Cross-Platform Benchmarking of the FHE Libraries: Novel Insights into SEAL and OpenFHE

Faneela[1], Jawad Ahmad[2], Baraq Ghaleb[1], Sana Ullah Jan[1], and William J. Buchanan[1]

[1] Blockpass ID Lab, Edinburgh Napier University, UK
[2] Cybersecurity Center, Prince Mohammad Bin Fahd University, Alkhobar, Saudi Arabia

Abstract. The rapid growth of cloud computing and data-driven applications has amplified privacy concerns, driven by the increasing demand to process sensitive data securely. Homomorphic encryption (HE) has become a vital solution for addressing these concerns by enabling computations on encrypted data without revealing its contents. This paper provides a comprehensive evaluation of two leading HE libraries, SEAL and OpenFHE, examining their performance, usability, and support for prominent HE schemes such as BGV and CKKS. Our analysis highlights computational efficiency, memory usage, and scalability across Linux and Windows platforms, emphasizing their applicability in real-world scenarios. Results reveal that Linux outperforms Windows in computation efficiency, with OpenFHE emerging as the optimal choice across diverse cryptographic settings. This paper provides valuable insights for researchers and practitioners to advance privacy-preserving applications using FHE.

Keywords: Homomorphic encryption, HE libraries, cross-platform, resource utilization, privacy-preserving

1 Introduction

Cloud computing has revolutionized modern computing by providing network access through cloud servers, which are remote storage systems that facilitate data processing and reduce the computational burden on local networks. While cloud servers offer significant advantages in terms of efficiency and scalability, their involvement also introduces critical security concerns, as sensitive data is entrusted to third-party entities. [1]. To address the security concerns associated with cloud computing, encryption techniques are employed to transform sensitive data into an unreadable form, ensuring its protection during transmission and storage. While essential for securing data during transmission and storage, traditional encryption techniques do not not fully address the privacy concerns in cloud computing as they typically require data to be decrypted

before any computation can occur, exposing sensitive information to potential vulnerabilities during processing. This limitation underscores the need for more advanced encryption methods, such as Homomorphic Encryption (HE), which enables computations to be performed directly on encrypted data without the need for decryption. HE resolves the privacy challenges by allowing data to remain confidential throughout its lifecycle, from storage to computation. Figure 1 illustrates the distinction between conventional cloud computing and HE-based computing, emphasizing the enhanced security and privacy protection provided by HE.
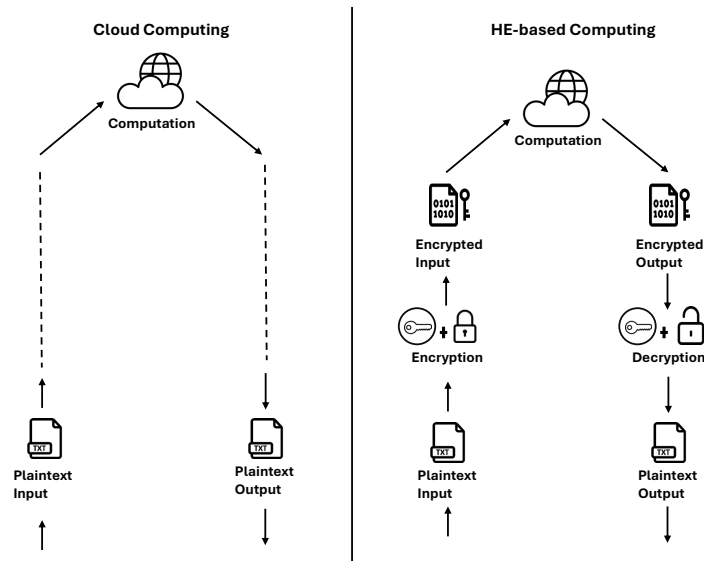

Fig. 1. Architectural Difference between Cloud and HE-based Computing

Homomorphic Encryption (HE) has made significant progress since its inception, evolving from a concept that allowed only simple arithmetic operations on encrypted data to a powerful tool for secure computations on large-scale data. Initially, HE was limited to performing either addition or multiplication on encrypted data, restricting its practical applicability. However, the field underwent a major breakthrough with Gentry's introduction of Fully Homomorphic Encryption (FHE) in 2009, which enabled both addition and multiplication operations on encrypted data simultaneously. This advancement opened the door to more complex and scalable computations, making HE suitable for a broader range of real-world applications [2]. In his research, he addressed the limitation of initial HE by introducing both types of homomorphic operations on encrypted data. Despite its theoretical promise, FHE faced challenges in practical implementa-

tion due to its high computational complexity. In response, several specialized libraries, such as Microsoft SEAL and OpenFHE, were developed, each offering efficient implementations of different FHE schemes, including the BGV and CKKS schemes. These libraries have made it possible to perform precise integer computations (BGV) or approximate floating-point operations (CKKS), further expanding the versatility and feasibility of HE in privacy-preserving computing. As the adoption of Homomorphic Encryption (HE) grows, it becomes essential to thoroughly analyze and evaluate the available HE libraries. Indeed, each library, such as Microsoft SEAL and OpenFHE, offers different trade-offs in terms of computational efficiency, ease of use, and support for various HE schemes. Therefore, a comprehensive evaluation is necessary to identify the strengths and limitations of these libraries in real-world scenarios. This evaluation will help researchers, developers, and organizations make informed decisions about which library best meets their specific needs, whether it be for secure data processing in cloud environments or for implementing scalable privacy-preserving systems. Furthermore, analyzing these libraries will provide valuable insights into their optimization and potential areas for improvement, ensuring that HE can be effectively deployed in practical, large-scale applications.

1. How do FHE libraries perform across different configuration settings?
2. How do varying HE parameters impact the performance of FHE libraries?
3. How does the choice of platform (Linux vs. Windows) influence the performance of FHE libraries?
4. How do different FHE schemes (e.g., BGV vs. CKKS) perform across various FHE libraries and OS platforms?

Hence, our study introduces a novel benchmark for evaluating the performance of Microsoft SEAL and OpenFHE, focusing on the BGV and CKKS schemes. We specifically examine two key FHE operations—homomorphic addition and multiplication. Additionally, we conduct a cross-platform analysis to assess the performance of these libraries in various environments.

2   Background

Practical implementations of Homomorphic Encryption (HE) have faced several challenges, including mathematical complexity and increased resource demands. To address these limitations, various encryption schemes have been proposed, such as RSA, ElGamal, and Paillier. In 1977, Rivest-Shamir-Adleman (RSA) proposed a practical method of HE using multiplication functions on encrypted messages, m_1, and m_2, such as:

$$\mathrm{Enc}(m_1) \cdot \mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 \cdot m_2) \tag{1}$$

Similarly, Paillier introduced an additive version of RSA's scheme where the operation is:

$$\mathrm{Enc}(m_1) + \mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 + m_2) \tag{2}$$

However, both RSA and Paillier schemes were limited to performing only one type of homomorphic operation—either addition or multiplication [3], [4]. This single-operation behavior of the aforementioned schemes is known as partial homomorphic encryption (PHE) and has limited their applications in practical scenarios [5]. To overcome the limitations of PHE, Craig Gentry introduced the first fully homomorphic encryption (FHE) scheme [6], which allowed both addition and multiplication operations on encrypted messages. For example, it enabled expressions like:

$$\text{Enc}(m_1), \text{Enc}(m_2) = \text{Enc}(m_1^2 + m_2 \cdot m_1) \tag{3}$$

Gentry's FHE scheme is based on lattice-based cryptography and bootstrapping procedures. Lattice-based cryptography ensures data privacy by using complex mathematical problems such as Learning with Errors (LWE) whereas Bootstrapping is a procedure that refreshes ciphertexts (encrypted messages) to minimize the noise growth during encryption.

Table 1. Focused Comparison of BGV and CKKS

Aspect	BGV	CKKS
Primary Use Case	Integer values	Floating-point values
Arithmetic Type	Modular arithmetics	Approximate arithmetics
Key Features	Batching technique	Scaling factor
Applications	Financial systems, cryptographic protocols.	Machine learning, data analytics.

2.1   FHE parameters

The adoption and effectiveness of Fully Homomorphic Encryption (FHE) in cryptographic applications depend significantly on the choice and optimization of several key parameters. These parameters directly influence the performance, security, and practicality of FHE schemes. Crucial parameters include the polymodulus degree, ciphertext modulus, plaintext modulus, scaling factor, and operation depth, all of which must be carefully chosen to balance security requirements and computational efficiency in FHE implementations [7], [8].

Polymodulus Degree. The polymodulus degree, denoted by n, determines the size of the polynomial ring used in the encryption process. It is critical in defining the complexity and security of the encryption scheme. The choice of n impacts both the computational overhead and the level of security against attacks. In ring theory, a polynomial ring R is constructed using polynomials $f(X)$, where the coefficients belong to certain rings, and is defined as:

$$R = \mathbb{Z}[X]/(X^n + 1) \tag{4}$$

The degree n determines the number of coefficients involved in the polynomial, thus influencing the size of the encryption parameters and the computational resources required for encryption and decryption.

Ciphertext Modulus. The ciphertext modulus q is a key parameter that influences the noise growth during homomorphic operations. Larger ciphertext moduli enable greater noise tolerance, allowing more operations to be performed on encrypted data before the ciphertext becomes corrupted.

Plaintext Modulus. The plaintext modulus t determines the size of the values that can be encrypted, directly impacting the range of data that can be securely handled by the encryption scheme.

Scaling Factor. The scaling factor Δ plays a vital role in controlling the noise growth during homomorphic operations, affecting the number of operations that can be performed on the ciphertext before it becomes too noisy for accurate decryption.

Operation Depth. The depth of operations L refers to the number of homomorphic operations that can be performed on the ciphertext before the noise grows to an irreparable level. This parameter is crucial in determining how many layers of computation are feasible within the encryption framework. These parameters—n, q, t, Δ, and L—must be carefully optimized to balance performance and security in FHE schemes.

2.2   FHE schemes: BGV and CKKS

Following Gentry's foundational approach, several Fully Homomorphic Encryption (FHE) schemes have been developed, with BGV and CKKS being two of the most widely used for practical applications [9]. The Brakerski-Gentry-Vaikuntanathan (BGV) scheme, introduced in 2012, is designed for efficient arithmetic operations, making it particularly suitable for exact computations, such as in cryptographic applications [10]. In 2017, the Cheon-Kim-Kim-Song (CKKS) scheme was introduced, which supports homomorphic computations on real numbers. CKKS employs approximation arithmetic, converting floating-point values into large integers to facilitate efficient homomorphic operations. [11], [12]. A key feature of CKKS is the scaling factor which ensures the precision of approximate calculations. This makes CKKS particularly well-suited for applications involving machine learning and other approximate computations, where exact precision is not always required. Table 1 highlights the key differences between the BGV and CKKS schemes.

2.3   FHE libraries: Microsoft SEAL and OpenFHE

To enable the practical deployment of Fully Homomorphic Encryption (FHE) schemes, several libraries have been introduced, with Microsoft SEAL and OpenFHE being among the most widely used. Microsoft SEAL is developed by Microsoft Research and is known for its user-friendly features and predefined FHE parameters, making it accessible to both researchers and practitioners [13].The library incorporates advanced techniques like memory pooling and multi-threading to

Algorithm 1: BGVadd(m, pk, sk, c1, c2)
Require: $m \in \mathbb{R}_q$, pk $\leftarrow$ (b, a), sk $\leftarrow$ s,
c1 $\leftarrow$ (c1$_0$, c1$_1$), c2 $\leftarrow$ (c2$_0$, c2$_1$).
considering
m: plaintext, pk: public key, sk: secret key,
c1, c2: ciphertexts
Ensure: c $\leftarrow$ (c$_0$, c$_1$), m $\in \mathbb{R}$
where
c: encrypted output, m: decrypted plaintext
Encryption:
1. u $\leftarrow$ Random ($\mathbb{R}_q$);
2. e $\leftarrow$ Noise ($\mathbb{R}_q$);
3. c$_0$ $\leftarrow$ b * u + m + e;
4. c$_1$ $\leftarrow$ a * u + e;
5. return c = (c$_0$, c$_1$);
Computation:
6. c$_0$ $\leftarrow$ (c1$_0$ + c2$_0$) mod q;
7. c$_1$ $\leftarrow$ (c1$_1$ + c2$_1$) mod q;
8. return c = (c$_0$, c$_1$);
Decryption:
9. m $\leftarrow$ (c$_0$ + c$_1$ * s) mod q;
10. return m;

Algorithm 2: CKKSadd(m, pk, sk, c1, c2, Δ)
Require: $m \in \mathbb{R}_q$, pk $\leftarrow$ (b, a), sk $\leftarrow$ s,
c1 $\leftarrow$ (c1$_0$, c1$_1$), c2 $\leftarrow$ (c2$_0$, c2$_1$), Δ.
considering
m: plaintext, pk: public key, sk: secret key,
c1, c2: ciphertexts
Ensure: c $\leftarrow$ (c$_0$, c$_1$), m $\in \mathbb{R}$
where
c: encrypted output, m: decrypted plaintext
Encryption:
1. u $\leftarrow$ Random ($\mathbb{R}_q$);
2. e $\leftarrow$ Noise ($\mathbb{R}_q$);
3. c$_0$ $\leftarrow$ b * u + Δ * m + e;
4. c$_1$ $\leftarrow$ a * u + e;
5. return c = (c$_0$, c$_1$);
Computation:
6. c$_0$ $\leftarrow$ (c1$_0$ + c2$_0$) mod q;
7. c$_1$ $\leftarrow$ (c1$_1$ + c2$_1$) mod q;
8. return c = (c$_0$, c$_1$);
Decryption:
9. m $\leftarrow$ ((c$_0$ + c$_1$ * s) / Δ) * mod q;
10. return m;

optimize resource utilization and performance. SEAL also supports cross-platform deployment, ensuring interoperability across various operating systems, including Windows, Linux, and macOS, which enhances its versatility in real-world applications. OpenFHE, on the other hand, is a research-driven library that also supports FHE implementations across multiple platforms. [14]. While both libraries differ in their design and architecture, they share a common goal: to facilitate the practical implementation of FHE schemes in real-world applications. Microsoft SEAL is well-suited for straightforward use cases, while OpenFHE provides more flexibility for cutting-edge research and complex application designs. This comparison provided valuable insights into the efficiency of these libraries under different conditions.

3   Related Work

Different studies have analyzed the performance of FHE libraries across different FHE schemes. The authors in [15] compared different FHE libraries such as HElib, SEAL, and FV-NFLlib and analyzed their performance with large plaintext moduli. Similarly, Jiang et. al evaluated these libraries under various FHE operations and provided a detailed analysis of their computational performance [16]. In another study, Gouert et al. examined the performance and usability of various FHE libraries, offering a comprehensive overview of their strengths and limitations [17].While these studies provide valuable insights into

the computational performance of FHE libraries, they do not address critical aspects such as memory usage and scalability across different platforms. These factors are essential for understanding the overall efficiency and adaptability of FHE libraries in diverse real-world environments. Recently, various research studies have also discussed the performance of specific FHE schemes to better understand FHE libraries. Chillotti et al. analysed the TFHE scheme to improve the performance of homomorphic operations [18]. The authors in [19] reviewed different FHE schemes such as BFV, CKKS, and TFHE schemes, and evaluated neural networks based on requirements. Ma et al. compared PyTFHE which is a Python wrapper of TFHE scheme, with other frameworks to highlight improvements in HE computations [20]. However, these studies did not explore the cross-platform performance of FHE schemes, especially in edge-cutting FHE libraries such as OpenFHE and SEAL. Several studies have also compared the performance of FHE libraries across various applications. Suzuki et al. analyzed the performance of SEAL and OpenFHE for convolutional neural networks, focusing on the trade-offs between processing latency and model accuracy [21]. Moreover, Zhai et al. presented GPU acceleration for encrypted computation using SEAL, focusing on hardware-specific optimizations [22]. While these studies provide valuable benchmarks for different FHE libraries, they fall short in addressing the cross-platform performance, memory usage, and operational depth of these libraries. The present study seeks to address these gaps by analyzing the SEAL and OpenFHE libraries across multiple platforms, focusing on both computation time and memory consumption. By providing new insights into these critical aspects, our work contributes to a deeper understanding of the performance of FHE libraries and promotes more efficient solutions for real-world FHE applications.

4   Proposed Methodology

This section outlines a benchmarking methodology designed to compare the performance of two FHE libraries: SEAL and OpenFHE. The study evaluates two key performance metrics—execution time and memory usage—across two operating systems, Windows and Linux. The analysis focuses on two widely adopted FHE schemes, BGV and CKKS, ensuring a comprehensive comparison. All experiments were conducted on a a laptop featuring an Intel Core i7-8550U processor, 16 GB of RAM, and a 512 GB SSD. To ensure uniformity and reliability of results, identical parameter settings were applied across all platforms. Performance evaluation was based on two fundamental FHE operations: addition and multiplication. The experiments explored various configurations parameters, including polymodulus degree, coefficient modulus, plaintext modulus, and scaling factor. The evaluation of the addition operations was conducted using a fixed set of operation depths (50, 100, 150, 200, 250, and 300) across polymodulus degrees of 13, 14, and 15. In contrast, the multiplication operations required dynamic parameter configurations due to their significantly higher computational complexity. As the operation depth increased, the polymodulus degree was

scaled accordingly to maintain computational accuracy and ensure correctness. Table 2 illustrates the correlation between operation depth and polymodulus degree, highlighting the adjustments necessary to accommodate deeper operations. Moreover, the coefficient modulus chain and scaling factor were also updated to accommodate the increasing computational complexity that resulted from the approximation arithmetic technique inherent in the CKKS scheme.

Table 2. Correlation between operation depth and polymodulus degree.

Operation Depth (L)	Polymodulus Degree (q)			
	SEAL		OpenFHE	
	BGV	CKKS	BGV	CKKS
2	12	13	14	14
3	13	13	14	14
4	13	14	14	14
5	13	14	14	15
6	13	14	14	15
7	14	14	14	15
8	14	14	14	15
9	14	15	14	15
10	14	15	14	15
11	14	15	14	16
12	14	15	14	16
13	15	15	15	16
14	15	15	15	16
15	15	15	15	16
16	15	15	15	16
17	15	15	15	16
18	15	15	15	16
19	15	15	15	16
20	15	15	15	16

5   Results and Discussion

This section presents a detailed benchmarking analysis of the SEAL and OpenFHE libraries, focusing on their performance under the BGV and CKKS schemes. The discussion is structured into two subsections, each dedicated to evaluating the performance of core homomorphic operations: FHE addition and FHE multiplication.

5.1   FHE-Addition Performance

This section presents a detailed benchmarking analysis of the SEAL and OpenFHE libraries, focusing on their performance under the BGV and CKKS schemes. The discussion is structured into two subsections, each dedicated to evaluating the performance of core homomorphic operations: FHE addition and FHE multiplication, outlined in Figures 2 to 5.
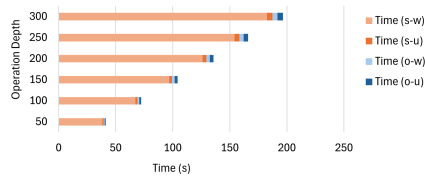

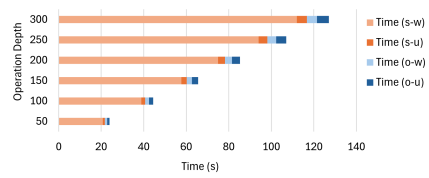**Fig. 2.** Execution Time for FHE-addition in BGV.
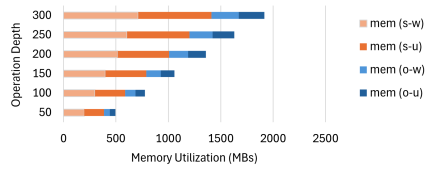

Fig. 3. Execution Time for FHE-addition in CKKS.

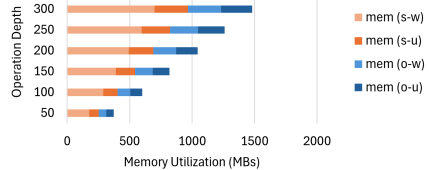**Fig. 4.** Memory Utilization for FHE-addition in BGV.

Fig. 5. Memory Utilization for FHE-addition in CKKS.

Figure 2 shows the execution time of FHE addition under the BGV scheme as a function of operation depth. Execution time increased gradually with operation depth for both libraries. SEAL initially performed well but exhibited a sharp rise in execution time on Windows as the operation depth grew. At a depth of 300, SEAL's execution time on Windows reached 182 seconds, highlighting significant scalability challenges. In contrast, OpenFHE demonstrated consistently faster execution times on both Windows and Linux, with Linux outperforming Windows in all cases, further emphasizing OpenFHE's optimized performance. Similarly, the execution times in CKKS were consistently higher than in BGV due to the further computational complexity introduced by the approximation technique. However, OpenFHE on Linux still has the additional benefit of completing 300 FHE additions in almost 4 seconds, compared to SEAL's 7 seconds (approx.) on the same platform. Figure 3 presents the execution time of the FHE-addition operation in CKKS across all configurations.

Moreover, 4 outlines a similar pattern for memory usage. In the BGV, SEAL performed slightly better on Linux compared to Windows. For instance, the

memory usage at depth 300 was 690 MB on Linux and 715 MB on Windows. In contrast, OpenFHE demonstrated considerably better memory usage across all configurations with 240 MB utilized for the same operation depth. Similarly, the CKKS consumed more memory than BGV for both libraries across all platforms, as shown in 5. SEAL demanded the highest memory on Windows, whereas OpenFHE remained memory efficient even at larger operation depths. This highlights its potential to effectively manage resource demands for CKKS.

5.2   FHE-Multiplication Performance

The FHE-multiplication analysis offers a comprehensive overview of the computational efficiency and memory usage of the aforementioned FHE libraries shown in Figures 6 to 9. The execution time for multiplication operations in BGV has risen continuously with operation depth across both libraries. The details are outlined in Figure 6. Overall, SEAL took significantly longer execution times on Windows. For instance, at operation depth 20, SEAL took more than 530 seconds to execute on Windows, which indicates the future challenges for scalability. Meanwhile, OpenFHE outperformed across each configuration, executing 20 multiplications in approximately 7 seconds on Linux, which significantly overtook SEAL on both platforms. Similarly, the execution time for CKKS was generally higher than in BG as described in FHE-addition. The details are shown in Figure 7. SEAL on Windows took about 300 seconds for depth 20, whereas OpenFHE on Linux exhibited the same operations in 12 seconds, which shows a clear performance efficiency.
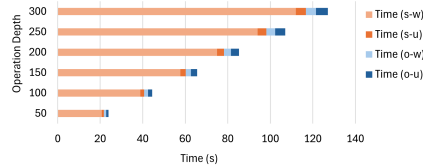

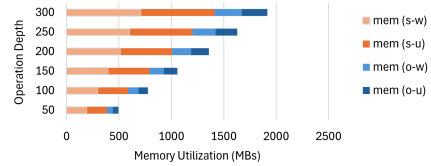**Fig. 6.** Execution Time for FHE-multiplication in BGV.

Fig. 7. Execution Time for FHE-multiplication in CKKS.

Moreover, the memory usage in BGV and CKKS is highlighted in Figures 7 and 8, respectively. It is observed that the memory demand for SEAL in BGV increased dramatically, such as for 20 FHE multiplications, which required 850 MB on Windows. However, while Linux showed slightly better performance, OpenFHE was found to be more memory efficient, requiring less than half of SEAL. For example, at depth 20, OpenFHE showcases its optimized resource management by consuming 400MB on the Linux platform, as shown in Figure 8. Similarly, for the memory usage in CKKS, SEAL exceeded 1200 MB at depth 20 on Windows, making it the most resource-intensive configuration, as illustrated

in Figure 9. Whereas, OpenFHE has again proven its efficiency by requiring under 800 MB on Linux for the same operation depth.
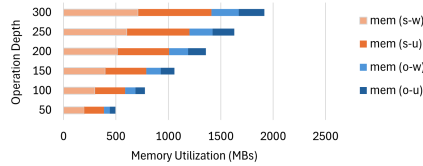

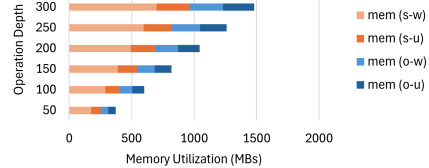**Fig. 8.** Memory Utilization for FHE-multiplication in BGV.

Fig. 9. Memory Utilization for FHE-multiplication in CKKS.

Furthermore, a significant increase in execution time and memory usage was observed with each transition to a higher polymodulus degree (n) to manage higher depths of multiplication operations. For each transition, SEAL showed more resource growth while OpenFHE demonstrated fewer fluctuations. For instance, in BGV, at operation depth 9, for transitioning from polymodulus degree 2^{13} to 2^{14}, SEAL on Windows consumed $5x$ more execution time than the regular pattern. Whereas, at depth 13, transitioning from polymodulus degree 2^{14} to 2^{15} in OpenFHE resulted in only a $2x$ increase on Linux, which confirms its improved scalability.

5.3   Comparative Insights

Overall, OpenFHE continuously outperformed SEAL in terms of execution time and memory usage. This outperforming behavior of OpenFHE became more evident at higher operation depths, particularly in CKKS to handle the scaling factors in managing computational overhead. However, the higher computational demand of SEAL may limit its scalability in advanced applications. The performance of SEAL was also greatly affected by the platform choice such as Linux consistently performed better than Windows. Hence, OpenFHE is an optimal choice for improved resource utilization which makes it suitable for cryptographic applications.

6   Conclusion

Our study benchmarks two widely used FHE libraries, SEAL and OpenFHE under BGV and CKKS schemes. This benchmarking provides new insights into computation overhead and the scalability of these libraries in a cross-platform approach. Our findings reveal that OpenFHE consistently outperformed SEAL in all configurations, where Linux appeared to be a more efficient platform for each operation setting. In the future, this study can be extended to include more libraries using different FHE schemes that would help researchers gain

comprehensive insights into FHE advancements. Moreover, different hardware settings, such as GPUs or specialized accelerators, can be involved to highlight the performance of homomorphic operations. Furthermore, the evaluation of multiple FHE schemes in neural networks using different libraries would offer new research directions in implementing privacy-preserving applications.

References

1. K. Munjal and R. Bhatia, "A systematic review of homomorphic encryption and its contributions in healthcare industry," *Complex & Intelligent Systems*, vol. 9, no. 4, pp. 3759–3786, 2023.
2. C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.
3. S. J. Mohammed and D. B. Taha, "Performance evaluation of rsa, elgamal, and paillier partial homomorphic encryption algorithms," in *2022 International Conference on Computer Science and Software Engineering (CSASE)*. IEEE, 2022, pp. 89–94.
4. P. V. Parmar, S. B. Padhar, S. N. Patel, N. I. Bhatt, and R. H. Jhaveri, "Survey of various homomorphic encryption algorithms and schemes," *International Journal of Computer Applications*, vol. 91, no. 8, 2014.
5. S. S. Reddy, S. Sinha, and W. Zhang, "Design and analysis of rsa and paillier homomorphic cryptosystems using pso-based evolutionary computation," *IEEE Transactions on Computers*, vol. 72, no. 7, pp. 1886–1900, 2023.
6. M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*. Springer, 2010, pp. 24–43.
7. J.-P. Bossuat, R. Cammarota, I. Chillotti, B. R. Curtis, W. Dai, H. Gong, E. Hales, D. Kim, B. Kumara, C. Lee *et al.*, "Security guidelines for implementing homomorphic encryption," *Cryptology ePrint Archive*, 2024.
8. M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter *et al.*, "Homomorphic encryption standard," *Protecting privacy through homomorphic encryption*, pp. 31–62, 2021.
9. A. Kim, M. Deryabin, J. Eom, R. Choi, Y. Lee, W. Ghang, and D. Yoo, "General bootstrapping approach for rlwe-based homomorphic encryption," *IEEE Transactions on Computers*, 2023.
10. A. Putra, Prasetiyo, Y. Chen, J. Kim, and J.-Y. Kim, "Strix: An end-to-end streaming architecture with two-level ciphertext batching for fully homomorphic encryption with programmable bootstrapping," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1319–1331.
11. P. Sathishkumar, K. Pugalarasan, C. Ponnparamaguru, and M. Vasanthkumar, "Improving healthcare data security using cheon-kim-kim-song (ckks) homomorphic encryption," in *2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS)*, vol. 1. IEEE, 2024, pp. 1–6.
12. Y. Pan, Z. Chao, W. He, Y. Jing, L. Hongjia, and W. Liming, "Fedshe: privacy preserving and efficient federated learning with adaptive segmented ckks homomorphic encryption," *Cybersecurity*, vol. 7, no. 1, p. 40, 2024.

13. F.-J. Valera-Rodriguez, P. Manzanares-Lopez, and M.-D. Cano, "Empirical study of fully homomorphic encryption using microsoft seal," *Applied Sciences*, vol. 14, no. 10, p. 4047, 2024.

14. A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee *et al.*, "Openfhe: Open-source fully homomorphic encryption library," in *proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography*, 2022, pp. 53–63.

15. C. Aguilar Melchor, M.-O. Kilijian, C. Lefebvre, and T. Ricosset, "A comparison of the homomorphic encryption libraries helib, seal and fv-nfllib," in *Innovative Security Solutions for Information Technology and Communications: 11th International Conference, SecITC 2018, Bucharest, Romania, November 8–9, 2018, Revised Selected Papers 11.* Springer, 2019, pp. 425–442.

16. L. Jiang and L. Ju, "Fhebench: Benchmarking fully homomorphic encryption schemes," *arXiv preprint arXiv:2203.00728*, 2022.

17. C. Gouert, D. Mouris, and N. Tsoutsos, "Sok: New insights into fully homomorphic encryption libraries via standardized benchmarks," *Proceedings on privacy enhancing technologies*, 2023.

18. I. Iliashenko and V. Zucca, "Faster homomorphic comparison operations for bgv and bfv," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 3, pp. 246–264, 2021.

19. P.-E. Clet, O. Stan, and M. Zuber, "Bfv, ckks, tfhe: Which one is the best for a secure neural network evaluation in the cloud?" in *Applied Cryptography and Network Security Workshops: ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, June 21–24, 2021, Proceedings.* Springer, 2021, pp. 279–300.

20. J. Ma, C. Xu, and L. W. Wills, "Pytfhe: An end-to-end compilation and execution framework for fully homomorphic encryption applications," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).* IEEE, 2023, pp. 24–34.

21. H. Zhu, T. Suzuki, and H. Yamana, "Performance comparison of homomorphic encrypted convolutional neural network inference among helib, microsoft seal and openfhe," in *2023 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE).* IEEE, 2023, pp. 1–7.

22. Y. Zhai, M. Ibrahim, Y. Qiu, F. Boemer, Z. Chen, A. Titov, and A. Lyashevsky, "Accelerating encrypted computing on intel gpus," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS).* IEEE, 2022, pp. 705–716.