University of Ibadan
Journal of Science and Logics in ICT Research

A Review of Open-Source Fully Homomorphic Encryption Libraries: Zama.ai Concrete Compiler, Applications and Vulnerability

[1]Benedict D. A., [2,✉]Giwa T. A., [3]Usman O. L. and [4]Ezenduka C. F.

[1]Department of Electrical and Electronics Engineering, Federal University of Technology, Akure, Nigeria.
[2]Department of Computer Science, University of Abuja, Nigeria.
[3]Department of Computer Science, Tai Solarin University of Education, Ogun State, Nigeria,
[4]Department of Mechanical Engineering, Federal University of Technology Owerri, Nigeria.

Emails of authors: benedict.adeyi@gmail.com, tawakalitu.giwa@uniabuja.edu.ng *(corresponding)*, usmanol@tasued.edu.ng, chidieberef47@gmail.com

Abstract
Fully Homomorphic Encryption (FHE) is an advanced cryptographic technique that enables computational operations to be performed on encrypted data without the need for decryption. In other words, FHE allows operations to be conducted directly on ciphertexts, producing encrypted results that, when decrypted, correspond to the output of the operations performed on the plaintext data. This revolutionary capability ensures data privacy and security throughout the entire computation process, as the data remains encrypted at all times, even during computation. FHE schemes typically involve complex mathematical operations and algorithms, often based on lattice-based cryptography or other mathematical structures, to enable secure and efficient computation on encrypted data. Substantial progress has been achieved in the realm of FHE and its application since 2015, yielding enhanced efficacy, heightened security, and augmented feasibility. This review paper discusses and reviews diverse FHE schemes/libraries, and the extent of progress attained hitherto and how the possibilities of adoption of the scheme in industry is being propagated, using research questions as a guide, we endeavor to utilize searches across various academic databases and industry repositories for peer-reviewed papers, articles, and books. While some of the examined papers suggested new techniques to improve the security of transferred data, several of the publications provided novel schemes for FHE to maximize efficiency and minimize noise. Special emphasis is placed on the open-source tools and libraries implementing FHE scheme, notably Concrete (developed using TFHE Scheme), an innovation by Zama.ai, a preeminent research establishment specializing in FHE research and development. Since writing FHE programs can be difficult, Concrete, based on LLVM, makes this process easier for developers with the ability to compile Python functions (that may include NumPy) to their FHE equivalents, to operate on encrypted data. The applications of the library are examined, encompassing accomplishments, limitations, and vulnerabilities. Conclusively, prospective avenues for advancement are underscored, deliberated upon, and illuminated.

Keywords: Fully Homomorphic Encryption (FHE), Zama.ai Concrete Compiler, Machine Learning, Security, Open-source Library.

1.0 Introduction

According to Gouert *et al.* [1], Homomorphic Encryption (HE) represents a powerful encryption technique that permits arithmetic operations to be carried out on the encrypted data, resulting in ciphertexts that represent the desired outcomes. This capability allows for the secure encryption of data such as x and the delegation of computations, for instance $f(x)$, to a remote server while preserving the privacy of x. The outcome of these computations is received in an encrypted form, as depicted in the underlying process illustrated in Figure 1. Subsequently, the user possesses the ability to decrypt the result using their private key [2]. It is noteworthy that the RSA algorithm is recognized as the pioneer HE algorithm applied in the realm of cloud computing security, a development dating back to 1978, primarily due to its multiplicative property [3]. In a broader context, there are four distinct categories of HE, which are defined by the nature of homomorphic computations they support and the depth of these computations that can be performed on ciphertexts. These categories encompass Partial Homomorphic Encryption (PHE), Somewhat Homomorphic Encryption (SWHE), Leveled Homomorphic Encryption (LHE), and Fully Homomorphic Encryption (FHE) [4].

Benedict D. A., Giwa T. A., Usman O. L. and Ezenduka C. F. (2024). A Review of Open-Source Fully Homomorphic Encryption Libraries: Zama.ai Concrete Compiler, Applications and Vulnerability, *University of Ibadan Journal of Science and Logics in ICT Research (UIJSLICTR)*, Vol. 11 No. 2, pp. 24 – 35

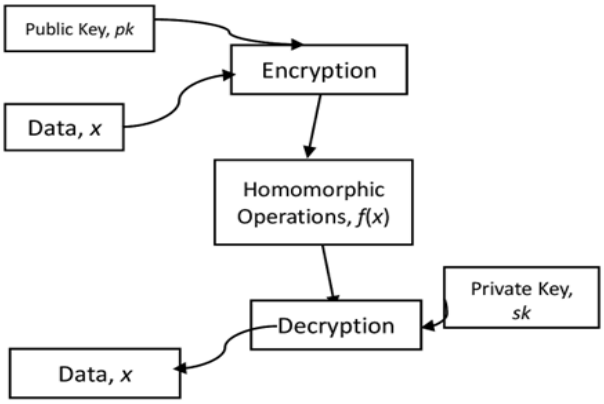

Figure 1. Typical flow of HE cycles.

The PHE scheme allows the user to carry out a single, restricted operation (either addition or multiplication) on encrypted data at a given moment, but it does not permit both operations simultaneously. On the other hand, SWHE schemes enable users to perform multiple operations on encrypted data; however, they have a constraint in terms of the number of addition and multiplication operations they can support [5, 6]. This limitation arises from the presence of noise in every ciphertext, and any homomorphic operation conducted on ciphertexts elevates the noise level in the resultant ciphertext. When the noise level surpasses a predefined threshold, the decrypted output is no longer accurate, and this imposes a restriction on the complexity of the polynomial that can be applied to ciphertexts [4, 7]. A Leveled Homomorphic Encryption (LHE) scheme can be described as a HE schemes that incorporates an additional parameter, denoted as *l*, which subsequently renders the scheme homomorphic for all binary arithmetic circuits of depth-*l*.

The Fully Homomorphic Encryption (FHE) scheme permits computations to be conducted on encrypted data while ensuring privacy preservation and information security [8, 9]. This scheme was initially conceptualized and put into practice by an IBM researcher, Craig Gentry [10, 11], based on ideal lattices, as detailed in his doctoral thesis published in 2009 at Stanford University. Nevertheless, subsequent studies revealed that Gentry's original FHE scheme was excessively intricate and challenging to implement [12]–[14] Consequently, there have been significant advancements in FHE schemes, leading to their categorization into three developmental phases: (i) FHE based on ideal lattices, (ii) FHE based on learning with error (LWE) and ring learning with error (RLWE) problems, and the Gentry, Sahai, and Waters (GSW) scheme. Despite these strides, further enhancements are required to render FHE schemes practical for real-world applications, particularly in the context of privacy-preserving machine learning (ML). Notably, recent developments have seen the integration of an open-source FHE scheme as a library, exemplified by Concrete developed by Zama.ai. Concrete serves as an open-source FHE compiler that streamlines the utilization of FHE. It possesses the capability to translate Python functions (including NumPy) into their FHE counterparts, enabling operations on encrypted data [15].

Prominent libraries in this domain include OpenFHE, HELib, NFLlib, TFHE-rs (a library for secure remote computing using FHE and trusted execution environments), and fast fully homomorphic encryption over Torus-TFHE, among others [16] – [18]. This paper conducts a critical evaluation of the progression of FHE libraries, emphasizing their advancements and vulnerabilities. Additionally, it delves into the diverse array of applications for FHE open-source libraries and demonstrates their potential impact across various application domains. Ultimately, this review underscores the importance of Zama Concrete libraries and compiler [15, 19] as indispensable tools for implementing and introducing FHE to developers.

The subsequent sections of this paper are structured as follows: In Section 2, we embark on a historical journey, tracking the evolution and advancements achieved thus far in the field of FHE library development. Section 3 provides an in-depth exploration of the diverse FHE open-source libraries offered by Zama.ai. Meanwhile, Section 3.1 and 3.2 shed light on the applications of open-source FHE libraries, along with an examination of their vulnerabilities. Section 4 is entirely devoted to the Zama.ai Concrete compiler, wherein we scrutinize its limitations by comparing its' performance against other known libraries, its' vulnerabilities, and present recommendations for future enhancements in Section 4.1 and 4.2 respectively. Finally, Section 5 serves as the conclusion, summarizing the key findings and insights gleaned from this study.

2.0 Historical and Progress Made So Far In FHE Development

The history of FHE is substantial, and work is still being done to improve efficiency, process, and practicability. Latency issues often take center stage when discussing FHE, though time and focused development will see continued improvement in the speed of programmable bootstrapping operations. Recent developments have focused on functionality and security, which helps to make FHE one of the fundamental methods of digital privacy protection [20]. Furthermore, introducing more efficient FHE libraries and frameworks has simplified the adoption and implementation of FHE in real-world scenarios.

Researchers have made significant strides in enhancing the efficiency and practicality of FHE schemes. The TFHE [21] was first presented as an improvement to the FHEW Scheme, but it quickly expanded in a more general direction. The scheme's security is built on a hard lattice issue known as Learning with Errors (LWE) and its derivatives, such as Ring LWE (RLWE). In reality, the vast majority of FHE methods in use today are LWE-based and employ noisy ciphertexts. The TFHE differs from the others in that it presents a unique bootstrapping method that is both quick and capable of evaluating a function while reducing noise.

The FHEW, introduced in 2014 [5], provided the ability to homomorphically compute basic bit operations while bootstrapping the outputs, lowering processing time from around 6 minutes per batch to approximately 0.69 seconds. FHEW prioritized bootstrapping, describing it as the main bottleneck in any practical implementation of FHE. Ducas and Micciancio [22] proved convincingly that macroscopic delays are not a necessary requirement of bootstrapped FHE computations and bootstrapping itself can be achieved at much higher speeds than previously thought possible. Therefore, using RLWE and adding the homomorphic NAND operation during bootstrapping helped to minimize latency and demonstrate the viability of FHE schemes [23].

Released in 2016, TFHE initially improved upon FHEW, adding more functionality and dramatically upgrading processing speed. Chillotti *et al.* [9, 21] improved latency to less than 0.1 seconds per gate based on bootstrapping operation. The scheme has since developed a programmable bootstrapping procedure into its process, speeding up FHE to make it practical for most use cases for web2 and web3 applications. Programmable bootstrapping enables the homomorphic evaluation of any function represented as a table lookup over a ciphertext with a controlled noise level. Only this bootstrapped mode is applicable for problems involving circuits of considerable depth and complexity. Deep neural networks and other machine learning techniques are prime use cases for libraries built on the TFHE scheme using programmable bootstrapping. The first implementation of a TFHE library was only for Boolean circuits, but today's state-of-the-art implementations, such as TFHE-rs [16], extend the original capabilities of TFHE to support programmable bootstrapping over integers.

3.0 Review of FHE Open-Source Libraries

This section reviews some recent open-source libraries and frameworks commonly used by FHE applications. Prominent among them include, but not limited to the following:

1. Microsoft SEAL Library: 2018-2022

Microsoft SEAL (Simple Encrypted Arithmetic Library) was created by researchers at Microsoft Research. It serves as a versatile tool for implementing and utilizing homomorphic encryption schemes [16]. Specifically, it supports two prominent schemes: the BFV (Brakerski-Vaikuntanathan) scheme and the CKKS (Cheon-Kim-Kim-Song) scheme [7]. BFV is a scheme that offers efficient operations on integers, while CKKS focuses on real and complex number computations. Microsoft SEAL aims to provide a user-friendly interface and optimized performance for these encryption schemes, making it accessible to researchers, developers, and practitioners. It emerged as a response to the growing need for practical and efficient homomorphic encryption tools.

Microsoft SEAL's development began around 2015 and was first released in December 2018. This library has continued with ongoing updates and improvements with the most stable version released in March 2022.

2. OpenFHE Library: 2022

OpenFHE is supported by NumFocus, an organization dedicated to supporting and promoting open-source scientific computing [17]. It involves contributions from multiple developers across the FHE community. This library is designed to be comprehensive, accommodating a wide range of FHE schemes, including BGV (Brakerski-Gentry-Vaikuntanathan), BFV, CKKS, TFHE (Fast Fully Homomorphic Encryption over the Torus), and FHEW (Fully Homomorphic Encryption over the Weil Descent) [16]. What sets OpenFHE apart is its multiparty support, allowing encrypted data to be shared and computed across multiple parties securely.

3. PALISADE Library: 2017-2022

PALISADE is a collaborative effort involving multiple organizations and developers in the field of cryptography, including academia, industry, and government contractors. The library was initially released in July 2017 and the stable version was released around May 2022. This library emphasizes support for multiple homomorphic encryption schemes, much like OpenFHE. PALISADE accommodates schemes such as BGV, BFV, CKKS, TFHE, and FHEW, and it also offers multiparty capabilities. PALISADE aims to provide a flexible and extensible platform for experimenting with and implementing various homomorphic encryption techniques and has since gained recognition as a significant open-source library for lattice-based cryptography and homomorphic encryption.

4. HELib Library: 2013-2021

HELib library was developed by researchers at IBM's Thomas J. Watson Research Center. Craig Gentry, Shai Halevi, and others [18] were involved in its creation. HELib's initial development took place around 2011-2012, marking it as one of the pioneering libraries in the field of homomorphic encryption using the CKKS and BGV schemes. The first released dated back to May 2013 while the stable version was released in October 2021. A noteworthy feature of HELib is its support for bootstrapping, a crucial technique for refreshing encrypted data to prevent decryption failures due to noise accumulation. The library was one of the pioneering tools in the field and has contributed significantly to the development and adoption of homomorphic encryption [24, 25]

5. HEAAN Library: 2016

Homomorphic Encryption for Arithmetic of Approximate Numbers (HEAAN) library is a specialized library that implements the CKKS homomorphic encryption scheme with a focus on native support for fixed-point approximate arithmetic [17]. This approach is particularly useful when precision needs to be balanced with computational efficiency, making it suitable for certain real-world applications [26].

6. Λ ○ λ Library: 2016

Λ ○ λ, pronounced as "L O L," is a Haskell library tailored for ring-based lattice cryptography. It supports Fully Homomorphic Encryption (FHE) and contributes to the landscape of FHE libraries available for researchers and developers working with Haskell [27].

7. NFLlib Library: 2016-2019

NFLlib library is an outcome of the European HEAT project, which involves collaboration among several European researchers and institutions [28]. It is dedicated to exploring high-performance homomorphic encryption using low-level processor primitives. This library is a testament to the ongoing efforts to improve the efficiency and practicality of homomorphic encryption techniques.

8. HEAT Library: 2015-2018

Homomorphic Encryption Acceleration Toolkit (HEAT) library is an initiative aimed at bridging FV-NFLib and HeLIB. The HEAT library serves as an API that bridges the FV-NFLib and HeLIB libraries. It focuses on enhancing the capabilities

of homomorphic encryption by providing a unified interface and potentially enabling interoperability between different FHE schemes [29].

9. HEAT Hardware Accelerator Library: 2022

The HEAT hardware accelerator is an extension of the HEAT project, the hardware acceleration component emerged later in the HEAT project's lifecycle, building upon the progress made with

FV-NFLib and HeLIB. This aspect of HEAT involves a hardware accelerator implementation for the FV-NFLlib. Hardware acceleration can significantly improve the computational efficiency of homomorphic encryption operations, making them more suitable for real-time and resource-constrained scenarios [30].

10. cuHE Library: 2016

The cuHE library project involves contributions from researchers and developers with expertise in cryptography and GPU programming [31]. The cuHE library explores the utilization of General-Purpose Graphics Processing Units (GPGPUs) to accelerate homomorphic encryption. GPGPUs are known for their parallel processing capabilities and leveraging them can lead to substantial speed improvements in FHE computations.

11. Lattigo Library: 2019-2022

Lattigo library was created by researchers and developers who sought to provide a Go-based library for lattice-based cryptography. Lattigo provides various tools and functionalities for researchers and developers interested in lattice-based cryptographic protocols, which have applications beyond just homomorphic encryption [32, 33].

12. Encrypted Vector Arithmetic (EVA):

An Encrypted Vector Arithmetic (EVA) is a compiler and optimizer designed for the CKKS scheme, with a primary target of Microsoft SEAL. This tool plays a crucial role in improving the performance and efficiency of homomorphic encryption operations, especially in the context of the CKKS scheme [30].

13. Fully Homomorphic Encryption over Torus (TFHE)

TFHE is an open-source library that provides implementations of FHE schemes optimized for speed [21]. It is designed to be efficient for large-scale homomorphic computations, making it suitable for practical applications. It stands as a robust framework designed to facilitate the implementation and utilization of fully homomorphic encryption (FHE) schemes. TFHE focuses on optimizing the speed of homomorphic computations, particularly for large-scale applications. It offers high throughput and low latency through various performance optimizations, including arithmetic operations on encrypted data and efficient bootstrapping procedures.

TFHE supports a wide range of homomorphic operations, allowing users to perform complex computations on encrypted data without decrypting it. Its security is based on lattice-based cryptography, providing strong guarantees against various cryptographic attacks, including quantum attacks.

TFHE is distributed as an open-source library, allowing for transparency, peer review, and community contributions. It emerged to address the increasing demand for practical and efficient homomorphic encryption tools, providing researchers, developers, and practitioners with a versatile platform for privacy-preserving computations.

Development of TFHE began around 2015 [21], and it has seen ongoing updates and improvements since then. The most current version of the library was released in February 2020, reflecting its commitment to continuous enhancement and refinement. TFHE's evolution underscores its significance as a fundamental tool in the field of homomorphic encryption, empowering users to leverage the benefits of privacy-preserving computations in various real-world applications.

Other notable libraries include: FHEW, TFHE-rs, FV-NFLlib, nuFHE, blyss, cuFHE, Cupcake, cuYASHE, FINAL, krypto libScarab, libshe, SparkFHE, Sunscreen, TenSEAL [22, 26, 34]. In summary, these libraries and tools collectively contribute to the advancement of homomorphic encryption and cryptography. They cater to a wide range of encryption schemes, optimizations, and applications, thereby fostering research.

3.1 Vulnerabilities in Fully Homomorphic Encryption

Despite the progress made, FHE schemes still have vulnerabilities. Side-channel attacks threaten FHE implementations' security, including timing and power analysis. Additionally, lattice-based FHE schemes are

susceptible to quantum attacks, highlighting the need for post-quantum secure FHE solutions. Ongoing research focuses on developing countermeasures to mitigate these vulnerabilities and strengthen the security of FHE.

3.2 Applications of Fully Homomorphic Encryption

The utilization of Fully Homomorphic Encryption (FHE) has significant promise across diverse application fields due to its capacity to facilitate computations while maintaining data in an encrypted state. These application areas include the following:

1. **Data Privacy and Security:** FHE offers the capability to conduct computations on confidential information while preserving the confidentiality of the original data. This becomes especially advantageous in situations where data needs to be sent for external processing, such as in cloud computing environments. Organizations can perform analyses on encrypted data without the necessity of decrypting it, thus safeguarding sensitive information throughout the process [2, 35, 36, 37].

2. **Healthcare:** Within the healthcare sector, FHE has the potential to facilitate the confidential and secure examination of patient data, serving purposes like medical research, diagnostics, and treatment strategy formulation. Hospitals and research institutes can engage in cooperative efforts involving encrypted patient records, thereby enabling the extraction of valuable insights while upholding the privacy of sensitive data [38].

3. **Finance:** Financial establishments have the opportunity to apply FHE for intricate financial computations conducted on encrypted data. This application facilitates secure evaluations of risk, fraud detection, and investment analysis, permitting collaborative data analysis among various financial entities while ensuring the confidentiality of sensitive customer data remains intact [26].

4. **Machine Learning and AI:** FHE can support the secure training and inference of models using encrypted data, thereby guaranteeing data privacy during the process of training machine learning models with sensitive datasets. This becomes particularly critical in scenarios where multiple organizations aim to cooperatively train models while abstaining from data sharing [39, 40].

5. **Internet of Things (IoT):** FHE can be implemented to ensure secure data processing within Internet of Things (IoT) applications. IoT devices can transmit encrypted data to centralized servers for analysis, and the outcomes can be returned in an encrypted format, thereby preserving the confidentiality of data generated by IoT devices [41].

6. **Supply Chain and Logistics:** Ensuring the security of data-sharing and analysis within the realm of supply chain management involves enabling collaboration among various stakeholders in a manner that employs encryption. This collaborative approach aids in enhancing logistical efficiency, monitoring shipments, and effectively handling inventory, all while safeguarding sensitive proprietary information [24].

7. **Genomics and Biotechnology:** FHE has the potential to facilitate secure cooperation and examination of genomic data in the fields of research and personalized medicine. Researchers and healthcare professionals can collaborate on encrypted genetic data to uncover valuable insights and advance the development of tailored medical interventions.

8. **Government and Defense:** Fully Homomorphic Encryption (FHE) holds significant importance in facilitating secure information exchange and data analysis within governmental and defense sectors. Classified data can undergo analysis while remaining in an encrypted state, thereby guaranteeing the preservation of the confidentiality of sensitive information.

9. **Academic and Scientific Research:** FHE finds applicability in collaborative research endeavors involving multiple institutions, wherein the analysis of sensitive data is required without the necessity of data sharing. This extends to various domains such as climate research, astronomy, and social sciences.

10. **Privacy-Preserving Analytics:** FHE schemes empowers organizations to conduct a diverse range of analytical operations on encrypted data while remaining compliant with

stringent data protection regulations like GDPR. These operations encompass the analysis of customer behavior, market research, and the extraction of data-driven business insights [2, 35].

4.0 ZAMA.AI Concrete Compiler: A TFHE Opensource Library

The Zama Concrete Library has emerged as a prominent tool for implementing Fully Homomorphic Encryption (FHE) based on the TFHE Scheme. Recently developed in March 2021 and built upon LLVM [15, 19], Zama Concrete introduces a high-level language designed for expressing FHE computations, along with the automatic generation of optimized FHE circuits [42]. This library incorporates several optimization strategies, including bootstrapping, packed ciphertexts, and Single Instruction Multiple Data (SIMD) instructions, to bolster the efficiency of FHE computations. It offers advanced capabilities such as automated noise management and reduced ciphertext sizes. SIMD instructions harness parallelism at the instruction level, further augmenting the performance of FHE computations. The Zama Concrete compiler significantly streamlines the development and deployment of FHE applications, effectively lowering the entry barrier for researchers and practitioners interested in working with FHE technology.

While Concrete provides a convenient means of loading the server library in Python, it is important to note that the Concrete library, as introduced by Zama [15], offers a variant of TFHE that supports floating-point plaintext encodings and bootstrapping capabilities, enabling the evaluation of univariate function

Nevertheless, it is essential to acknowledge certain constraints associated with its utility for general-purpose computation due to its reliance on a bootstrapping mechanism that necessitates the use of (imprecise) low-precision arithmetic. To illustrate this, CKKS bootstrapping allows for precision of up to 40 bits [25, 34], whereas Concrete is limited to a precision of less than 12 bits [23]. Furthermore, it is worth highlighting that the cumulative impact of rounding errors resulting from low-precision arithmetic can become substantial over time, particularly in deep applications, as evidenced by the reported loss of accuracy in deep neural networks in [22].

Architecture and Design Principles

The Zama.ai Concrete Compiler is built on robust architectural foundations, leveraging cutting-edge techniques to enable efficient FHE computations. At its core, the compiler employs a combination of mathematical optimizations and algorithmic innovations to minimize overhead while ensuring strong security guarantees. The architecture is modular and extensible, allowing for easy integration with various platforms and programming languages.

Supported Cryptographic Primitives and Operations

Zama.ai Concrete supports a wide range of cryptographic primitives and operations essential for homomorphic computations. Table 1 shows Concrete possible operations as at the latest version against other schemes [43], these include arithmetic operations (addition, multiplication), comparison operations (equality, inequality), and logical operations (AND, OR, NOT). Additionally, the library provides support for advanced functionalities such as bootstrapping, which enables the evaluation of arbitrary circuits over encrypted data.

Table 1 Operations of FHE Schemes.

Operation	BFV	BGV	CKKS	FHEW	TFHE	Concrete - TFHE
Add/Sub	Yes	Yes	Yes	No	No	Yes
Mult	Yes	Yes	Yes	No	No	Yes
Boolean Logic	No	Yes	No	Yes	Yes	Yes
Bootstrapping	No	No	No	Yes	Yes	Yes

Integration with Programming Languages and Platforms

Concrete offers seamless integration with popular programming languages such as C++, Python, and Java, facilitating the development of FHE-enabled applications across diverse domains. Moreover, the library is designed to be platform-agnostic, enabling deployment on a variety of computing environments ranging from cloud servers to edge devices. Table 2 shows comparison of various FHE libraries.

Table 2. Comparison of FHE Libraries.

Libraries/ Schemes	Microsoft SEAL	OpenFHE	Zama Concrete
BGV	No	Yes	No
BFV	Yes	Yes	No
CKKS	Yes	Yes	No
FHEW	No	Yes	No
TFHE	No	Yes	Yes

4.1 Performance Review of Concrete

Benchmarking Methodologies

To assess the performance of the Zama.ai Concrete Library, we employ benchmarking methodologies tailored to measure key metrics over a range of randomly generated data.

All Computation was executed on a PC with:

- 2.4 GHz Quad-core CPU;
- 16 GB RAM;
- 2 GB GPU
- Debian 12.4.

We conducted a comparative analysis of Zama.ai Concrete against other prominent FHE libraries, including SEAL, and OpenFHE. The comparison encompasses factors such as computational efficiency, memory overhead, and scalability, providing insights into the relative strengths and weaknesses of each library.

Execution Time:
The time taken for common operations (e.g., encryption, decryption, addition, and multiplication).

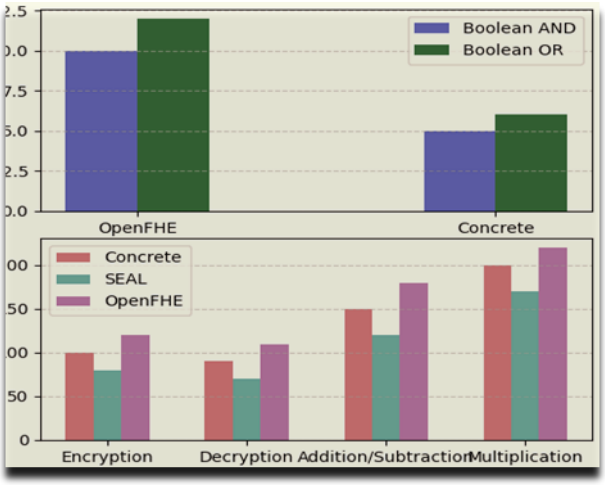

Figure 2: Average Execution Time for Computation Across FHE Libraries

Figure 2 performance assessment provides information about the effectiveness and scalability of Concrete Library, especially with regard to its capacity for homomorphic operations. The evaluation offers a thorough grasp of the library's capabilities and limits through a methodical study of performance indicators over a range of data sizes and compute complexity.

The computational cost of the library in comparison to other FHE Libraries, such the OpenFHE [17] is one important point that the performance assessment emphasizes. The analysis shows that homomorphic processes need a lot of computing power by nature, which makes processing times longer. But even with these difficulties, Concrete shows encouraging progress in reducing computational cost, especially when using better methods like as TFHE.

Also noted, the effect of noise accumulation during homomorphic processes, which can have a big influence on computation accuracy. The library tackles this problem by using a variety of noise reduction strategies, including sophisticated bootstrapping algorithms and error correction codes, improving the accuracy and dependability of FHE calculations.

But it's critical to recognize the shortcomings and weaknesses that the performance assessment brought to light. Security flaws like side-channel and timing attacks put FHE schemes at serious danger, thus it's important to keep working to create strong defenses and carry out in-depth analysis and security assessments.

4.2 Benefits, Shortcomings, and Vulnerabilities of Zama Concrete Library

Despite the progress made by Zama.ai's concrete compiler, several challenges and vulnerabilities remain. FHE schemes, including those incorporated by the compiler, suffer from high computational complexity, limiting their efficiency and scalability. Homomorphic operations require significant computational resources, resulting in longer processing times than traditional non-homomorphic computations such as advanced encryption standard (AES), and encryption algorithm recommended by the National Institute of Standard and Technology (NIST).

Although the concrete compiler leverages the improved TFHE scheme, which enhances the performance of FHE computations, there is still a need for ongoing research to develop more efficient algorithms and hardware architectures to reduce computational overhead. Noise accumulation during homomorphic operations remains a significant challenge, affecting the accuracy of computations.

Security vulnerabilities, such as side-channel and timing attacks, also pose risks to FHE schemes. Side-channel attacks exploit information leaked through physical characteristics of the system, such as power consumption or timing behavior. In contrast, timing attacks exploit variations in execution times to infer sensitive information. To ensure the security of FHE, ongoing efforts are required to develop robust countermeasures against these types of attacks and to conduct thorough security evaluations of these FHE schemes.

Zama's concrete compiler has achieved significant advancements in the practicality, application and efficiency of FHE schemes going as far as serving as a base for the development of the Concrete ML framework. It aims to simplify the use of FHE for data scientists to help them automatically turn machine learning models into their homomorphic equivalent [15]. Concrete ML was designed with ease-of-use in mind, so that data scientists can use it without knowledge of cryptography. Notably, the Concrete ML model classes are similar to those in scikit-learn and it is also possible to convert PyTorch models to FHE without being a cryptographer, this is a promising tool for developers and data analysts.

One of the notable achievements of Zama.ai's Concrete ML framework is its ability to efficiently perform the computation of complex machine-learning models on encrypted data. This capability has profound implications for privacy-preserving machine learning applications. For example, organizations can securely train machine learning models on sensitive data without exposing the raw data; this ensures the confidentiality and privacy of the data, making it suitable for scenarios such as collaborative machine learning or outsourced computation.

The library addresses the computational overhead associated with homomorphic operations, which has historically been a significant challenge in FHE. The library reduces the computational complexity by optimizing the translation of high-level computations to FHE circuits, resulting in faster and more practical FHE applications. This improvement is crucial for real-world deployment, as it enhances the performance and feasibility of using FHE in resource-constrained environments.

Furthermore, Zama.ai Concrete library mitigates the noise accumulation problem inherent in FHE schemes. Noise growth is a fundamental challenge in FHE that affects the accuracy of computations. By employing various noise reduction techniques, such as advanced bootstrapping methods and error correction codes, the compiler helps minimize noise accumulation's impact [24]. This advancement improves the reliability and precision of FHE computations, making them more suitable for sensitive tasks that require high accuracy.

4.3 Future Recommendations

To improve and tackle the highlighted challenges and vulnerabilities, the FHE scheme research community may consider exploring the following recommendations:

i. **Noise Reduction Techniques:** Additional research efforts can be directed towards the advancement of noise reduction techniques specifically tailored to the Concrete compiler and other tools utilizing FHE scheme. This endeavor aims to foster wider adoption and expansion in the application of homomorphic encryption. Such research avenues may encompass the exploration of enhanced bootstrapping techniques, the development of more efficient error correction codes, or the investigation of innovative low-noise FHE schemes.

ii. **Hardware Acceleration:** In order to mitigate the computational burden associated with FHE computations, the possibility of investigating specialized hardware architectures designed for optimized FHE operations merits consideration. Hardware acceleration has the potential to substantially augment the efficiency and effectiveness of FHE, rendering it a more viable solution for resource-limited contexts [30].

iii. **Algorithmic Improvements:** Ongoing research is imperative for the refinement of

algorithms used in FHE computations. This encompasses investigating approaches to streamline the intricacy of homomorphic operations, enhancing the packing and unpacking processes of ciphertexts, and optimizing the execution of Single Instruction, Multiple Data (SIMD) instructions.

iv. **Comprehensive Security Analysis:** Thorough security assessments are essential to detect and rectify potential weaknesses or vulnerabilities within the Concrete compiler and FHE schemes. This may entail evaluating their resilience against side-channel attacks, timing attacks, and other evolving threats, alongside the pursuit of robust countermeasures aimed at preserving the confidentiality and integrity of encrypted data.

v. **Standardization and Interoperability:** Emphasis should be placed on advancing the adoption of established standards and fostering interoperability among diverse FHE implementations, including Zama.ai Concrete compiler. Standardization serves as a catalyst for collaboration, promotes widespread acceptance, and elevates the compatibility of FHE solutions across a spectrum of platforms and applications.

Research efforts should center on the development of an efficient hardware architectures and finely-tuned algorithms aimed at diminishing the computational burden associated with FHE computations. Additionally, there should be ongoing endeavors to tackle potential vulnerabilities and bolster the security of FHE schemes, encompassing the implementation of resilient countermeasures against side-channel attacks and emerging threats.

5.0 Conclusion

In summary, this paper undertook a literature review focusing on Fully Homomorphic Encryption and its applications. The objective was to gain insights into Libraries implementing Fully Homomorphic Encryption Schemes and identify the most effective and up-to-date implementation approaches. Following the introduction of Fully Homomorphic Encryption Algorithms by Gentry [11], subsequent years saw various enhancements, including schemes based on Integer (DGHV), BGV, Multi-key, and GSW. The research questions formulated for this review encompassed the applications, advantages, disadvantages, and optimal implementation approaches of Fully Homomorphic Encryption Schemes. The methodology involved deriving keywords from the research questions to search for relevant peer-reviewed articles in academic directories. Which were then narrowed down through selection criteria. Each paper was categorized based on its contribution to the research questions and underwent analysis. Additionally, the selected papers were assessed for quality. Zama.ai Concrete library represents a significant milestone in advancing FHE, making secure and privacy-preserving computations more practical. The achievements of the library in enabling efficient evaluation of complex machine learning models, optimizing performance, and reducing noise growth are noteworthy. However, challenges related to computational complexity, noise accumulation, and security vulnerabilities still need to be addressed.

The Zama.ai Concrete library has significantly advanced the feasibility and effectiveness of Fully Homomorphic Encryption (FHE), expanding the horizons for secure and privacy-preserving computations. While substantial strides have already been made, there is room for continued advancement through persistent research and development. Implementing specific suggestions, such as noise reduction techniques, hardware acceleration, algorithmic enhancements, thorough security assessments, and standardization initiatives, has the potential to enhance the efficiency, security, and versatility of FHE. Ultimately, this could establish FHE as a foundational cornerstone for secure data processing in diverse domains.

References

[1] C. Gouert, D. Mouris, and N. Tsoutsos, "SoK: New Insights into Fully Homomorphic Encryption Libraries via Standardized Benchmarks," Proc. Priv. Enhancing Technol., vol. 2023, no. 3, pp. 154–172, 2023, doi: 10.56553/popets-2023-0075.

[2] O. L. Usman, R. C. Muniyandi, K. Omar, and M. Mohamad, "Privacy-Preserving Classification Method for Neural-Biomarkers using Homomorphic Residue Number System CNN: HoRNS-CNN," in 2022 International Conference on Business Analytics for Technology and Security, ICBATS 2022, IEEE, 2022. doi: 10.1109/ICBATS54253.2022.9759007.

[3] R. L. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” Commun. ACM, vol. 21, no. 2, pp. 120–126, 1978.
[4] K. Benzekki, A. El Fergougui, A. El, and B. El, “A Secure Cloud Computing Architecture Using Homomorphic Encryption,” vol. 7, no. 2, pp. 293–298, 2016.
[5] N. Vamshinath, K. R. Ramya, S. Krishna, P. Gopi Bhaskar, G. L. Mwaseba, and T.-H. Kim, “Homomorphic Encryption for Cluster in Cloud,” Int. J. Secur. Its Appl., vol. 9, no. 5, pp. 319–324, 2015, doi: 10.14257/ijsia.2015.9.5.31.
[6] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds,” in Advances in Cryptology – ASIACRYPT 2016. ASIACRYPT 2016. Lecture Notes in Computer Science, J. Cheon and T. Takagi, Eds., Berlin, Heidelberg: Springer, 2016, pp. 3–33. doi: 10.1007/978-3-662-53887-6_1.
[7] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic Encryption for Arithmetic of Approximate Numbers,” in Proceedings of the 2023 International Conference on Intelligent Systems for Communication, IoT and Security, ICISCoIS 2023, 2023, pp. 505–509. doi: 10.1109/ICISCoIS56541.2023.10100464.
[8] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, “Fast homomorphic evaluation of deep discretized neural networks,” in 2018 International Cryptology Conference, Springer, 2018, pp. 483–512. doi: 10.1007/978-3-319-96878-0_17.
[9] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2016, pp. 3–33. doi: 10.1007/978-3-662-53887-6_1.
[10] C. Gentry, “Computing Arbitrary Functions of Encrypted Data,” Commun. ACM, vol. 53, no. 3, pp. 97–105, 2010, doi: 10.1145/1666420.1666444.
[11] C. Gentry, “A Fully Homomorphic Encryption Scheme,” Standford University, 2009. [Online]. Available: http://cs.au.dk/~stm/local-cache/gentry-thesis.pdf
[12] C. Gentry and S. Halevi, “Implementing Gentry” s Fully-Homomorphic Encryption Scheme,” pp. 1–29, 2011.
[13] K. J. Muhammed, R. M. Isiaka, A. W. Asaju-Gbolagade, K. S. Adewole, and K. A. Gbolagade, “Improved Cloud-based N-Primes Model for Symmetric-based Fully Homomorphic Encryption using Residue Number System,” in Machine Learning and Data Mining for Emerging Trend in Cyber Dynamics, H. Chiroma, P. Abdulhamid, S M Fournier-Viger, and N. M. Garcia, Eds., Springer, Cham., 2021, pp. 197–216. doi: 10.1007/978-3-030-66288-2_8.
[14] L. O. Usman and K. A. Gbolagade, “A Review of Homomorphic Encryption Schemes for Cloud Computing Security: A Case for Residue Number System,” in 5th International Conference of U6 Initiative for Development, Malete: Kwara State University Malete, Nigeria, 2017, pp. 1–18.
[15] M. Arthur, C.-M. Benoit, F. Jordan, S. Andrei, B. Roman, and M. A. C. K. Luis, “What is Concrete ML.pdf,” 2022. https://docs.zama.ai/concrete-ml/v/0.5-1/
[16] L. Brenna, I. S. Singh, H. D. Johansen, and D. Johansen, “TFHE-rs: A library for safe and secure remote computing using fully homomorphic encryption and trusted execution environments,” Array, vol. 13, no. December 2021, pp. 1–8, 2022, doi: 10.1016/j.array.2021.100118.
[17] A. Al Badawi et al., “OpenFHE: Open-Source Fully Homomorphic Encryption Library,” in WAHC 2022 - Proceedings of the 10th Workshop on Encrypted Computing and Applied Homomorphic Cryptography, co-located with CCS 2022, 2022, pp. 53–63. doi: 10.1145/3560827.3563379.
[18] S. Halevi and V. Shoup, “Algorithms in HElib,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 8616 LNCS, no. PART 1, pp. 554–571, 2014, doi: 10.1007/978-3-662-44371-2_31.
[19] I. Chillotti, M. Joye, D. Ligier, J.-B. Orfila, and S. Tap, “CONCRETE: Concrete Operates oN Ciphertexts Rapidly by Extending TfhE,” in Proceedings of ACM Conference (Conference’17), Association for Computing Machinery, 2020, pp. 57–63. [Online]. Available: https://zama.ai
[20] P. Panzade and D. Takabi, “Towards Faster Functional Encryption for Privacy-preserving Machine Learning,” in Proceedings - 2021 3rd IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications, TPS-ISA 2021, 2021, pp. 21–30. doi: 10.1109/TPSISA52974.2021.00003.
[21] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “TFHE: Fast Fully Homomorphic Encryption Over the Torus,” J. Cryptol., vol. 33, no. 1, pp. 34–91, 2020, doi: 10.1007/s00145-019-09319-x.
[22] L. Ducas and D. Micciancio, “FHEW: Bootstrapping homomorphic encryption in less than a second,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9056, pp.

617–640, 2015, doi: 10.1007/978-3-662-46800-5_24.
[23] I. Chillotti, M. Joye, and P. Paillier, “Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2021, pp. 1–19. doi: 10.1007/978-3-030-78086-9_1.
[24] B. Li and D. Micciancio, “On the Security of Homomorphic Encryption on Approximate Numbers,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 12696 LNCS, pp. 648–677, 2021, doi: 10.1007/978-3-030-77870-5_23.
[25] H. V. L. Pereira, “Bootstrapping Fully Homomorphic Encryption over the Integers in Less than One Second,” in International Association for Cryptologic Research 2021, PKC 2021, LNCS 12710, International Association for Cryptologic Research, 2021, pp. 331–359. [Online]. Available: https://doi.org/10.1007/978-3-030-75245-3_13
[26] J. Zhang, X. Cheng, L. Yang, J. Hu, X. Liu, and K. Chen, “SoK: Fully Homomorphic Encryption Accelerators,” 2022, [Online]. Available: http://arxiv.org/abs/2212.01713
[27] E. Crockett and C. Peikert, “Λολ: Functional lattice cryptography,” in Proceedings of the ACM Conference on Computer and Communications Security, 2016, pp. 993–1005. doi: 10.1145/2976749.2978402.
[28] C. Aguilar-Melchor, J. Barrier, S. Guelton, A. Guinet, M. O. Killijian, and T. Lepoint, “NFLlib: NTT-based fast lattice library,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2016, pp. 341–356. doi: 10.1007/978-3-319-29485-8_20.
[29] D. .-E. F. Report, “HEAT: Homomorphic Encryption Applications and Technology,” 2018.
[30] S. Di Matteo, M. Lo Gerfo, and S. Saponara, “VLSI Design and FPGA Implementation of an NTT Hardware Accelerator for Homomorphic SEAL-Embedded Library,” IEEE Access, vol. 11, no. May, 2023, doi: 10.1109/ACCESS.2023.3295245.
[31] W. Dai and B. Sunar, “cuHE: A homomorphic encryption accelerator library,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2016, pp. 169–186. doi: 10.1007/978-3-319-29172-7_11.
[32] C. Mouchet, J. Bossuat, J. Troncoso-pastoriza, and J.-P. Hubaux, Lattigo : a Multiparty Homomorphic Encryption Library in Go, vol. 1, no. 1. Association for Computing Machinery, 2020. [Online]. Available: https://homomorphicencryption.org/wp-content/uploads/2020/12/wahc20_demo_christian.pdf
[33] J. A. G. Ed and G. Goos, PKC 2021 Lecture Notes in Computer Science. 2021.
[34] J. W. Lee, E. Lee, Y. Lee, Y. S. Kim, and J. S. No, “High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 12696 LNCS, pp. 618–647, 2021, doi: 10.1007/978-3-030-77870-5_22.
[35] O. L. Usman and R. C. Muniyandi, “CryptoDL : Predicting Dyslexia Biomarkers from Encrypted Neuroimaging Dataset Using Energy-E ffi cient Residue Number System and Deep Convolutional Neural Network,” Symmetry (Basel)., vol. 12, no. 5, pp. 1–24, 2020, doi: 10.3390/sym12050836.
[36] M. A. Usman, O. L. Usman, and R. C. Muniyandi, “Pixel-based Homomorphic Residue Number System Scheme for Privacy-Preserving Neuroimaging Datasets Encryption and Decryption,” TASUED J. Pure Appl. Sci., vol. 2, no. 1, pp. 1–9, 2023.
[37] A. Vizitiu, C. I. Niă, A. Puiu, C. Suciu, and L. M. Itu, “Applying Deep Neural Networks over Homomorphic Encrypted Medical Data,” Comput. Math. Methods Med., vol. 2020, 2020, doi: 10.1155/2020/3910250.
[38] J. Chao et al., “CaRENets: Compact and Resource-Efficient CNN for Homomorphic Inference on Encrypted Medical Images,” arXiv:1901.10074v1, pp. 1–11, 2019, [Online]. Available: http://arxiv.org/abs/1901.10074
[39] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, “Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy,” 33rd Int. Conf. Mach. Learn. ICML 2016, vol. 1, pp. 342–351, 2016.
[40] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, “Faster CryptoNets-Leveraging Sparsity for Real-World Encryption Inference,” arXiv:1811.09953v1, 2018.
[41] W. Ouyang, C. Ma, G. Zhang, and K. Diao, “Achieving Message-Encapsulated Leveled FHE for IoT Privacy Protection,” Mob. Inf. Syst. Journal, Hindawi, vol. 2020, pp. 1–10, 2020, doi: 10.1155/2020/8862920.
[42] Zama.ai, “Zama Concrete,” 2023. https://github.com/zama-ai/concrete
[43] L. Jiang and L. Ju “FHEBench: Benchmarking Fully Homomorphic Encryption Schemes” [Online]. Available: https://doi.org/10.48550/arXiv.2203.00728