gain is 0.824 times for a 10×10 matrix and the worst case is 1.008 times with an 11×11 matrix.

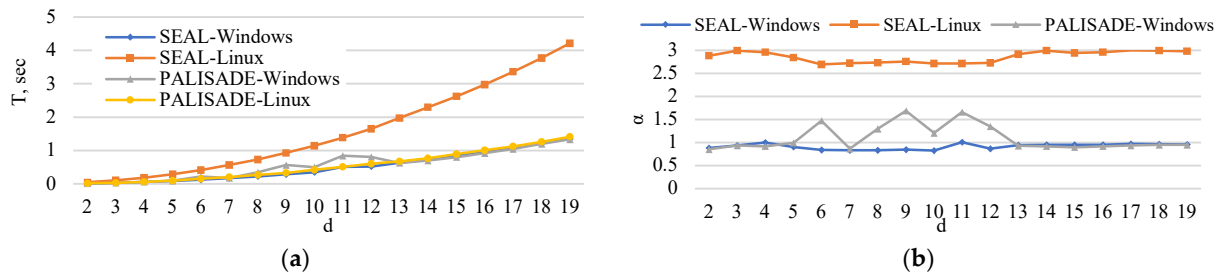


Figure 2. Encoding time of U^T matrix: (a) encoding time; (b) degradation over PALISADE-Linux (times).

The SEAL-Windows implementation also leads in encoding the V^k matrix, see Figure 3, requiring 0.92 times the time of PALISADE-Linux for the same operation. In this case, the best result is also observed for a matrix size of 10×10 with 0.82 times. In the worst case, the improvement is about 1.05 times for a 4×4 matrix.

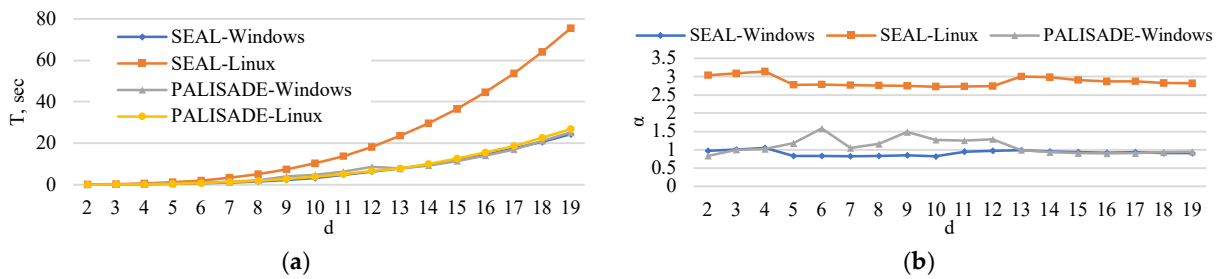


Figure 3. Encoding time of V^k matrix: (a) encoding time; (b) degradation over PALISADE-Linux (times).

As in the previous cases, encoding the W^k matrix with SEAL-Windows is more efficient (see Figure 4). It takes on average only 87% of the time that PALISADE-Linux needs for the same operation. In the best case, the time drops to 0.73 for a matrix size of 19×19 . The worst case occurs for an 11×11 matrix, where the implementation takes 1.09 times more time than PALISADE-Linux.

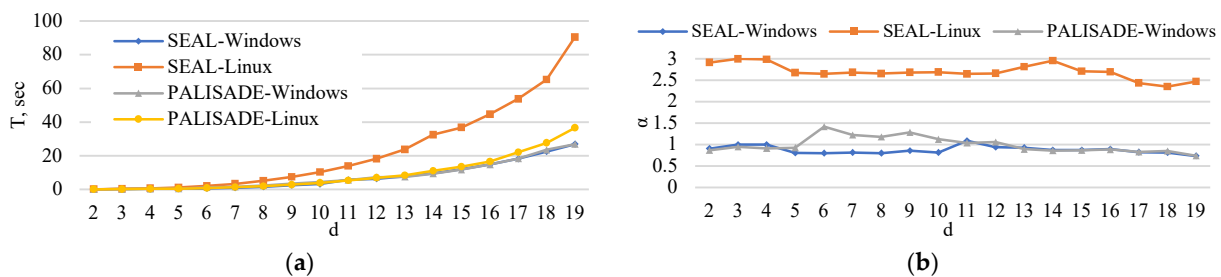


Figure 4. Encoding time of W^k matrix: (a) encoding time; (b) degradation over PALISADE-Linux (times).

Figure 5 shows the encoding time of the input matrix and confirms that SEAL-Windows provides the best implementation for encoding. On average, SEAL-Windows takes 0.835 of the time that PALISADE-Linux takes; the best reduction is only 63.1% of the time for an 18×18 matrix. The worst case occurs with a 4×4 matrix and an increase of 8% of the time. Furthermore, the PALISADE Windows implementation outperforms the PALISADE Linux implementation.

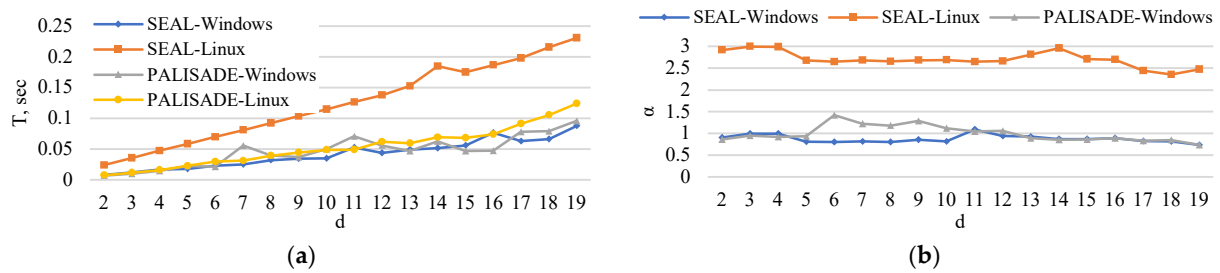


Figure 5. Encoding time of the input matrix: (a) encoding time; (b) degradation over PALISADE-Linux (times).

5.2. Encryption Time

For matrix encryption, SEAL-Linux is the most advantageous implementation, followed by PALISADE-Windows (see Figure 6). On average, SEAL-Linux requires only 93.4% of the time required by PALISADE-Linux. The best result is 0.58 times for the 17×17 matrix and the worst result is 1.13 times for the 14×14 matrix.

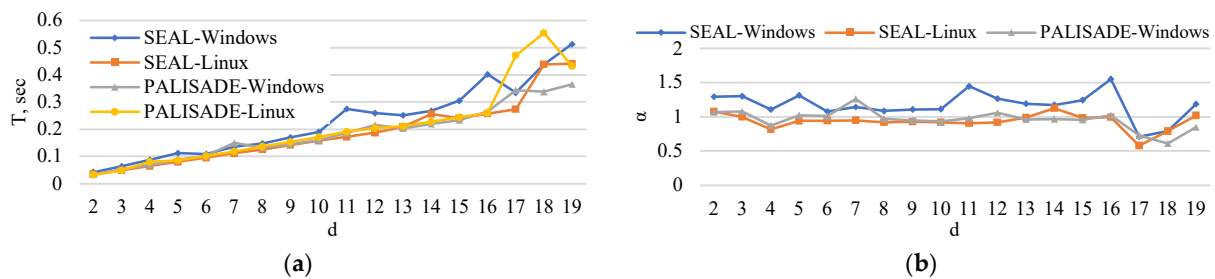


Figure 6. Encryption time of the input matrix: (a) encoding time; (b) degradation over PALISADE-Linux (times).

Figure 7 shows that the PALISADE-Linux implementation has the best performance in encrypting the input matrix. PALISADE-Windows is the second most efficient implementation with an average time gain of 20% for this operation. The minimum difference between PALISADE-Linux and SEAL-Windows is 1.002 times for the 4×4 matrix and the maximum difference is 16.37 times for the 17×17 matrix.

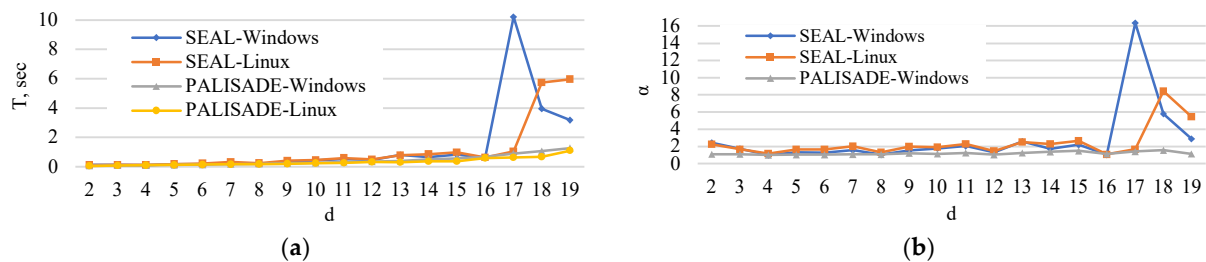


Figure 7. Encryption time of the input matrix into a single vector: (a) encoding time; (b) degradation over PALISADE-Linux (times).

5.3. Matrix Multiplication Time

The PALISADE Linux implementation is also the most efficient on average at performing matrix multiplication (see Figure 8). Other implementations take at least 1.3-times more to complete this operation. However, the PALISADE-Windows implementation is more advantageous for small matrices (up to 8×8), and its best result is observed for a matrix size of 5×5 . SEAL-Windows shows the worst performance for a 2×2 matrix, taking 6.5 times longer than the PALISADE-Linux implementation.

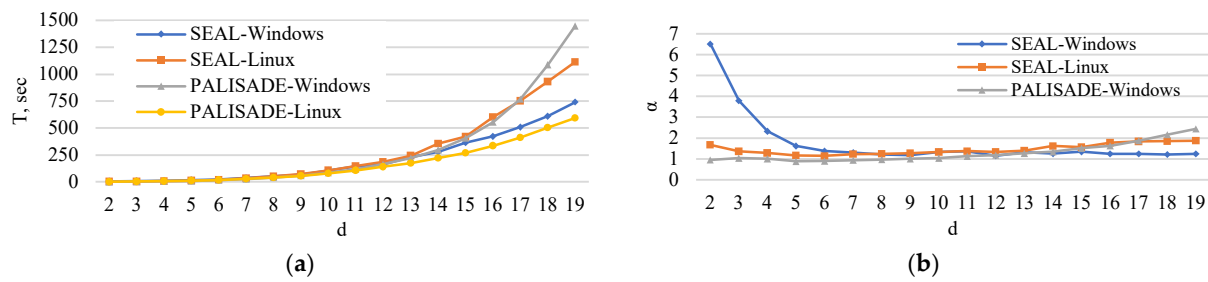


Figure 8. Matrix multiplication time: (a) encoding time; (b) degradation over PALISADE-Linux (times).

5.4. Decryption Time

Figure 9 shows the decryption time of the resulting matrix. The SEAL-Windows implementation is the most efficient with an average decryption time of only 0.3 times compared to PALISADE-Linux. The best performance is observed for SEAL-Windows with the 5×5 matrix, requiring 0.15 of the time spent by PALISADE-Linux. SEAL-Linux performs worst with the 16×16 matrix, taking 2.06 times as much time as the PALISADE-Linux implementation.

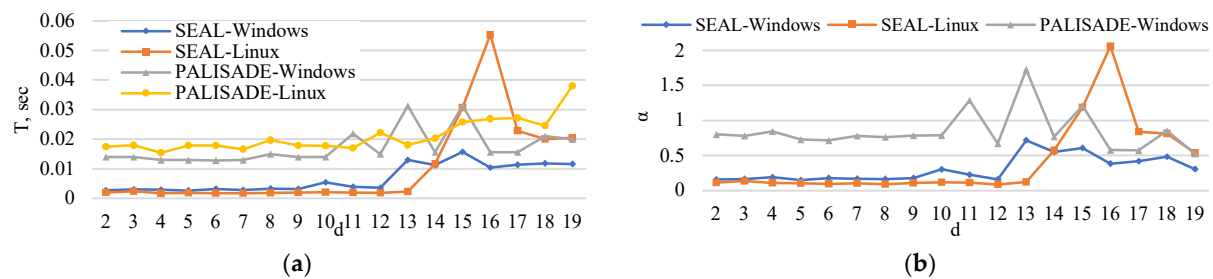


Figure 9. Decrypting time of the resulting matrix: (a) encoding time; (b) degradation over PALISADE-Linux (times).

5.5. Decoding Time

The PALISADE Linux implementation is the most efficient in decoding, see Figure 10. Other implementations take at least 69.6 times longer to complete this process. This result is observed for the implementation with a matrix size of 15×15 . The worst performance is provided by SEAL-Linux with a matrix size of 14×14 and takes more time than PALISADE-Linux.

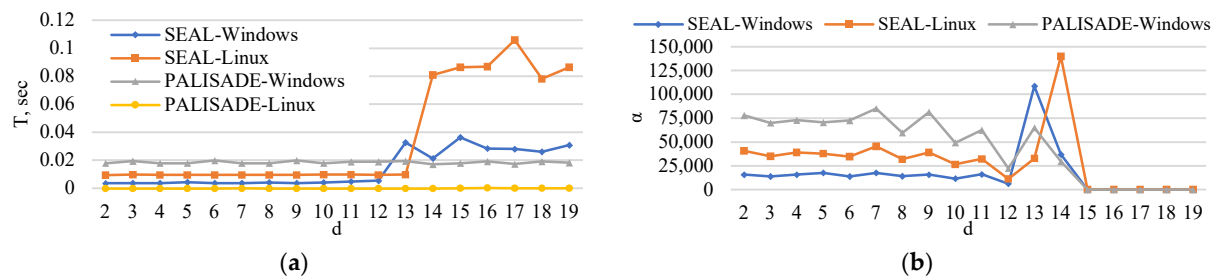


Figure 10. Decoding time of the resulting matrix: (a) encoding time; (b) degradation over PALISADE-Linux (times).

5.6. Execution Time

Figure 11 shows the execution time of the entire algorithm. All implementations require on average at least 1.28 times more time to execute the entire algorithm. In general, PALISADE-Linux outperforms PALISADE-Windows, SEAL-Linux, and SEAL-Windows by 1.28, 1.59, and 1.67 times, respectively, on average.

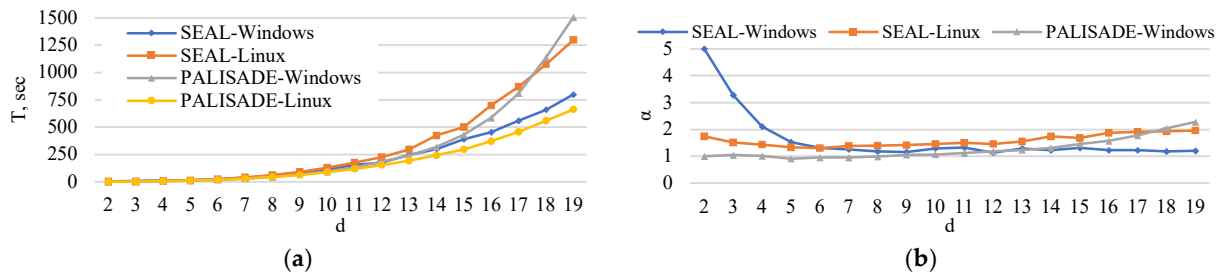


Figure 11. Execution time of the whole algorithm: (a) encoding time; (b) degradation over PALISADE-Linux (times).

However, for matrices from 5×5 to 8×8 , the PALISADE-Windows implementation is more efficient than PALISADE-Linux, taking 0.91 times on average. The SEAL-Windows implementation takes five times longer than the PALISADE-Linux implementation for the 2×2 matrix.

The difference in efficiency between the libraries and systems considered is that the PALISADE library is optimized for HE implementation under the Linux operating system, while Microsoft SEAL is optimized for HE implementation under the Windows operating system.

6. Extrapolation

To obtain benchmarks for approximate values outside the tested range of values, we further extrapolate the obtained curves by computing the approximate polynomials and reliability values. Deriving the approximate values outside the tested range allows us to evaluate the computational resources for implementing matrix multiplication in a cloud environment.

We estimate the efficiency of the HE matrix multiplication algorithm by inferring unknown values from trends in the known data.

The least squares method allows fine-tuning of the numerical parameters of a model function to fit a data set as well as possible.

The dataset of resulting measurements consists of $n = 18$ observations (d_i, T_i) , $i = \overline{1, n}$, where $d_i \in \{2, 3, \dots, 19\}$ is the order of a square matrix, and T_i defines the execution time of the entire algorithm. Estimates of the execution time outside the original observation can be found based on the relationship between the observed time and the order of the square matrix.

We use a polynomial extrapolation of the form:

$$f(d) = \sum_{i=0}^m a_i d^i \quad (15)$$

where m is the degree of a polynomial $f(d)$ and a_i is a coefficient of the polynomial $f(d)$.

The residual is a deviation measure used to evaluate the fit of a model to a data point. It is defined as the difference between the actual value of the dependent variable and the value predicted by the model:

$$r(d_i, T_i) = f(d_i) - T_i \quad (16)$$

The least-squares method finds the best parameter values by minimizing the sum S_{res} of squared residuals:

$$S_{res} = \sum_{i=1}^n r^2(d_i, T_i) \rightarrow \min \quad (17)$$

The coefficients a_i are calculated by solving the matrix equation $B \times A = C$ where $b_{i,j} = \sum_{k=1}^n d_k^{i+j-2}$, $c_i = \sum_{k=1}^n T_k d_k^{i-1}$, and $i, j \in \overline{1, m+1}$, see, Equation (15).

We use the coefficient of determination R^2 to estimate the polynomial extrapolation; it shows the degree of agreement of the mathematical model with the original data. This

value can be between 0 and 1. The closer the value is to 1, the more accurately the model describes the available data. The value of R^2 is calculated according to the following formula:

$$R^2 = 1 - \frac{S_{res}}{S_{tot}} \quad (18)$$

where $S_{tot} = \sum_{i=1}^n (T_i - \bar{T})^2$ and $\bar{T} = \frac{1}{n} \sum_{i=1}^n T_i$.

The value of R^2 for each implementation depends on the degree of polynomial extrapolation. Therefore, determining a degree of extrapolation with sufficient accuracy is essential.

Figure 12 shows the coefficient R^2 for the four implementations (library OS) and six polynomial extrapolations. It can be seen that R^2 is greater than 0.99 for all implementations only when the degree of polynomial extrapolation m is equal to or greater than 3.

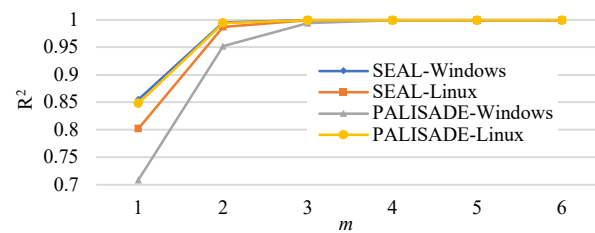


Figure 12. Coefficient R^2 for the four implementation (library-OS) and six polynomial extrapolations.

We compute a third-degree extrapolation polynomial for each implementation as they can provide approximations with sufficient accuracy ($R^2 \geq 0.99$ values):

- SEAL-Windows

$$f_{S-W}(d) = 0.14d^3 - 0.209d^2 + 2.315d + 2 \quad (19)$$

- SEAL-Linux:

$$f_{S-L}(d) = 0.403d^3 - 4.440d^2 + 23.516d - 26.728 \quad (20)$$

- PALISADE-Windows:

$$f_{P-W}(d) = 0.807d^3 - 14.28d^2 + 82.607d - 110.15 \quad (21)$$

- PALISADE-Linux:

$$f_{P-L}(d) = 0.147d^3 - 0.991d^2 + 7.588d - 9.826 \quad (22)$$

Asymptotically, the degradations of the other implementations compared to PALISADE-Linux (times) for $d \rightarrow \infty$ are given by:

$$\begin{aligned} \lim_{d \rightarrow \infty} \frac{f_{S-W}(d)}{f_{P-L}(d)} &= \frac{0.14}{0.147} \approx 0.952; \\ \lim_{d \rightarrow \infty} \frac{f_{S-L}(d)}{f_{P-L}(d)} &= \frac{0.403}{0.147} \approx 2.741; \\ \lim_{d \rightarrow \infty} \frac{f_{P-W}(d)}{f_{P-L}(d)} &= \frac{0.807}{0.147} \approx 5.490. \end{aligned}$$

After extrapolation analysis, PALISADE-Linux proves to be the best implementation for multiplying square matrices of order less than or equal to 104, since $\frac{f_{S-W}(104)}{f_{P-L}(104)} \approx 1.0003$. Similarly, SEAL-Windows would be a recommended option when matrices of order greater than or equal to 105 are used, since $\frac{f_{S-W}(105)}{f_{P-L}(105)} \approx 0.9999$.

7. Discussion

HE has the potential to be used in the development of privacy-friendly matrix operations that protect confidential data while enabling efficient processing and analysis. Matrix operations are fundamental to many machine-learning algorithms, including PPNN, and often require the processing of large amounts of data. HE can help protect these data while enabling efficient computation.

Using HE to create privacy-friendly matrix operations offers several benefits:

Improved privacy: HE provides a high level of data protection by encrypting the data before processing. This ensures that data remain secure throughout the computational process.

Efficient processing: HE allows computations to be performed on encrypted data without decryption, significantly reducing the computational cost of privacy-preserving matrix operations.

Flexibility: HE is a universal technique that can be used for various matrix operations, including those used in machine learning.

Compliance with regulations: HE can help organizations comply with regulations by providing a privacy-compliant solution for processing sensitive data.

Despite these benefits, there are also challenges associated with using HE to build matrix operations. These challenges include the high computational cost of homomorphic encryption, which can make it impractical for large-scale matrix operations, and the difficulty of implementing homomorphic encryption in existing matrix operation algorithms.

Overall, HE has significant potential for building privacy-friendly matrix operations, and as the technology advances, it is likely to become an increasingly important tool for protecting sensitive data while enabling efficient processing and analysis.

8. Conclusions

In this paper, we focus on the comparative analysis of two state-of-the-art HE matrix multiplication algorithms with the best time and space complexities for secure outsourcing. We analyze the Cheon–Kim–Kim–Song (CKKS) fixed-point homomorphic encryption scheme based on the Microsoft SEAL and PALISADE libraries on Windows and Linux. We show that the Windows operating system is preferred for the SEAL library and the Linux operating system is the best option for the PALISADE library.

PALISADE-Linux outperforms PALISADE-Windows, SEAL-Linux, and SEAL-Windows by an average of 1.28, 1.59, and 1.67 times, respectively, in most cases.

SEAL-Windows is more efficient at encrypting matrices and decrypting the resulting matrix. SEAL-Linux provides the best implementation for input matrix encryption. PALISADE-Linux shows the best performance in encrypting the input matrix into a single vector, in matrix multiplication, and in decrypting the resulting matrix.

To provide effective guidance in selecting a good implementation, we provide high-precision extrapolation formulas to asymptotically estimate the computation time of HE multiplication of larger matrices.

We found that polynomial extrapolation of at least degree $m = 3$ has a coefficient of determination $R^2 \geq 0.99$.

In future work, we will investigate the performance of homomorphic matrix multiplication under other factors such as CPU architecture, cache size, bus, memory parameters, libraries, and other scenarios. We will also develop an optimization of matrix multiplication for the SEAL and PALISADE libraries, considering their characteristics.

Author Contributions: Conceptualization, A.T., E.G. and M.B.; methodology, M.B.; software, E.G., G.V. and E.S.; validation, A.T., E.G. and E.S.; formal analysis, M.B.; investigation, E.G., T.E. and M.B.; resources, A.T. and L.A.G.; data curation, L.A.G. and T.E.; writing—original draft preparation, A.T., L.B.P.-G. and T.E.; writing—review and editing, A.T. and T.E.; visualization, G.V. and E.S.; supervision, A.T.; project administration, M.B. and A.A.; funding acquisition, A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Ministry of Education and Science of the Russian Federation (Project 075-15-2020-788).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

In this appendix, we briefly introduce the CKKS method. It is defined by the security parameters N , Q and ω . The parameters affect the security of the procedure, the types of plaintexts, the complexity, and thus the overall performance.

$N = 2^{\log N}$ refers to the dimension of the ring, i.e., a set closed by the operations of addition and multiplication. N is considered as the degree of the cyclotomic polynomial and the number of coefficients in the polynomials used as plaintext/ciphertext.

This method performs the encryption over $\mathbb{C}^{N/2}$. The batch encoding is represented as

$$\mathbb{C}^{\frac{N}{2}} \leftrightarrow \frac{Z_Q[X]}{X^N + 1},$$

where two elements are defined to be congruent if the difference between them is a multiple of $X^N + 1$.

Q defines the ciphertext modulus as a product of small coprime moduli q_i , where $q_i \equiv 1 \pmod{2N}$.

ω describes the variance used for the error polynomials.

The scaling of the plaintext is a parameter that affects the accuracy of the calculations in the CKKS method. First, the original values are scaled by the scaling factor and rounded to the nearest integer. Then, the generated integers are encoded using a polynomial with integer coefficients.

The main functions in a CKKS scheme are:

- **CKKS.Setup()**: Setting a ring of dimension N , a ciphertext modulus Q , a modulus p coprime to q , a key and error distribution χ and Ω over R , correspondingly.
- **SymEnc** (m_e, sk): $m_e \in R$ is an input plaintext and $sk = s \in R_{Qp}$ is a secret key. a and e are randomly picked from $U(R_{qp})$ and error Ω distributions, i.e., $a \leftarrow U(R_{Qp})$ and $e \leftarrow \Omega$. $b = -a \cdot s + e \in R_{Qp}$, where p is a word-sized prime number. It returns the ciphertext $ct = (c_0, c_1) = (b, a)$.
- **CKKS.KeyGen()**: Secret key $sk = s$, where s is drawn from key distribution χ , i.e., $s \leftarrow \chi$, and public key $pk = \text{SymEnc}(0, sk)$.
- **CKKS.Dec** (ct, sk): Converts ciphertext ct to plaintext. Given $ct = (c_0, c_1) \in R_{q_\ell}^2$ is a ciphertext at the ℓ -th level, the plaintext $c_0 + c_1 \cdot s \pmod{q_\ell}$ is returned.
- **Add** (ct_0, ct_1): Adds two ciphertexts ct_0 and ct_1 . The result is ciphertext $ct = ct_0 \oplus ct_1$.
- **Mult** (ct_0, ct_1): Multiplies two ciphertexts ct_0 and ct_1 . The result is ciphertext $ct = ct_0 \otimes ct_1$.
- **CMult** (ct, u): Multiplies ciphertext ct with some scalar u .
- **Rot** (ct, ℓ): Transforms an encryption ct of $m_e = (m_0, \dots, m_{n-1}) \in \mathcal{R}^n$ into an encryption of $\rho(m_e; \ell) := (m_\ell, \dots, m_{n-1}, m_0, \dots, m_{\ell-1})$.

References

1. Kamara, S.; Lauter, K. Cryptographic Cloud Storage. In Proceedings of the International Conference on Financial Cryptography and Data Security, Tenerife, Spain, 25–28 January 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 136–149.
2. Alabdulatif, A.; Kumarage, H.; Khalil, I.; Atiquzzaman, M.; Yi, X. Privacy-Preserving Cloud-Based Billing with Lightweight Homomorphic Encryption for Sensor-Enabled Smart Grid Infrastructure. *IET Wirel. Sens. Syst.* **2017**, *7*, 182–190. [\[CrossRef\]](#)
3. Borrego, C.; Amadeo, M.; Molinaro, A.; Jhaveri, R.H. Privacy-Preserving Forwarding Using Homomorphic Encryption for Information-Centric Wireless Ad Hoc Networks. *IEEE Commun. Lett.* **2019**, *23*, 1708–1711. [\[CrossRef\]](#)

4. Bouti, A.; Keller, J. Towards Practical Homomorphic Encryption in Cloud Computing. In Proceedings of the 2015 IEEE Fourth Symposium on Network Cloud Computing and Applications (NCCA), Munich, Germany, 11–12 June 2015; pp. 67–74.
5. Brakerski, Z. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In Proceedings of the Annual Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 868–886.
6. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory (TOCT)* **2014**, *6*, 1–36. [[CrossRef](#)]
7. dos Santos, L.C.; Bilar, G.R.; Pereira, F.D. Implementation of the Fully Homomorphic Encryption Scheme over Integers with Shorter Keys. In Proceedings of the 2015 7th IEEE International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 27–29 July 2015; pp. 1–5.
8. Chauhan, K.K.; Sanger, A.K.; Verma, A. Homomorphic Encryption for Data Security in Cloud Computing. In Proceedings of the 2015 IEEE International Conference on Information Technology (ICIT), Bhubaneswar, India, 21–23 December 2015; pp. 206–209.
9. Chen, J. Cloud Storage Third-Party Data Security Scheme Based on Fully Homomorphic Encryption. In Proceedings of the 2016 IEEE International Conference on Network and Information Systems for Computers (ICNISC), Wuhan, China, 15–17 April 2016; pp. 155–159.
10. Derfouf, M.; Eleuldi, M. Cloud Secured Protocol Based on Partial Homomorphic Encryptions. In Proceedings of the 2018 4th IEEE International Conference on Cloud Computing Technologies and Applications (Cloudtech), Brussels, Belgium, 26–28 November 2018; pp. 1–6.
11. El Makkaoui, K.; Ezzati, A.; Hssane, A.B. Challenges of Using Homomorphic Encryption to Secure Cloud Computing. In Proceedings of the 2015 IEEE International Conference on Cloud Technologies and Applications (CloudTech), Marrakech, Morocco, 2–4 June 2015; pp. 1–7.
12. El-Yahyaoui, A.; El Kettani, M.D.E.-C. A Verifiable Fully Homomorphic Encryption Scheme to Secure Big Data in Cloud Computing. In Proceedings of the 2017 IEEE International Conference on Wireless Networks and Mobile Communications (WINCOM), Rabat, Morocco, 1–4 November 2017; pp. 1–5.
13. Felipe, M.R.; Aung, K.M.M.; Ye, X.; Yonggang, W. Stealthycrm: A Secure Cloud Crm System Application That Supports Fully Homomorphic Database Encryption. In Proceedings of the 2015 IEEE International Conference on Cloud Computing Research and Innovation (ICCCRI), Singapore, 26–27 October 2015; pp. 97–105.
14. Kim, J.; Koo, D.; Kim, Y.; Yoon, H.; Shin, J.; Kim, S. Efficient Privacy-Preserving Matrix Factorization for Recommendation via Fully Homomorphic Encryption. *ACM Trans. Priv. Secur. (TOPS)* **2018**, *21*, 1–30. [[CrossRef](#)]
15. Peng, H.-T.; Hsu, W.W.; Ho, J.-M.; Yu, M.-R. Homomorphic Encryption Application on FinancialCloud Framework. In Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, Greece, 6–9 December 2016; pp. 1–5.
16. Hrestak, D.; Picsek, S. Homomorphic Encryption in the Cloud. In Proceedings of the 2014 37th IEEE International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 26–30 May 2014; pp. 1400–1404.
17. Jubrin, A.M.; Izegbu, I.; Adebayo, O.S. Fully Homomorphic Encryption: An Antidote to Cloud Data Security and Privacy Concerns. In Proceedings of the 2019 15th IEEE International Conference on Electronics, Computer and Computation (ICECCO), Abuja, Nigeria, 10–12 December 2019; pp. 1–6.
18. Kangavalli, R.; Vagdevi, S. A Mixed Homomorphic Encryption Scheme for Secure Data Storage in Cloud. In Proceedings of the 2015 IEEE International Advance Computing Conference (IACC), Bangalore, India, 12–13 June 2015; pp. 1062–1066.
19. Kavya, A.; Acharva, S. A Comparative Study on Homomorphic Encryption Schemes in Cloud Computing. In Proceedings of the 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 18–19 May 2018; pp. 112–116.
20. Ghanem, S.M.; Moursy, I.A. Secure Multiparty Computation via Homomorphic Encryption Library. In Proceedings of the 2019 Ninth IEEE International Conference on Intelligent Computing and Information Systems (ICICIS), Cairo, Egypt, 8–10 December 2019; pp. 227–232.
21. Kocabas, O.; Soyata, T. Utilizing Homomorphic Encryption to Implement Secure and Private Medical Cloud Computing. In Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing, New York, NY, USA, 27 June–2 July 2015; pp. 540–547.
22. Kocabas, O.; Soyata, T.; Couderc, J.-P.; Aktas, M.; Xia, J.; Huang, M. Assessment of Cloud-Based Health Monitoring Using Homomorphic Encryption. In Proceedings of the 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, USA, 6–9 October 2013; pp. 443–446.
23. Lupascu, C.; Togan, M.; Patriciu, V.-V. Acceleration Techniques for Fully-Homomorphic Encryption Schemes. In Proceedings of the 2019 22nd IEEE International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 28–30 May 2019; pp. 118–122.
24. Babenko, M.; Tchernykh, A.; Chervyakov, N.; Kuchukov, V.; Miranda-López, V.; Rivera-Rodriguez, R.; Du, Z.; Talbi, E.-G. Positional Characteristics for Efficient Number Comparison over the Homomorphic Encryption. *Program. Comput. Softw.* **2019**, *45*, 532–543. [[CrossRef](#)]
25. Marwan, M.; Kartit, A.; Ouahmane, H. Applying Homomorphic Encryption for Securing Cloud Database. In Proceedings of the 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt), Tangier, Morocco, 24–26 October 2016; pp. 658–664.

26. Murthy, S.; Kavitha, C.R. Preserving Data Privacy in Cloud Using Homomorphic Encryption. In Proceedings of the 2019 3rd IEEE International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 12–14 June 2019; pp. 1131–1135.
27. Hoffstein, J.; Pipher, J.; Silverman, J.H. NTRU: A Ring-Based Public Key Cryptosystem. In Proceedings of the International Algorithmic Number Theory Symposium, Portland, OR, USA, 21–25 June 1998; Springer: Berlin/Heidelberg, Germany, 1998; pp. 267–288.
28. Sun, X.; Zhang, P.; Sookhak, M.; Yu, J.; Xie, W. Utilizing Fully Homomorphic Encryption to Implement Secure Medical Computation in Smart Cities. *Pers. Ubiquitous Comput.* **2017**, *21*, 831–839. [\[CrossRef\]](#)
29. Tebaa, M.; El Hajji, S.; El Ghazi, A. Homomorphic Encryption Method Applied to Cloud Computing. In Proceedings of the 2012 IEEE National Days of Network Security and Systems, Marrakech, Morocco, 20–21 April 2012; pp. 86–89.
30. Van Dijk, M.; Gentry, C.; Halevi, S.; Vaikuntanathan, V. Fully Homomorphic Encryption over the Integers. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, France, 30 May–3 June 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 24–43.
31. Zhao, F.; Li, C.; Liu, C.F. A Cloud Computing Security Solution Based on Fully Homomorphic Encryption. In Proceedings of the 16th IEEE International Conference on Advanced Communication Technology, Pyeongchang, Republic of Korea, 16–19 February 2014; pp. 485–488.
32. Ni, Z.; Kundi, D.-e.-S.; O'Neill, M.; Liu, W. A High-Performance SIKE Hardware Accelerator. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2022**, *30*, 803–815. [\[CrossRef\]](#)
33. Cintas Canto, A.; Mozaffari Kermani, M.; Azarderakhsh, R. Reliable architectures for finite field multipliers using cyclic codes on FPGA utilized in classic and post-quantum cryptography. *IEEE Trans. Circuits Syst. I* **2023**, *1*, 157–161. [\[CrossRef\]](#)
34. Tian, J.; Wu, B.; Wang, Z. High-Speed FPGA Implementation of SIKE Based on an Ultra-Low-Latency Modular Multiplier. *IEEE Trans. Circuits Syst. I* **2021**, *68*, 3719–3731. [\[CrossRef\]](#)
35. Ogburn, M.; Turner, C.; Dahal, P. Homomorphic Encryption. *Procedia Comput. Sci.* **2013**, *20*, 502–509. [\[CrossRef\]](#)
36. Lu, W.; Sakuma, J. More Practical Privacy-Preserving Machine Learning as a Service via Efficient Secure Matrix Multiplication. In Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Toronto, ON, Canada, 19 October 2018; pp. 25–36.
37. Armknecht, F.; Boyd, C.; Carr, C.; Gjosteen, K.; Jäschke, A.; Reuter, C.A.; Strand, M. A Guide to Fully Homomorphic Encryption. Available online: <https://eprint.iacr.org/2015/1192> (accessed on 8 April 2023).
38. Kim, S.; Lee, K.; Cho, W.; Cheon, J.H.; Rutenbar, R.A. FPGA-Based Accelerators of Fully Pipelined Modular Multipliers for Homomorphic Encryption. In Proceedings of the 2019 IEEE International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 9–11 December 2019; pp. 1–8.
39. Kuang, L.; Yang, L.T.; Feng, J.; Dong, M. Secure Tensor Decomposition Using Fully Homomorphic Encryption Scheme. *IEEE Trans. Cloud Comput.* **2015**, *6*, 868–878. [\[CrossRef\]](#)
40. Lee, Y.; Lee, J.-W.; Kim, Y.-S.; No, J.-S. Near-Optimal Polynomial for Modulus Reduction Using L2-Norm for Approximate Homomorphic Encryption. *IEEE Access* **2020**, *8*, 144321–144330. [\[CrossRef\]](#)
41. Mert, A.C.; Öztürk, E.; Savaş, E. Design and Implementation of Encryption/Decryption Architectures for BFV Homomorphic Encryption Scheme. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *28*, 353–362. [\[CrossRef\]](#)
42. Al Badawi, A.; Chao, J.; Lin, J.; Fook Mun, C.; Jie Sim, J.; Meng Tan, B.H.; Nan, X.; Aung, K.M.M.; Ramaseshan Chandrasekhar, V. Towards the AlexNet Moment for Homomorphic Encryption: HCNN, TheFirst Homomorphic CNN on Encrypted Data with GPUs. *arXiv* **2018**, arXiv:1811. [\[CrossRef\]](#)
43. Cheon, J.H.; Kim, D.; Kim, Y.; Song, Y. Ensemble Method for Privacy-Preserving Logistic Regression Based on Homomorphic Encryption. *IEEE Access* **2018**, *6*, 46938–46948. [\[CrossRef\]](#)
44. Ciocan, A.; Costea, S.; Țăpuș, N. Implementation and Optimization of a Somewhat Homomorphic Encryption Scheme. In Proceedings of the 2015 14th IEEE RoEduNet International Conference-Networking in Education and Research (RoEduNet NER), Craiova, Romania, 24–26 September 2015; pp. 198–202.
45. Foster, M.J.; Lukowiak, M.; Radziszowski, S. Flexible HLS-Based Implementation of the Karatsuba Multiplier Targeting Homomorphic Encryption Schemes. In Proceedings of the 2019 MIXDES-26th IEEE International Conference “Mixed Design of Integrated Circuits and Systems”, Rzeszow, Poland, 27–29 June 2019; pp. 215–220.
46. Crainic, T.G.; Perboli, G.; Tadei, R. Recent advances in multi-dimensional packing problems. *New Technol. Trends Innov. Res.* **2012**, *1*, 91–110.
47. Cheon, J.H.; Kim, A.; Yhee, D. Multi-dimensional packing for heaan for approximate matrix arithmetics. Available online: <https://eprint.iacr.org/2018/1245> (accessed on 8 April 2023).
48. Microsoft SEAL 2022. Available online: <https://github.com/Microsoft/SEAL> (accessed on 8 April 2023).
49. Files Master-PALISADE/PALISADE Release GitLab. Available online: <https://gitlab.com/palisade/palisade-release/-/tree/master> (accessed on 8 April 2023).
50. Abadi, M.; Chu, A.; Goodfellow, I.; McMahan, H.B.; Mironov, I.; Talwar, K.; Zhang, L. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 308–318.
51. Li, L.; Fan, Y.; Tse, M.; Lin, K.Y. A review of applications in federated learning. *Comput. Ind. Eng.* **2020**, *149*, 106854. [\[CrossRef\]](#)

52. Du, W.; Atallah, M.J. Secure multi-party computation problems and their applications: A review and open problems. In Proceedings of the 2001 Workshop on New Security Paradigms, Cloudcroft, NM, USA, 10–13 September 2001; pp. 13–22.
53. Hirasawa, K.; Ohbayashi, M.; Koga, M.; Harada, M. Forward propagation universal learning network. In Proceedings of the IEEE International Conference on Neural Networks (ICNN'96), Washington, DC, USA, 3–6 June 1996; Volume 1, pp. 353–358.
54. Rumelhart, D.E.; Durbin, R.; Golden, R.; Chauvin, Y. Backpropagation: The basic theory. In *Backpropagation: Theory, Architectures and Applications*; Psychology Press: East Sussex, UK, 1995; pp. 1–34.
55. Bottou, L. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 421–436.
56. Albawi, S.; Mohammed, T.A.; Al-Zawi, S. Understanding of a convolutional neural network. In Proceedings of the 2017 IEEE International Conference on Engineering and Technology (ICET), Antalya, Turkey, 21–23 August 2017; pp. 1–6.
57. Rivest, R.L.; Adleman, L.; Dertouzos, M.L. On Data Banks and Privacy Homomorphisms. *Found. Secur. Comput.* **1978**, *4*, 169–180.
58. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic Encryption for Arithmetic of Approximate Numbers. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2017), Hong Kong, China, 3–7 December 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 409–437.
59. Gentry, C. A Fully Homomorphic Encryption Scheme; Stanford University. 2009. Available online: <https://crypto.stanford.edu/craig/craig-thesis.pdf> (accessed on 8 April 2023).
60. Homomorphic Encryption Standardization. Available online: <https://homomorphicencryption.org/> (accessed on 8 April 2023).
61. López-Alt, A.; Tromer, E.; Vaikuntanathan, V. On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 20–22 May 2012; pp. 1219–1234.
62. Fan, J.; Vercauteren, F. Somewhat Practical Fully Homomorphic Encryption. Available online: <https://eprint.iacr.org/2012/144> (accessed on 8 April 2023).
63. Gentry, C.; Sahai, A.; Waters, B. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In Proceedings of the Annual Cryptology Conference (CRYPTO 2013), Santa Barbara, CA, USA, 18–22 August 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 75–92.
64. Bos, J.W.; Lauter, K.; Loftus, J.; Naehrig, M. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In Proceedings of the IMA International Conference on Cryptography and Coding (IMACC 2013), Oxford, UK, 17–19 December 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 45–64.
65. Hariss, K.; Chamoun, M.; Samhat, A.E. Cloud Assisted Privacy Preserving Using Homomorphic Encryption. In Proceedings of the 2020 4th IEEE Cyber Security in Networking Conference (CSNet), Lausanne, Switzerland, 21–23 October 2020; pp. 1–8.
66. Kee, R.; Sie, J.; Wong, R.; Yap, C.N. Arithmetic Circuit Homomorphic Encryption and Multiprocessing Enhancements. In Proceedings of the 2019 IEEE International Conference on Cyber Security and Protection of Digital Services (Cyber Security), Oxford, UK, 3–4 June 2019; pp. 1–5.
67. Oppermann, A.; Grasso-Toro, F.; Yurchenko, A.; Seifert, J.-P. Secure Cloud Computing: Communication Protocol for Multithreaded Fully Homomorphic Encryption for Remote Data Processing. In Proceedings of the 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), Guangzhou, China, 12–15 December 2017; pp. 503–510.
68. Silva, E.A.; Correia, M. Leveraging an Homomorphic Encryption Library to Implement a Coordination Service. In Proceedings of the 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, 31 October–2 November 2016; pp. 39–42.
69. Halevi, S.; Shoup, V. Algorithms in Helib. In Proceedings of the Annual Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 554–571.
70. Jiang, X.; Kim, M.; Lauter, K.; Song, Y. Secure Outsourced Matrix Computation and Application to Neural Networks. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 1209–1222.
71. Kim, M.; Song, Y.; Wang, S.; Xia, Y.; Jiang, X. Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation. *JMIR Med. Inform.* **2018**, *6*, e8805. [\[CrossRef\]](#)
72. Pulido-Gaytan, B.; Tchernykh, A.; Cortés-Mendoza, J.M.; Babenko, M.; Radchenko, G.; Avetisyan, A.; Drozdov, A.Y. Privacy-Preserving Neural Networks with Homomorphic Encryption: Challenges and Opportunities. *Peer-to-Peer Netw. Appl.* **2021**, *14*, 1666–1691. [\[CrossRef\]](#)
73. Sun, X.; Zhang, P.; Liu, J.K.; Yu, J.; Xie, W. Private Machine Learning Classification Based on Fully Homomorphic Encryption. *IEEE Trans. Emerg. Top. Comput.* **2018**, *8*, 352–364. [\[CrossRef\]](#)
74. Yamada, Y.; Rohloff, K.; Oguchi, M. Homomorphic Encryption for Privacy-Preserving Genome Sequences Search. In Proceedings of the 2019 IEEE International Conference on Smart Computing (SMARTCOMP), Washington, DC, USA, 12–15 June 2019; pp. 7–12.
75. Ducas, L.; Micciancio, D. FHEW: Bootstrapping Homomorphic Encryption in Less than a Second. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, 26–30 April 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 617–640.
76. FFTW. Available online: <http://www.fftw.org/> (accessed on 10 December 2022).

77. TFHE. Available online: <https://tfhe.github.io/tfhe/> (accessed on 10 December 2022).
78. A GPU Implementation of Fully Homomorphic Encryption on Torus. Available online: <https://github.com/nucypher/nufhe> (accessed on 10 December 2022).
79. Lattigo: Lattice-Based Multiparty Homomorphic Encryption Library in Go 2022. Available online: <https://github.com/tuneinsight/lattigo> (accessed on 10 December 2022).
80. Crockett, E.; Peikert, C. $\Lambda\sigma$: Functional Lattice Cryptography. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 993–1005.
81. Dai, W.; Sunar, B. CuHE: A Homomorphic Encryption Accelerator Library. In Proceedings of the International Conference on Cryptography and Information Security in the Balkans, Koper, Slovenia, 3–4 September 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 169–186.
82. Doröz, Y.; Shahverdi, A.; Eisenbarth, T.; Sunar, B. Toward Practical Homomorphic Evaluation of Block Ciphers Using Prince. In Proceedings of the International Conference on Financial Cryptography and Data Security (FC 2014), Christ Church, Barbados, 3–7 March 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 208–220.
83. Concrete. Available online: <https://github.com/zama-ai/concrete-core> (accessed on 10 December 2022).
84. CuFHE. Available online: <https://github.com/vernamlab/cuFHE> (accessed on 10 December 2022).
85. Alves, P.; Aranha, D. Efficient GPGPU Implementation of the Leveled Fully Homomorphic Encryption Scheme YASHE. Available online: https://www.ic.unicamp.br/~ra085994/reports_and_papers/outros/drafts/efficient_gpgpu_implementation_of_yashe-draft.pdf (accessed on 8 April 2023).
86. Angelou, N. Node-Seal, A Homomorphic Encryption Library for TypeScript or JavaScript Using Microsoft SEAL. 2022. Available online: <https://github.com/s0l0ist/node-seal> (accessed on 8 April 2023).
87. Pyfhel. Available online: <https://github.com/ibarrond/Pyfhel> (accessed on 10 December 2022).
88. SEAL-Python. Available online: <https://github.com/Huelse/SEAL-Python> (accessed on 10 December 2022).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.