Fully Homomorphic Encryption for Computer Architects: A Fundamental Characterization Study

Subhankar Pal*, Karthik Swaminathan*, Ehud Aharoni†, Eyal Kushnir†, Nir Drucker†, Hayim Schaul†
Alper Buyuktosunoglu*, Omri Soceanu†, Pradip Bose*
*IBM T.J. Watson Research Center †IBM Research – Israel

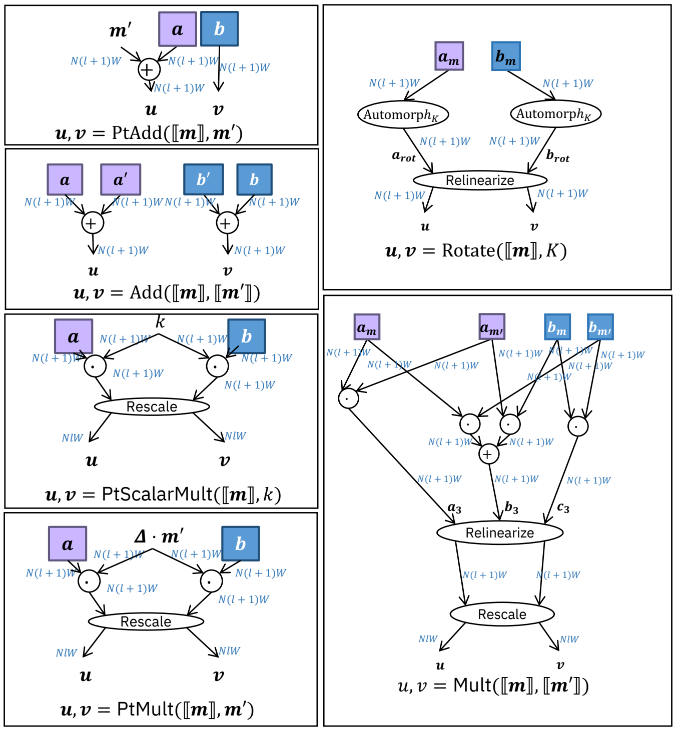
***Abstract*—Fully Homomorphic Encryption (FHE) has become a key enabler for secure computation with end-to-end quantum-safe encryption. Unlike previous cryptographic paradigms which necessitate decryption of the data in order to carry out computations, FHE makes it possible to compute in the encrypted space, thus eliminating potential vulnerabilities that may arise from storing unencrypted data on untrusted third-party cloud servers. However, the aforementioned encrypted computation operations, while quantum-safe, come with substantial performance and energy overheads, in comparison to the equivalent unencrypted operations on plaintext. Several algorithmic improvements have significantly reduced this overhead, and yet encrypted computations remain 3–4 orders of magnitude slower and less efficient than the corresponding unencrypted operations. This has resulted in the emergence of several novel architecture proposals, software libraries and runtimes with the aim of bridging this gap.**

In this paper, we carry out a detailed characterization of popular FHE libraries on a server-class CPU and GPU across key FHE primitives, and on an end-to-end application use case for detecting credit card fraud through inference on encrypted data. We study the effect of both algorithmic parameters such as the security level, polynomial degree, integer/floating point precision and computational depth, as well as system-level parameters, such as types of processing engine and number of allocated threads, on the performance and energy efficiency of execution. We hope that this characterization can help bridge the gap between cryptographers and system architects in order to realize a commercially-viable system enabled with end-to-end Fully Homomorphic Encryption.

I. Introduction

In the age of ubiquitous computing and data-driven applications, the need to protect sensitive information while still allowing for computation on that data has become paramount. Fully Homomorphic Encryption (FHE) has emerged as a promising cryptographic technique that addresses this fundamental challenge. Unlike traditional encryption schemes that require the data to be decrypted before any computation can take place, FHE allows for data to remain encrypted while performing indefinite number of computations on it.

In this work, we present a detailed characterization of *(i)* fundamental FHE primitives, and *(ii)* a Privacy-Preserving Machine Learning (PPML) inference workload for detecting fraud in credit card transactions. We characterize the strong and weak scaling trends and power consumption of several state-of-the-art HE libraries, namely, Lattigo [11], OpenFHE [2], HEaaN [4], SEAL [10]. Our characterization is performed using HElayers [1], which is a software development kit (SDK) for the practical and efficient execution of encrypted workloads using FHE and is designed to enable application developers and data scientists to apply privacy-preserving techniques without requiring domain knowledge of cryptography. HELayers uses data structures known as tile tensors that enable the implementation of generalizable data packing techniques across several different HE library implementations. This allows us to carry out effective comparisons across several FHE configurations, libraries, hardware and software platforms.



m'
a
b
$N(l+1)W$
u
v
u, v = PtAdd($[\![m]\!]$, m')
a
a'
b'
b
u, v = Add($[\![m]\!]$, $[\![m']\!]$)
k
Rescale
NlW
u, v = PtScalarMult($[\![m]\!]$, k)
$\Delta \cdot m'$
Rescale
u, v = PtMult($[\![m]\!]$, m')
a_m
b_m
Automorph$_K$
Automorph$_K$
a_{rot}
b_{rot}
Relinearize
u, v = Rotate($[\![m]\!]$, K)
a_m
$a_{m'}$
b_m
$b_{m'}$
a_3
b_3
c_3
Relinearize
Rescale
u, v = Mult($[\![m]\!]$, $[\![m']\!]$)

Fig. 1. Dataflow graphs of basic CKKS-FHE primitives. Edge annotations show the data movement in terms of parameters N, l, and W.

II. FHE Primitives

All evaluations in this paper focus on Cheon-Kim-Kim-Song (CKKS) [4], which is an FHE scheme that allows for computation on fixed-point numbers. This makes it a good fit for PPML applications that operate on real numbers. In Figure 1, we present dataflow graphs of the basic CKKS-FHE primitives that form the building blocks of any FHE application. We refer the reader to [5] for further reading.

$PtAdd$ performs the element-wise addition of an encrypted message with a plaintext encoded message. $PtMultRaw$

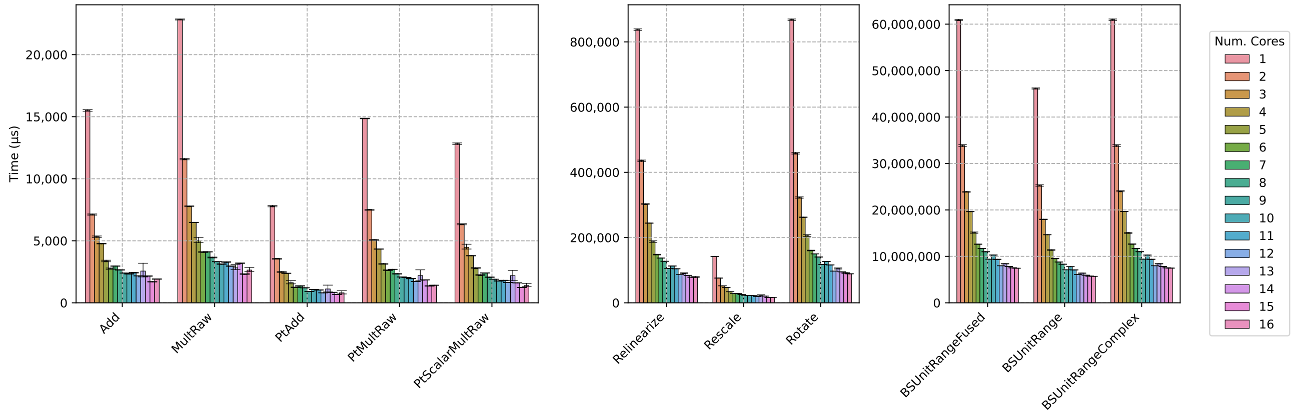


Fig. 2. Execution time profiling results for HEaaN on Intel Xeon CPU with different number of threads for $N = 128k, L = 29, \lambda = 128, fp = 51, ip = 9$. Each experiment is repeated 100 times. Error bars show 95% confidence interval.

performs the element-wise multiplication of an encrypted message with a plaintext encoded message, each with a scaling factor of Δ; however, this results in a ciphertext that has a scaling factor of Δ^2. In order to prevent the scale from growing with subsequent operations, an operation called $Rescale$ is performed to scale the result *approximately* back to Δ. $PtMultRaw$ combined with $Rescale$ constitutes $PtMult$. A variant of $PtMultRaw$, $PtScalarMultRaw$, considers multiplication of an encrypted message with a scalar plaintext value. Add performs the element-wise addition of two encrypted messages. $MultRaw$ performs the element-wise multiplication of two encrypted messages. However, this operation results in three polynomials, which need to be homomorphically converted back to two polynomials in order to prevent a growth in the number of polynomials for chained operations; this is done using $Relinearize$. A final $Rescale$ step is required for the same reason as mentioned for $PtMultRaw$. Thus, $Mult$ comprises of $MultRaw$, $Relinearize$, and $Rescale$. $Rotate$ performs a homomorphic rotation of the slots of the encrypted message. It comprises of automorphisms, which are specific permutations of the coefficients of the ciphertext, followed by a $Relinearize$.

In addition, some schemes support bootstrapping, which is a sequence of the aforementioned operations that reduces the noise within the ciphertext to an acceptable level, allowing further homomorphic operations to be performed without errors. HElayers supports several variants of bootstrapping. $BSUnitRange$ bootstraps a ciphertext that encrypts values where the real part is in the range $[-1, 1]$ and imaginary part is 0. $BSUnitRangeComplex$ bootstraps complex values where the real and imaginary parts are each in the range $[-1, 1]$. $BSUnitRangeFused$ bootstraps two real valued ciphertexts in the range $[-1, 1]$ together by packing the second ciphertext into the complex part of the first ciphertext. Note that bootstrapping is only supported by a few FHE libraries and in this work we limit our bootstrapping evaluation to HEaaN.

III. Results

We present results detailing our comparative evaluation of performance and power for different implementations of key FHE primitives across HEaaN, SEAL, Lattigo and OpenFHE libraries. We also evaluate a representative application for credit card fraud detection across these different HE libraries.

Figure 2 shows strong scaling results for the basic FHE primitives using the HEaaN library on a Xeon Gold 6258R CPU. We first note that performance scaling saturates around 16 cores for a multiplicative depth (L) of 29. *Relinearize* and *Rescale* (which internally performs automorphism + *Relinearize*) are much more complex than element-wise operations, such as additions and multiplications. The bootstrap operation is extremely expensive, taking up to a second even with several threads. Keeping Amdahl's law in mind, it is therefore crucial to design accelerators with bootstrapping in mind. This has been reflected in recent works on FHE hardware acceleration [7]–[9], [12].

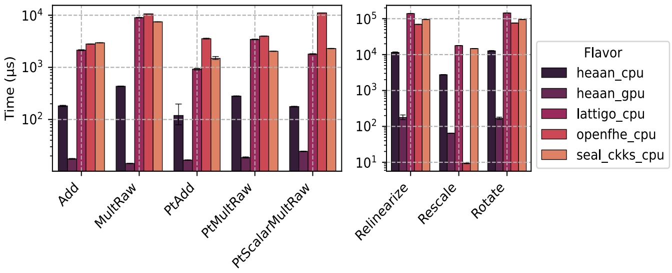


Fig. 3. Execution time profiling results on Intel Xeon CPU and NVIDIA A100 GPU with different HE library implementations for $n_{threads} = 8, N = 32k, L = 4, \lambda = 128, fp\_prec = 40, ip\_prec = 12$.

Figure 3 shows the comparison of execution times for different primitives with different HE library implementations. It is evident that the GPU implementation using the HEaaN library (run on an NVIDIA A100 GPU) is significantly faster for all the primitives. The CPU implementation of HEaaN also scales much better than the other libraries resulting in 1 to 2 orders of magnitude faster performance for this library compared to the others. Note that the same configuration may not be valid across all HE libraries, even when the security level is fixed. Hence, one must carry out a detailed design space exploration of these parameters in order to achieve a fair comparison across the different libraries. In our work, we perform a grid search through all valid (N, L, fp_prec, ip_prec) parameters for a fixed security level using valid values from [3]. Additionally, we note that other library implementations such as Lattigo and SEAL do not utilize multiple cores within the same ciphertext, or use them in a highly limited manner (as in the case of OpenFHE). In contrast, HEaaN exploits one or more levels of parallelism among coefficient-level, RNS-limb-level and/or polynomial-level.

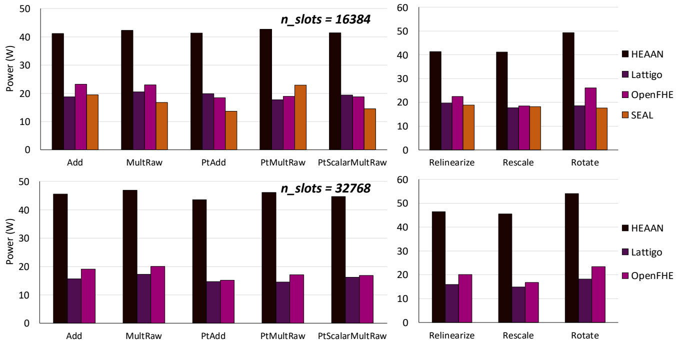


Fig. 4. Power comparison across various HE primitives for different HE library implementations using $n_{threads} = 8, N = 16384, L = 10, \lambda = 128, fp\_prec = 40, ip\_prec = 2$ (top) and $n_{threads} = 8, N = 32768, L = 32, \lambda = 128, fp\_prec = 44, ip\_prec = 2$ (bottom). Note that the selected $N = 32k$ configuration was invalid for SEAL.

We also compare the dynamic power observed while running these primitives on the Xeon CPU, as shown in Figure 4. The figures on the top and bottom represent two different configurations with 16k and 32k slots respectively, running with 8 threads on a single package consisting of 4 Hyper-threaded (HT) cores. We observe that HEaaN consumes over 2× power compared to the other libraries due to the fact that it scales to a larger number of threads. However, the significantly lower execution time observed in HEaaN results in it being the most energy-efficient implementation despite its higher power.

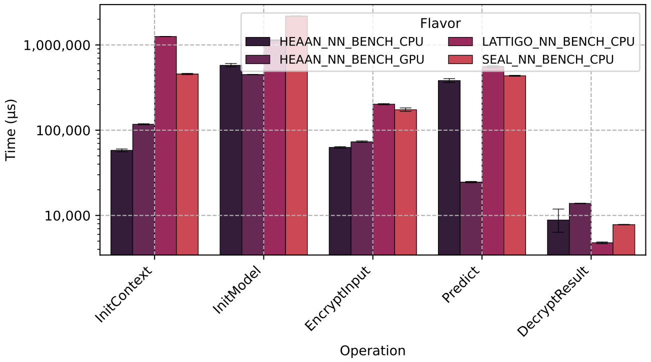


Fig. 5. Execution time profiling results for the credit card fraud detection network (encrypted weights) with different HE library implementations for $n_{threads} = 8, batch\_size = 4096, \lambda = 128$.

In addition to the FHE primitives, we also evaluate an end-to-end application for credit card fraud detection. We use the *FraudNet* model, which is a 3-layer multi-layered perceptron (MLP) for fraud detection evaluated on the Kaggle dataset, consisting of nearly 300k user-anonymized credit card transactions [6]. The application is split into various sub-operations that cover initialization, encrypting the input data, carrying out prediction on the encrypted data and finally decrypting the output. The results are shown in Figure 5. It is interesting to observe that, in contrast to the primitive, the GPU implementation is comparable to, or even slower than the corresponding CPU implementation for the initialization, encryption and decryption operations. On further investigation, we find that the GPU is slower than the CPU for *EncryptInput* as HElayers generates different optimal graphs (consisting of different number of ciphertexts) for the CPU and GPU cases.

Unlike the initialization operations which only occur at the beginning of the program, the encryption, decryption and prediction operations scale with number of inputs and are hence the focus of our analysis. We observe that encrypting the data takes 3× as long as the fraud prediction operation on a GPU, for the given batch size of 4096. Even for a batch size of 1, *EncryptInput* takes 40% of the time of *Predict*, indicating an Amdahl bottleneck. Hence, further research into realizing novel architectures for FHE-encrypted AI applications must necessarily focus on encryption and decryption, in addition to the encrypted inference operations. On an isolated experiment where we profiling the encode+encrypt of data samples during the fraud network inference, using HEaaN-CPU and HEaaN GPU libraries, we observed the GPU to be only 1.5× faster (compared to an order-of-magnitude speedup on *Predict*), with the bottleneck being the encryption. This, further, calls for research into accelerating HE encryption.

IV. Conclusion

In this paper we characterized the various operations in Fully Homomorphic Encryption (FHE) applications, based on detailed performance and power measurements on real hardware. We compared the implementations of individual FHE primitives, as well as an end-to-end credit card fraud detection application, using different standard FHE libraries, namely HEaaN, SEAL, Lattigo and OpenFHE. Based on our evaluations, we arrived at several key conclusions,

- The optimal configuration, in terms of (N, L, λ, fp, ip), can vary significantly depending on the exact HE library being used, even when the security level is fixed.
- HEaaN offers the most scalable, high-performance, and energy efficient implementation of both the evaluated FHE primitives as well as the end-to-end application (credit card fraud detection).
- While GPUs are highly effective in speeding up encrypted inference operations, the benefits are much lower for the encryption operation itself. In fact, FHE encryption can often be the key performance bottleneck, especially in a hybrid cloud-edge application scenario, which might entail encryption/decryption being implemented on-premise in a resource constrained manner.

References

[1] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkowich, D. Murik *et al.*, “Helayers: A tile tensors framework for large neural networks on encrypted data,” *arXiv preprint arXiv:2011.01805*, 2020.

[2] 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.

[3] 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.

[4] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[5] L. de Castro, R. Agrawal, R. Yazicigil, A. Chandrakasan, V. Vaikuntanathan, C. Juvekar, and A. Joshi, “Does fully homomorphic encryption need compute acceleration?” *arXiv preprint arXiv:2112.06396*, 2021.

[6] Kaggle. Credit card fraud detection. [Online]. Available: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

[7] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, “Sharp: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption,” in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA ’23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589053

[8] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, “Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse,” in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1237–1254.

[9] S. Kim, J. Kim, M. J. Kim, W. Jung, J. Kim, M. Rhu, and J. H. Ahn, “Bts: An accelerator for bootstrappable fully homomorphic encryption,” in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 711–725. [Online]. Available: https://doi.org/10.1145/3470496.3527415

[10] Microsoft. (2023) Microsoft/seal: Microsoft simple encrypted arithmetic library. [Online]. Available: https://github.com/Microsoft/SEAL

[11] C. V. Mouchet, J.-P. Bossuat, J. R. Troncoso-Pastoriza, and J.-P. Hubaux, “Lattigo: A multiparty homomorphic encryption library in go,” in *Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, no. CONF, 2020, pp. 64–70.

[12] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez, “Craterlake: A hardware accelerator for efficient unbounded computation on encrypted data,” in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 173–187. [Online]. Available: https://doi.org/10.1145/3470496.3527393