Evaluating Homomorphic Operations on a Real-World Processing-In-Memory System

Harshita Gupta* Mayank Kabra* Juan Gómez-Luna Konstantinos Kanellopoulos Onur Mutlu

ETH Zürich

Computing on encrypted data is a promising approach to reduce data security and privacy risks, with homomorphic encryption serving as a facilitator in achieving this goal. In this work, we accelerate homomorphic operations using the Processing-in-Memory (PIM) paradigm to mitigate the large memory capacity and frequent data movement requirements. Using a real-world PIM system, we accelerate the Brakerski-Fan-Vercauteren (BFV) scheme for homomorphic addition and multiplication. We evaluate the PIM implementations of these homomorphic operations with statistical workloads (arithmetic mean, variance, linear regression) and compare to CPU and GPU implementations. Our results demonstrate $50-100\times$ *speedup with a real PIM system (UPMEM) over the CPU and* $2-15\times$ *over the GPU in vector addition. For vector multiplication, the real PIM system outperforms the CPU by* $40-50\times$*. However, it lags* $10-15\times$ *behind the GPU due to the lack of native sufficiently wide multiplication support in the evaluated first-generation real PIM system. For mean, variance, and linear regression, the real PIM system performance improvements vary between* $30\times$ *and* $300\times$ *over the CPU and between* $10\times$ *and* $30\times$ *over the GPU, uncovering real PIM system trade-offs in terms of scalability of homomorphic operations for varying amounts of data. We plan to make our implementation open-source in the future.*

1. Introduction

Traditional security measures that operate on plain (unencrypted) data often expose the actual data during processing, creating security and privacy vulnerabilities. Homomorphic Encryption (HE) [1-8] addresses this by enabling calculations on encrypted data without revealing sensitive information.

A user can (1) encrypt data, and (2) send it to the server. Then, (3) computing resources in the server operate on the data without decrypting it, using HE, and (4) the encrypted results are returned to the user, preserving data privacy [5, 9-11]. However, HE is very costly due to the use of large ciphertexts and computation intensive operations [12-15]. For example, performing homomorphic multiplication on two fully homomorphic (FHE) encrypted integers may require tens of millions of operations [16-18]. The complexity is further compounded by intricate mathematical operations, as each of these operations is executed on data that can be up to $1000\times$ larger in size than the original plain data [2, 16, 19].

Recent research proposes the implementation of homomorphic operations on CPUs [17, 20-22], GPUs [4, 22-25], FPGAs [26-31], and ASICs [13, 32-37], but these implementations do not fundamentally solve the *data movement bottleneck* associated with homomorphic operations.

Processing-in-Memory (PIM), i.e., equipping memory with compute capabilities [16, 38-61], can effectively alleviate the data movement needs. Recent PIM-based HE solutions [11, 16, 62, 63] leverage high parallelism and memory bandwidth inside the memory chips for acceleration. However, there is no evaluation of homomorphic operations on real PIM systems, which have recently been introduced [38-47, 50].

To our knowledge, this study is the first to implement and evaluate homomorphic operations on a real PIM system. Using a real PIM system (UPMEM) [38, 44, 64], we accelerate the Brakerski-Fan-Vercauteren (BFV) scheme [65, 66] for homomorphic addition and multiplication. Our evaluation shows that the real PIM system accelerates the homomorphic addition operation by $50-100\times$ over a state-of-the-art CPU and by $2-15\times$ over a state-of-the-art GPU. For the homomorphic multiplication operation, the real PIM system provides a speedup of $30-50\times$ over the CPU, but lags $10-15\times$ behind the GPU due to the lack of native sufficiently wide multiplication support on the evaluated first-generation UPMEM PIM system. We also evaluate our implementation of three statistical workloads (mean, variance, linear regression) using homomorphic addition and homomorphic multiplication. In our evaluation, the real PIM system achieves up to $300\times$ speedup over the CPU for all workloads and up to $30\times$ over the GPU for arithmetic mean. However, it lags by up to $50\times$ compared to the GPU for variance and linear regression, due to the low performance of multiplication on the first real-world PIM system.

Our work makes the following contributions:

- We develop the first implementation of homomorphic addition and multiplication on a real PIM system.
- We evaluate the performance of homomorphic addition and multiplication on a real PIM system for different bit-key security levels (27-109 bits). We use three real-world statistical workloads (arithmetic mean, variance, linear regression) for evaluation.
- Our findings demonstrate the capabilities and tradeoffs of real PIM systems for efficient cryptographic operations, providing a foundation for future developments in this direction.

2. Background and Motivation

Homomorphic encryption (HE) [1-8] enables processing (e.g., addition, multiplication, rotation) on encrypted data while preserving privacy. We focus on the BFV (Brakerski-Fan-Vercauteren) scheme for HE [65, 66], but the implementation techniques that we propose are also applicable to other HE schemes (e.g., BGV [67] and CKKS [68]). HE types include

*Equal contribution.

Fully Homomorphic Encryption (FHE), Partially Homomorphic Encryption (PHE), and Somewhat Homomorphic Encryption (SHE) [69]. FHE enables unrestricted operations, PHE permits one type of operation, and SHE supports both addition and multiplication with constraints on multiplicative depth. FHE, SHE, and PHE offer different trade-offs between security and efficiency [1, 5, 7, 8, 70-72]. In this paper, we focus on SHE as it provides a balance between security and efficiency, allowing some computations (e.g., addition, multiplication) on encrypted data while still maintaining a high level of security.

HE poses two main **challenges** that limit its use in real-world applications.

1) Large memory footprint: HE schemes require very long vectors with wide elements to encode information [13]. Prior work [32] shows that multiplying 2MB ciphertexts requires 32MB of auxiliary data, and 25MB ciphertexts would require over 1.4GB of auxiliary data. This amount of auxiliary data is too large to fit on a processor-centric chip which limits the scalability and performance of HE.

2) Frequent data movement: The large amount of data that homomorphic algorithms need to operate on is moved back-and-forth between off-chip memory/storage units and compute units. Prior work [73] shows that homomorphic operations exhibit low arithmetic intensity (<1 operations/byte). As a result, in processor-centric systems, such as CPUs and GPUs, it is challenging to efficiently offset the performance and energy expenses incurred when transferring large amounts of data.

Several recent works [4, 13, 22-37] explore domain-specific architectures, such as GPUs, FPGAs, and ASICs, to accelerate homomorphic operations. These efforts have achieved significant speedups compared to CPUs. However, challenges remain in resource limitations, data movement, and practical implementation of especially ASIC-based accelerators [35].

In this work, our **goal** is to evaluate the suitability of real-world general-purpose processing-in-memory architectures to compute homomorphic operations. To this end, we implement homomorphic addition and multiplication on the UPMEM PIM system [38, 39, 44], and evaluate them on real-world statistical and machine learning workloads.

Processing-in-memory (PIM) [16, 38-61] systems can accelerate memory-intensive applications [46, 64, 74-76] by equipping memory arrays with compute capabilities. These systems can potentially address the challenge of large ciphertexts in HE algorithms by reducing the overhead of data transfers between the memory and the CPU [45, 77]. In addition to reducing data movement, PIM also offers high levels of parallelism [38, 39], which are useful for performing costly homomorphic operations. Thus, by computing directly in memory, PIM can significantly improve the performance of HE. Various real-world PIM systems have recently been introduced [38-47, 50]. These real-world PIM systems have some common characteristics [64]: there is a central host processor connected to conventional main memory, alongside PIM-enabled memory chips with processing elements that access memory with high bandwidth and low latency. In this work, we use the UPMEM PIM system [38, 39, 44, 78], which consists of fine-grained multithreaded PIM cores near DRAM banks. For more details on the UPMEM PIM system, we refer the reader to [16, 38, 39, 44, 48-60].

3. Implementation

We consider an environment where users offload computations on encrypted data to a PIM system. Users handle key generation, encryption, and decryption to guarantee their data privacy. Computation of homomorphic operations takes place in a PIM system. In this work, we implement addition and multiplication operations.

The security level of HE relies on the polynomial modulus degree [79], affecting ciphertext length, vulnerability to attacks, and noise tolerance. For instance, for 27-bit security, we need a polynomial that has 1024 27-bit coefficients, which indicates a relatively lower security level in HE. Increasing the bit length enhances security. In this work, we also evaluate 54-bit (2048-coefficient polynomial) and 109-bit (4096-coefficient polynomial) security levels. To represent 27-, 54-, and 109-bit coefficients, we use integers of 32, 64, and 128 bits, respectively. The reason is that the UPMEM PIM system that we use in our evaluation has native support for 32-bit integers.

Homomorphic Addition. We implement homomorphic addition using polynomial addition [80, 81] on the UPMEM PIM system. Each PIM thread running on a PIM core performs the element-wise addition of the coefficients of two polynomials. UPMEM PIM cores [44] support native 32-bit integer addition (`add`) and 32-bit integer addition with carry-in (`addc`), which we use to implement 64- and 128-bit addition (and can be extended to any multiple of 32 bits).

Homomorphic Multiplication. We implement homomorphic multiplication using polynomial multiplication and polynomial addition [82-85]. Each PIM thread running on a PIM core performs the polynomial multiplication and polynomial addition of the coefficients of two polynomials to generate the desired result. For 32-bit coefficients, we rely on the compiler-generated 32-bit shift-and-add based multiplication.[1] For 64- and 128-bit multiplications, we divide the bits into chunks of 32-bits and apply the Karatsuba algorithm [86], which requires less operations than the traditional multiplication algorithm. We do not incorporate Number Theoretic Transform (NTT) [87, 88] techniques to optimize multiplication. We leave them for future work.

Statistical Workloads. We implement three statistical workloads (arithmetic mean, variance, linear regression) using homomorphic addition and homomorphic multiplication techniques. The arithmetic mean [89, 90] workload employs polynomial addition performed on the UPMEM PIM cores and scalar division performed on the host processor. The variance [91, 92] workload uses polynomial multiplication which is performed on the UPMEM PIM cores and a final scalar di-

[1]The UPMEM PIM system performs 8-bit and 16-bit multiplications using the native 8-bit hardware multipliers, but employs a software-based shift-and-add algorithm for higher bit widths [38, 44, 64].

vision performed on the host processor. Similarly, linear regression [93, 94] workload uses both polynomial addition and multiplication to perform the vector-matrix multiplication, which is employed on the UPMEM PIM cores.

4. Evaluation

4.1. Methodology

We evaluate homomorphic addition and multiplication on a first-generation UPMEM PIM system [38, 39, 44, 78], a 4-core Intel i5-8250U CPU [95], and an NVIDIA A100 GPU [96]. The UPMEM system contains 2,524 PIM cores (running at 425 MHz) and 158GB of PIM-enabled memory with a total bandwidth of 2,145 GB/s. We compare our PIM implementations to our own custom CPU and GPU implementations. We also compare to an optimized CPU implementation, the SEAL CPU library [79], which leverages the Residue Number System (RNS) [97] and the Number Theoretic Transform (NTT) [98] implementations for faster operations.

We first evaluate microbenchmarks for vector addition and vector multiplication (Section 4.2). We experiment with different numbers of ciphertexts between 20,480 to 327,680 for addition, and between 5,120 and 81,920 for multiplication. We run experiments for integers of 32 bits (27-bit coefficients), 64 bits (54-bit coefficients), and 128 bits (109-bit coefficients). We then evaluate SHE implementations of three statistical workloads (arithmetic mean, variance, linear regression) that employ our PIM-based homomorphic encryption operations (Section 4.3). We plan to open-source all workloads.

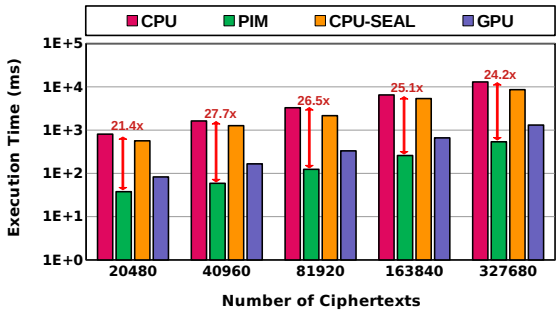
4.2. Vector Addition and Multiplication

Figure 1 shows the execution time of vector addition (1(a)) and multiplication (1(b)) on homomorphically encrypted ciphertexts for our real-world UPMEM PIM-based implementation (PIM), our custom CPU and GPU implementations, and the SEAL library (CPU-SEAL). The figure also shows the speedup of PIM over the custom CPU implementations.

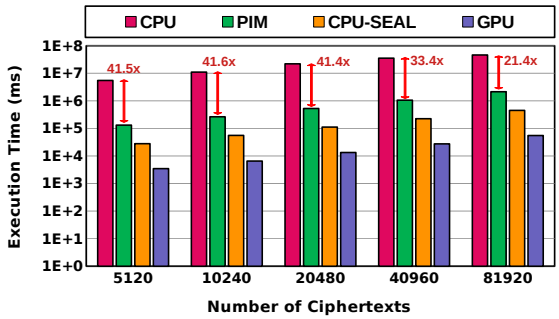
We make several observations about these experimental results. First, the performance of PIM implementations saturates at 11 or more PIM threads (not shown in Figure 2). This is in line with the observations in prior works [38, 45, 64]. Second, the large number of PIM cores and the native support for 32-bit integer addition in PIM cores result in fast execution of vector addition on the PIM system. Figure 1(a) shows the results for 128-bit addition. The trends are the same for 32-bit and 64-bit addition. For 32-, 64-, and 128-bit addition, the PIM implementation outperforms CPU, CPU-SEAL, and GPU by $20-150\times$, $35-80\times$, and $15-50\times$, respectively.

Key Takeaway 1. *With native hardware support for 32-bit integer addition and large number of PIM cores, the UPMEM PIM system outperforms CPU and GPU for homomorphic addition.*

Third, vector multiplication on the UPMEM PIM system suffers from the lack of native 32-bit multiplication hardware, as multiplication wider than 16 bits is based on compiler generated shift-and-add algorithm. Figure 1(b) shows the results for 128-bit multiplication. We observe similar trends for 32-bit and 64-bit multiplication. For 32-, 64-, and 128-bit multiplication,



(a) **128-bit ciphertext vector addition**



(b) **128-bit ciphertext vector multiplication**

Figure 1: Execution time (ms) of ciphertext vector addition (a) and vector multiplication (b) for 128-bit (109-bit) wide polynomial coefficients on CPU, PIM, CPU-SEAL and GPU.

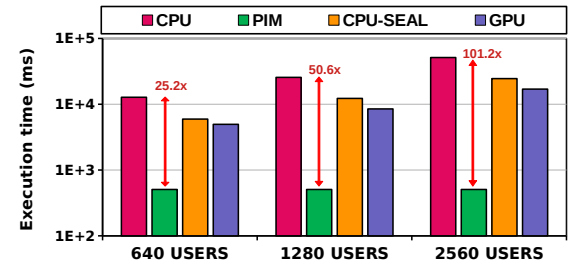
the PIM implementation outperforms CPU by $40-50\times$, and CPU-SEAL for 32 bits by $2\times$. However, the PIM implementation is $12-15\times$ slower than GPU, and $2-4\times$ slower than CPU-SEAL for 64 and 128 bits.

Key Takeaway 2. *The lack of native support for 32-bit integer multiplication hampers the performance of PIM for homomorphic multiplication. Future PIM systems with native 32-bit multiplication hardware could potentially outperform CPUs and GPUs.*

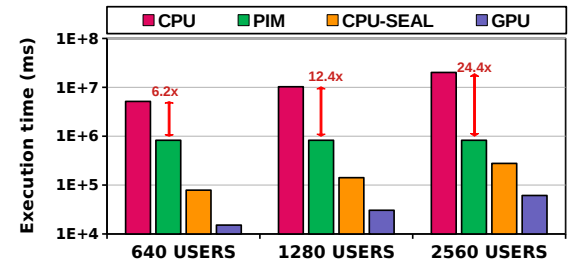
4.3. Statistical Workloads

We implement and evaluate the performance of three real-world statistical workloads (arithmetic mean, variance, linear regression) that utilize homomorphic addition and multiplication for the CPU, real-world PIM, CPU-SEAL and GPU implementations. Figure 2 shows the execution times of the three workloads on CPU, PIM, CPU-SEAL, and GPU. For arithmetic mean and variance, we evaluate scenarios with 640, 1280, and 2560 users. For linear regression, we evaluate 640 users, and 32 and 64 ciphertexts per user (data samples with 3 features).

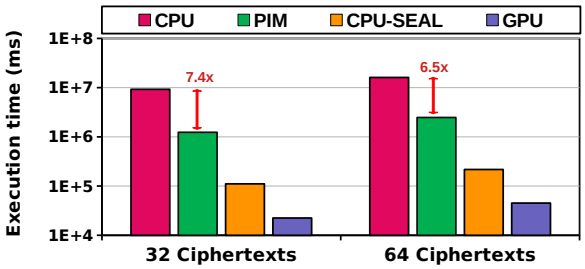
We make several observations from Figure 2. First, arithmetic mean uses only homomorphic addition. As a result, PIM is significantly faster than CPU, CPU-SEAL, and GPU. Figure 2(a) shows PIM speedups of $25-100\times$ over CPU, $11-50\times$ over CPU-SEAL, and $9-34\times$ over GPU for different numbers of users. Second, as variance uses the square operation (i.e., homomorphic multiplication of two equal numbers), the PIM implementation is heavily burdened by the slow multiplication. In Figure 2(b), we observe that PIM outperforms only the custom CPU implementation (by $6-25\times$) for different numbers of users. CPU-SEAL and GPU are, respectively, $2-10\times$ and $13-50\times$ faster than PIM. Third, for linear regression the



(a) **Arithmetic Mean**



(b) **Variance**



(c) **Linear Regression**

Figure 2: Execution time (ms) of aritmetic mean (a), variance (b), and linear regression (c) for 128-bit (109-bit) wide polynomial coefficients on CPU, PIM, CPU-SEAL and GPU.

trends are the same as for variance, given that linear regression also uses multiplication heavily. Figure 2(c) shows that PIM is only faster than the custom CPU implementation (by 7.5×) for 32 ciphertexts. CPU-SEAL and GPU are, respectively, 11.4× and 54.9× faster than PIM for 64 ciphertexts. Fourth, we observe that PIM execution time remains constant for different numbers of users. This is achieved by dynamically adjusting the utilization of PIM cores, which is particularly beneficial in our experiments as they involve a large number of users. This approach differs from CPUs and GPUs, which have a limited number of cores and must use them regardless of the number of users in our experiment.

Key Takeaway 3. *The computational power of PIM scales with memory capacity [99, 100] via the addition of more memory banks and corresponding PIM cores. This memory-capacity-proportional performance scalability provided by PIM holds promise for accommodating expanding numbers of users and more parallel computations as memory capacity grows.*

5. Related Work

Several recent works explore the suitability of real-world processing-in-memory (PIM) architectures [16, 38-61] to accelerate a variety of memory-intensive tasks [46, 64, 74-76]. To our knowledge, this is the first work to explore the use of a real PIM system to accelerate homomorphic operations.

Acceleration of homomorphic operations on GPUs, FPGAs, or ASICs is the subject of various recent works. All these processor-centric techniques suffer from data movement bottlenecks between memory and compute units. GPUs can accelerate HE schemes [4, 22-25]. However, GPUs suffer from high power consumption for homomorphic operations [101, 102]. FPGAs can also accelerate homomorphic operations [26-31], but they are limited in hardware resources and suffer from data movement bottlenecks [103, 104]. Several recent works propose ASIC designs [13, 32-37] for CKKS algorithms, but they are only evaluated in simulation. Similarly, PIM-based solutions [11, 16, 105] for accelerating homomorphic operations are also limited to simulation.

6. Conclusion

We presented initial results on the use of a real-world general-purpose PIM architecture (i.e., the UPMEM PIM system [38, 50]) to accelerate homomorphic operations. Our PIM implementations of homomorphic addition, multiplication and statistical workloads (mean, variance, linear regression) show great promise when compared to CPU and GPU implementations, as long as the necessary integer operations are natively supported by the PIM hardware. We aim to implement more homomorphic operations and optimizations as future work.

References

[1] M. Ogburn *et al.*, "Homomorphic Encryption," *Procedia Computer Science*, 2013.
[2] D. Tourky *et al.*, "Homomorphic Encryption The "Holy Grail" of Cryptography," in *ICCC 2016.*
[3] C. Gentry and S. Halevi, "Implementing Gentry's Fully Homomorphic Encryption Scheme," in *EUROCRYPT 2011.*
[4] A. Al Badawi *et al.*, "Towards The Alexnet Moment For Homomorphic Encryption: HCNN, The First Homomorphic CNN on Encrypted Data With GPUs," *TETC 2020.*
[5] C. Gentry, "Fully Homomorphic Encryption using Ideal Lattices," in *STOC 2009.*
[6] M. Van Dijk *et al.*, "Fully Homomorphic Encryption over the Integers," in *EUROCRYPT 2010.*
[7] D. Boneh *et al.*, "Fully Key-Homomorphic Encryption, Arithmetic Circuit ABE, and Compact Garbled Circuits," IACR 2014.
[8] D. Boneh *et al.*, "Private Database Queries using Somewhat Homomorphic Encryption," in *ACNS*, 2013.
[9] C. Moore *et al.*, "Practical Homomorphic Encryption: A Survey," in *ISCAS 2014.*
[10] P. Chaudhary *et al.*, "Analysis and Comparison of Various Fully Homomorphic Encryption Techniques," in *GUCON 2019.*
[11] S. Gupta and T. Š. Rosing, "Accelerating Fully Homomorphic Encryption with Processing-in-memory," in *DAC 2021.*
[12] N. Samardzic *et al.*, "F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption," in *MICRO 2021.*
[13] N. Samardzic *et al.*, "Craterlake: A Hardware Accelerator for Efficient Unbounded Computation on Encrypted Data," in *ISCA 2022.*
[14] B. Alaya *et al.*, "Homomorphic Encryption Systems Statement: Trends and Challenges," *CSR 2020.*
[15] K. El Makkaoui *et al.*, "Challenges of Using Homomorphic Encryption to Secure Cloud Computing," in *CloudTech 2015.*
[16] S. Gupta *et al.*, "MemFHE: End-to-end Computing with Fully Homomorphic Encryption in Memory," *TECS 2022.*
[17] X. Cao *et al.*, "Optimised Multiplication Architectures For Accelerating Fully Homomorphic Encryption," *TC 2015.*
[18] Y. Su *et al.*, "A Highly Unified Reconfigurable Multicore Architecture to Speed-up NTT/INTT for Homomorphic Polynomial Multiplication," *TVLSI 2022.*
[19] Y. Doröz *et al.*, "Homomorphic AES Evaluation using the Modified LTV Scheme," *DCC 2016.*
[20] A. C. Mert *et al.*, "Design and Implementation of Encryption/Decryption Architectures for BFV Homomorphic Encryption Scheme," *TVLSI 2019.*
[21] S. Meftah *et al.*, "Towards High Performance Homomorphic Encryption for Inference Tasks on CPU: An MPI Approach," *FGCS 2022.*
[22] T. Morshed *et al.*, "CPU and GPU Accelerated Fully Homomorphic Encryption," in *HOST 2020.*

[23] W. Dai and B. Sunar, "cuHE: A Homomorphic Encryption Accelerator Library," in *BalkanCryptSec 2016*.
[24] A. Al Badawi *et al.*, "Multi-GPU Design and Performance Evaluation of Homomorphic Encryption on GPU Clusters," *TPDS 2020*.
[25] W. Dai *et al.*, "Accelerating NTRU based Homomorphic Encryption using GPUs," in *HPEC 2014*.
[26] W. Wang and H. Xinming, "FPGA Implementation Of a Large-number Multiplier for Fully Homomorphic Encryption," in *ISCAS 2013*.
[27] R. Agrawal *et al.*, "FAB: An FPGA-based Accelerator for Bootstrappable Fully Homomorphic Encryption," in *HPCA 2023*.
[28] D. B. Cousins *et al.*, "Designing an FPGA-accelerated Homomorphic Encryption Co-processor," *TETC 2016*.
[29] D. B. Cousins *et al.*, "An FPGA Co-processor Implementation of Homomorphic Encryption," in *HPEC 2014*.
[30] I. Syafalni *et al.*, "Efficient Homomorphic Encryption Accelerator with Integrated PRNG using Low-cost FPGA," *IEEE Access 2022*.
[31] C. Jayet-Griffon *et al.*, "Polynomial Multipliers for Fully Homomorphic Encryption on FPGA," in *ReConFig 2015*.
[32] A. Feldmann *et al.*, "F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption," *MICRO 2021*.
[33] Y. Yang *et al.*, "Poseidon: Practical Homomorphic Encryption Accelerator," in *HPCA 2023*.
[34] E. Öztürk *et al.*, "A Custom Accelerator for Homomorphic Encryption Applications," *TC 2016*.
[35] S. Kim *et al.*, "BTS: An Accelerator for Bootstrappable Fully Homomorphic Encryption," in *ISCA 2022*.
[36] M. Nabeel *et al.*, "CoFHEE: A Co-processor for Fully Homomorphic Encryption Execution," in *DATE 2023*.
[37] X. Cao *et al.*, "High-speed Fully Homomorphic Encryption over the Integers," in *FC 2014*.
[38] J. Gómez-Luna *et al.*, "Benchmarking Memory-centric Computing Systems: Analysis of Real Processing-in-Memory Hardware," in *IGSC 2021*.
[39] J. Gómez-Luna *et al.*, "Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-memory System," *IEEE Access 2022*.
[40] J. H. Kim *et al.*, "Aquabolt-XL: Samsung HBM2-PIM with In-memory Processing for ML Accelerators and Beyond," in *HCS 2021*.
[41] J. H. Kim *et al.*, "Aquabolt-XL HBM2-PIM, LPDDR5-PIM with In-memory Processing, and AXDIMM with Acceleration Buffer," *IEEE MICRO 2022*.
[42] D. Lee *et al.*, "Improving In-Memory Database Operations with Acceleration DIMM (AxDIMM)," in *DaMoN*, 2022.
[43] L. Ke *et al.*, "Near-Memory Processing in Action: Accelerating Personalized Recommendation with AxDIMM," *IEEE Micro*, 2021.
[44] "UPMEM SDK," https://sdk.upmem.com/2023.1.0/.
[45] J. Gómez-Luna *et al.*, "Machine Learning Training on a Real Processing-in-Memory System," in *ISVLSI 2022*.
[46] M. Item *et al.*, "TransPimLib: Efficient Transcendental Functions for Processing-in-Memory Systems," in *ISPASS*, 2023.
[47] J. Kim and Y. Kim, "HBM: Memory Solution for Bandwidth-hungry Processors," in *HCS*, 2014.
[48] S. Ghose *et al.*, "The Processing-in-Memory Paradigm: Mechanisms to Enable Adoption," in *Beyond-CMOS Technologies for Next Generation Computer Design 2019*.
[49] G. F. Oliveira *et al.*, "DAMOV: A New Methodology And Benchmark Suite For Evaluating Data Movement Bottlenecks," *IEEE Access 2021*.
[50] F. Devaux, "The True Processing in Memory Accelerator," in *HCS*, 2019.
[51] S. Ghose *et al.*, "Processing-in-memory: A Workload-driven Perspective," *IBM JRD 2019*.
[52] O. Mutlu *et al.*, "A Modern Primer on Processing-in-memory," in *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann 2022*.
[53] V. Seshadri *et al.*, "Ambit: In-memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO 2017*.
[54] H. S. Stone, "A Logic-in-Memory Computer," *IEEE TC*, 1970.
[55] W. H. Kautz, "Cellular Logic-in-Memory Arrays," *IEEE TC*, 1969.
[56] A. Farmahini-Farahani *et al.*, "DRAMA: An Architecture for Accelerated Processing-near-memory," *IEEE Computer Architecture Letters*, 2014.
[57] G. Singh *et al.*, "A Review of Near-memory Computing Architectures: Opportunities and Challenges," in *Euromicro DSD 2018*.
[58] Y. Kwon *et al.*, "TensorDIMM: A Practical Near-memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning," in *MICRO 2019*.
[59] S. F. Yitbarek *et al.*, "Exploring Specialized Near-memory Processing for Data-intensive Operations," in *DATE 2016*.
[60] N. Hajinazar *et al.*, "SIMDRAM: A Framework for Bit-serial SIMD Processing-using-DRAM," in *ASPLOS 2021*.
[61] C. Lim *et al.*, "Design and Analysis of a Processing-in-DIMM Join Algorithm: A Case Study with UPMEM DIMMs," *ACM SIGMOD*, 2023.
[62] W. Li *et al.*, "Leveraging Memory PUFs and PIM-based Encryption to Secure Edge Deep Learning Systems," in *VTS 2019*.
[63] D. Reis *et al.*, "Computing-in-memory for Performance and Energy-efficient Homomorphic Encryption," *TVLSI 2020*.
[64] J. Gómez-Luna *et al.*, "An Experimental Evaluation of Machine Learning Training on a Real Processing-in-Memory System," *arXiv preprint arXiv:2207.07886*, 2022.
[65] S. Halevi *et al.*, "An Improved RNS Variant of the BFV Homomorphic Encryption Scheme," in *CT-RSA 2019*.
[66] F. Wibawa *et al.*, "BFV-Based Homomorphic Encryption for Privacy-Preserving CNN Models," *Cryptography 2022*.
[67] J. Mono *et al.*, "Finding and Evaluating Parameters for BGV," *Cryptology ePrint Archive*, 2022.
[68] J. H. Cheon *et al.*, "Remark on the Security of CKKS Scheme in Practice," *Cryptology ePrint Archive*, 2020.
[69] A. Acar *et al.*, "A Survey on Homomorphic Encryption Schemes: Theory and Implementation," *ACM CSUR 2018*.
[70] A. B. Alexandru *et al.*, "Cloud-based Quadratic Optimization with Partially Homomorphic Encryption," *IEEE TAC*, 2020.
[71] I. Damgård *et al.*, "Multiparty Computation From Somewhat Homomorphic Encryption," in *CRYPTO 2012*.
[72] M. Yasuda *et al.*, "Practical Packing Method in Somewhat Homomorphic Encryption," in *DPM 2013 and SETOP 2013*.
[73] L. de Castro *et al.*, "Does Fully Homomorphic Encryption Need Compute Acceleration?" *IACR 2021*.
[74] S. Diab *et al.*, "A Framework for High-throughput Sequence Alignment using Real Processing-in-Memory Systems," *Bioinformatics*, 2023.
[75] C. Giannoula *et al.*, "Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-in-Memory Architectures," in *SIGMETRICS*, 2022.
[76] G. F. Oliveira *et al.*, "Accelerating Neural Network Inference with Processing-in-DRAM: From the Edge to the Cloud," *IEEE Micro 2022*.
[77] O. Mutlu *et al.*, "Processing Data Where It Makes Sense: Enabling In-memory Computation," *MICPRO*, 2019.
[78] J. Gómez-Luna *et al.*, "Evaluating Machine Learning Workloads on Memory-Centric Computing Systems," in *ISPASS 2023*.
[79] "Microsoft SEAL," https://www.microsoft.com/en-us/research/project/microsoft-seal/.
[80] E. D. Sontag, "Real Addition and the Polynomial Hierarchy," *IPL 1985*.
[81] R. Zippel, *"Effective Polynomial Computation"*. SSBM 1993.
[82] R. T. Moenck, "Practical Fast Polynomial Multiplication," in *SYMSAC 1976*.
[83] D. Harvey *et al.*, "Faster Polynomial Multiplication over Finite Fields," *JACM 2017*.
[84] D. Harvey *et al.*, "Polynomial multiplication over finite fields in time," *JACM 2022*.
[85] D. D. Chen *et al.*, "High-speed Polynomial Multiplication Architecture for Ring-LWE and SHE Cryptosystems," *TCAS-I 2014*.
[86] C. Eyupoglu, "Performance Analysis of Karatsuba Multiplication Algorithm for Different Bit Lengths," *Procedia: SBS 2015*.
[87] M. Bisheh-Niasar *et al.*, "High-Speed NTT-based Polynomial Multiplication Accelerator For CRYSTALS-Kyber Post-Quantum Cryptography," *ICAR 2021*.
[88] T. Fritzmann and J. Sepúlveda, "Efficient and Flexible Low-Power NTT for Lattice-Based Cryptography," in *HOST 2019*.
[89] E. Jacquier *et al.*, "Geometric or Arithmetic Mean: A Reconsideration," *FAJ*, 2003.
[90] T.-H. Zhao *et al.*, "On Approximating the Quasi-arithmetic Mean," *JIA*, 2019.
[91] M. G. Larson, "Analysis of Variance," *Circulation*, 2008.
[92] M. Davidian *et al.*, "Variance Function Estimation," *JASA*, 1987.
[93] X. Su *et al.*, "Linear Regression," *WIREs Comp Stats*, 2012.
[94] D. Maulud *et al.*, "A Review on Linear Regression Comprehensive in Machine Learning," *JASTT*, 2020.
[95] Intel, "Intel® Core™ i5-8250U Processor," https://ark.intel.com/content/www/us/en/ark/products/124967/intel-core-i58250u-processor-6m-cache-up-to-3-40-ghz.html, 2017.
[96] NVIDIA, "NVIDIA A100 Tensor Core GPU Architecture. White Paper," https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf, 2020.
[97] M. Gomathisankaran *et al.*, "HORNS: A Homomorphic Encryption Scheme for Cloud Computing using Residue Number System," in *IEEE CISS 2011*.
[98] A. W. Mohsen *et al.*, "Performance Analysis of Number Theoretic Transform for Lattice-based Cryptography," in *ICCES 2018*.
[99] J. Ahn *et al.*, "A Scalable Processing-in-memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.
[100] J. Ahn *et al.*, "Retrospective: A Scalable Processing-in-memory Accelerator for Parallel Graph Processing," *arXiv preprint arXiv:2306.15577*, 2023.
[101] S. Tan *et al.*, "CryptGPU: Fast Privacy-preserving Machine Learning on the GPU," in *SP 2021*.
[102] W. Wang *et al.*, "Exploring the Feasibility of Fully Homomorphic Encryption," *IEEE TC 2013*.
[103] S. S. Roy *et al.*, "HEPCloud: An FPGA-based Multicore Processor for FV Somewhat Homomorphic Function Evaluation," *TC 2018*.
[104] S. S. Roy *et al.*, "FPGA-based High-performance Parallel Architecture For Homomorphic Computing on Encrypted Data," in *HPCA 2019*.
[105] H. Nejatollahi *et al.*, "CryptoPIM: In-memory Acceleration for Lattice-based Cryptographic Hardware," in *DAC 2020*.