PriML: An Electro-Optical Accelerator for Private Machine Learning on Encrypted Data

Mengxin Zheng † Fan Chen † Lei Jiang † Qian Lou ⋆
† Indiana University Bloomington ⋆ University of Central Florida
† {zhengme, fc7, jiang60}@iu.edu ⋆ qian.lou@ucf.edu

2023 24th International Symposium on Quality Electronic Design (ISQED) | 979-8-3503-3475-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISQED57927.2023.10129302

***Abstract*—The widespread use of machine learning is changing our daily lives. Unfortunately, clients are often concerned about the privacy of their data when using machine learning-based applications. To address these concerns, the development of privacy-preserving machine learning (PPML) is essential. One promising approach is the use of fully homomorphic encryption (FHE) based PPML, which enables services to be performed on encrypted data without decryption. Although the speed of computationally expensive FHE operations can be significantly boosted by prior ASIC-based FHE accelerators, the performance of key-switching, the dominate primitive in various FHE operations, is seriously limited by their small bit-width datapaths and frequent matrix transpositions. In this paper, we present an electro-optical (EO) PPML accelerator, PriML, to accelerate FHE operations. Its 512-bit datapath supporting 510-bit residues greatly reduces the key-switching cost. We also create an in-scratchpad-memory transpose unit to fast transpose matrices. Compared to prior PPML accelerators, on average, PriML reduces the latency of various machine learning applications by $> 94.4\%$ and the energy consumption by $> 95\%$.**

***Index Terms*—Privacy-preserving machine learning, Fully homomorphic encryption, Electro-optical FHE accelerator**

I. Introduction

Machine Learning is increasingly being used in many fields, transforming our daily lives, such as product recommendation [1], image recognition [2], language translation [3], and sentiment analysis [4]. However, when using machine learning in privacy-sensitive fields such as finance or healthcare, data privacy must be protected. As such, various laws and regulations [5], such as the California Consumer Privacy Act and the EU General Data Protection Regulation, have been established to govern how clients' personal data can be stored, transferred, and processed. This has led to an increased demand for privacy-preserving computing solutions that protect data confidentiality.

Fully homomorphic encryption (FHE) [6]–[9] emerges as one of the most promising solutions to guaranteeing data privacy by allowing computations to directly happen on ciphertexts. Though trusted execution environments protect both code and data, they are susceptible to side-channel attacks [10]. Secure multi-party computation (SMPC) [11] protects data privacy in an interactive way. However, SMPC requires the continuous online presence of all involved parties, each of which needs a high-bandwidth network connection. For instance, a SMPC-based private network [11] has to exchange $2GB$ data for only an inference on an encrypted CIFAR-10 image. In contrast, FHE enables a client to encrypt his/her data and to send only the ciphertexts to an untrusted server, which can perform computations on the ciphertexts even when the client is offline. Whenever the client comes back online, he/she can retrieve and decrypt the encrypted results. *The client and the server do not have to communicate during FHE computations*. Since encrypted intermediate results in the server are not shared, there is less risk of leakage. Therefore, we use FHE to enable privacy-preserving machine learning (PPML) in this paper.

FHE operations are extremely time-consuming, i.e., one FHE bootstrapping costs several seconds on a CPU. Recent ASIC-based hardware accelerators [12]–[17] greatly speed up various FHE operations. However, *the performance of prior FHE accelerators is limited by their small bit-width datapaths and frequent matrix transpositions*. An FHE ciphertext consists of two polynomials of large degrees (e.g., several thousand) with large integer coefficients (e.g., several hundred bits). To efficiently compute with polynomials, FHE schemes (e.g., CKKS [12]) adopt Residue Number System (RNS) [12], and Number Theoretic Transform (NTT). First, to compute with large integer coefficients, RNS divides each coefficient into multiple smaller bit-width (e.g., 60-bit) residues, each of which can be processed by the datapath of prior FHE accelerators. The latency of an FHE multiplication, rotation, or bootstrapping is dominated by expensive key-switching (KS) primitives [7] that makes output ciphertexts to be encrypted by the same secret key as the input ciphertext(s). *The computational overhead of KS*, i.e., the number of NTTs, additions, and multiplications in a KS, *greatly increases with an enlarging number of residues*. Second, for two large degree-N polynomials, NTT and inverse NTT (iNTT) reduces the time complexity of their multiplications to $\mathcal{O}(N \log N)$. But it is difficult to perform an (i)NTT, i.e., NTT or iNTT, on a large degree-N polynomial directly. Prior FHE accelerators [12], [13] place it as an $n \times n$ matrix, where $N = n^2$, perform an (i)NTT on each row, multiply the matrix with some constants, transpose the matrix, and perform an (i)NTT on each row again. As a result, *frequent matrix transpositions greatly prolong the latency of KS in various FHE operations by introducing huge volumes of on-chip memory traffic during (i)NTTs*.

We propose an electro-optical (EO) FHE accelerator, *PriML*, to support a large bit-width datapath and free its computing units from matrix transpositions. Our contribution is summarized as:

- **A 512-bit EO CU**. We propose a 512-bit EO Computing

Units (CUs) built upon ultra-fast EO integer adders and multipliers to process polynomials with large coefficients. Its 512-bit datapath greatly reduces the number of residues, and thus the computational overhead of each key-switching primitive.

- **An in-SPM TU**. We build a low-power eDRAM-based on-chip scratchpad (SPM) system, where we present an in-SPM Transpose Unit (TU) to transpose matrices for (i)NTTs without sending matrices having low spatial locality to the CUs via H-trees of the SPM or NoCs.
- **Latency and energy efficiency**. Compared to prior ASIC-based FHE accelerators, PriML reduces the latency of various FHE applications by $> 94.4\%$ and the energy consumption by $> 95\%$.

II. Background

A. Fully Homomorphic Encryption

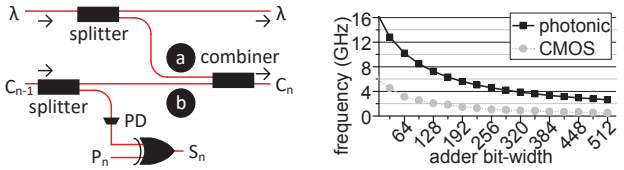
Fully Homomorphic Encryption. FHE [7] allows computations to directly occur on ciphertexts. To guarantee security, FHE introduces *noise* into ciphertexts. An FHE operation enlarges the noise in the ciphertext which can accommodate only a limited number of FHE operations (a.k.a., multiplicative depth) without a decryption failure. *Bootstrapping*s keep the noise in check without a security key. But a bootstrapping is slow, e.g., several minutes.

CKKS. CKKS [18] is one of the widely used FHE schemes in encrypted machine learning [19], since it supports complex arithmetics. CKKS encrypts a vector of complex numbers as a polynomial, whose coefficients are integers modulo Q and polynomial degree is N. For a given N, a ciphertext can encrypt up to $N/2$ complex numbers for SIMD operations, each of which is an element-wise multiplication or addition between two ciphertexts. Its critical operations include:

- **FMUL**: An FHE multiplication (FMUL) is implemented using polynomial multiplications and additions.
- **FROT**: An FHE rotation (FROT) rotates the vector encrypted in a ciphertext. ROT performs an automorphism on the ciphertext.
- **FBOT**: An FHE bootstrapping (FBOT) reduces noises in a ciphertext, and refreshes its consumed multiplicative depth.

Key-Switching. Key-Switching (KS) [7] makes the resulting ciphertext of an FMUL, FROT, and FBOT stay encrypted by the same secret key as the input ones. *KS is the most expensive and dominate kernel in these operations*, i.e., it takes **>90%** [12], [13] of all operations. KS expands the input polynomial to use wider coefficients, multiplies the polynomial by KS hint matrices [12], and then converts the resulting polynomial back to use original coefficients. By using different KS hint matrices, KS can adjust the trade-off between the noise cost and the time complexity of an FHE operation.

Number Theoretic Transform. Multiplying 2 polynomials of degree-N has the time complexity of $\mathcal{O}(N^2)$. NTT [20], a variant of FFT for modular arithmetic, reduces the polynomial multiplication time complexity to $\mathcal{O}(N \log N)$. Specifically, $NTT(\mathbf{a} \cdot \mathbf{b}) = NTT(\mathbf{a}) \bigodot NTT(\mathbf{b})$, where $\mathbf{a}$ and $\mathbf{b}$ are two



(a) A n-bit Electro-Optical full adder (PD: photo-detector). (b) The frequency comparison between CMOS & photonic design.

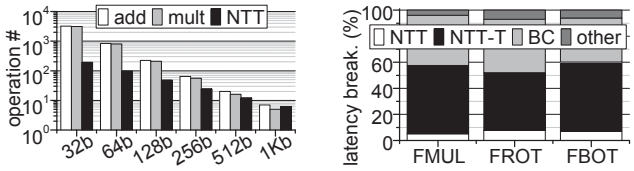
Fig. 1: (a) A photonic ripple-carry adder deign illustration; (b) the comparison of adder frequency between CMOS and photonic design.

polynomials of degree-N, and $\bigodot$ is element-wise multiplication. Typically, a large degree-N ($N = n^2$) polynomial is placed as an $n \times n$ matrix. The NTT operation on the large polynomial can be performed in four steps [12]. First, n n-element NTT operations, each of which occurs on one row of the matrix, can be done. Second, the matrix is multiplied with some constants. Third, the matrix is transposed. Finally n n-element NTT operations, each of which happens on one row of the matrix, are conducted. *Matrix transpositions greatly prolong the latency of an NTT by introducing huge volumes of memory traffic* [12].

Residue Number System. RNS [20] denotes an integer by a set of residues modulo predefined pairwise co-prime moduli. Through RNS, a CKKS operation working with a large Q is converted to a set of computations on Q/W residues, where W is the datapath bit-width of the FHE accelerator. For instance, a 560-bit Q is decomposed to 10 60-bit residues on a CPU, 11 54-bit residues on a FPGA [20], or 20 28-bit residues on a GPU [21] and ASIC [12]. However, for a given Q, *smaller bit-width residues significantly increase the computational overhead of KS*, i.e., the number of NTTs, multiplications, and additions in each KS, in FHE operations.

B. Electro-optical Ripple-Carry Adders

Optical microdisk [22] emerges as one of the most promising technologies for photonic computing, due to its CMOS compatibility, low power consumption, and ultra-fast speed. A 1-bit microdisk-based electro-optical (EO) full adders (FAs) [22] is shown in Figure 1(a). Via n FAs, an n-bit EO ripple-carry adder can be built. For the adder, its sum is calculated as $S_n = C_{n-1} \oplus (A_n \oplus B_n) = C_{n-1} \oplus P_n$, while its carry is computed as $C_n = (A_n \oplus B_n) \cdot C_{n-1} + A_n \cdot B_n = P_n \cdot C_{n-1} + G_n$, where P_n is the propagate bit, and G_n is the generate bit. Since the carry computation is on the critical path of a ripple-carry adder, CMOS gates are used to compute P_n, G_n, and S_n, but microdisks are adopted to compute the carry. The optical carry signal C_{n-1} is separated into two halves by a splitter. One half of C_{n-1} is sent to a photo-detector and converted to an electrical signal, which goes through an XOR gate together with P_n to compute the sum S_n. The other half of C_{n-1} goes through a microdisk b controlled by P_n to produce $P_n \cdot C_{n-1}$. A combiner merges $P_n \cdot C_{n-1}$ and G_n generated by another microdisk a. The



(a) Operation # v.s. datapath width in various FHE operations.

(b) Latency breakdown in various basic FHE operations (NTT-T: matrix transposition in NTT operations; BC: base conversion).

Fig. 2: The bottlenecks of prior FHE accelerators.

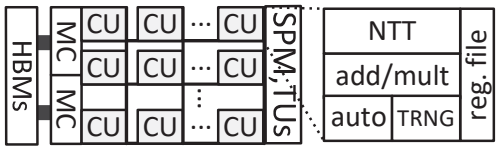


Fig. 3: The architecture of PriML (MC: memory controller; CU: computing unit; SPM: scratchpad memory; TU: transpose unit; auto: automorphism unit; and TRNG: true random number generation).

continuous wave λ is tuned to ensure the balance of light intensities at the two arms of combiners. When the bit-width of a ripple-carry adder increases, as Figure 1(b) shows, the adder frequency significantly decreases [23]. Although a carry look-ahead adder can maintain high frequency for large bit-widths, the carry look-ahead logic components consume nontrivial power. Compared to the electrical counterpart, the n-bit EO ripple-carry adder greatly enhances its frequency without introducing large power overhead. We build a wide datapath operating at high frequency by n-bit EO ripple-carry adders in this paper.

C. Related Work and Motivation

ASIC-based FHE accelerators, CraterLake [12] and BTS [13], obtain the state-of-the-art performance. However, *their performance is seriously limited by their narrow datapaths and intensive matrix transpositions*. First, the narrow datapaths of prior FHE accelerators greatly increase the number of additions (adds), multiplications (mults), and NTTs in a key-switching (KS), which is the dominate primitive in FMULs, FROTs, and FBOTs. CraterLake [12] has a 32-bit modular integer datapath, while BTS [13] supports 64-bit modular arithmetics. To compute a KS with a 1024-bit ciphertext modulus Q, 32-bit CraterLake performs 3.2K adds, 3.1K mults, and 192 NTTs, while 64-bit BTS computes 832 adds, 800 mults, and 96 NTTs, as shown in Figure 2(a). A narrower datapath exponentially increases the number of adds and mults, and linearly enlarges the number of NTTs in a KS. In contrast, a wider datapath may greatly decrease the number of adds, mults, and NTTs in each KS. However, a wider datapath (e.g., 512-bit) also exponentially increases the power consumption and the chip area of an FHE accelerator. Second, frequent matrix transpositions greatly prolong the latency of an NTT on prior FHE accelerators by introducing huge volumes of memory traffic. To transpose a large (e.g., 256×256) matrix, all processing elements of prior FHE accelerators [12], [13] have to frequently access scratchpad memory arrays, and thus cannot focus on the NTT computations. As a result, the matrix transpositions (NTT-T) during NTTs in various FHE operations become the largest bottleneck, as highlighted in Figure 2(b), while the computations of NTTs (NTT) consume only $< 10\%$ of the FHE operation latency.

III. PriML

A. The Architecture of PriML

We propose an EO FHE accelerator, PriML, to process FHE operations. The architecture of PriML is shown in Figure 3. PriML has 2048 512-bit computing units (CUs). All CUs are connected to a 512-bank 512MB eDRAM scratchpad memory (SPM) system by optical crossbar NoCs. In each SPM bank, there are multiple transpose units (TUs) to transpose matrices for (i)NTTs without sending the data to CUs via NoCs. PriML has two memory controllers (MCs) and two HBM2 PHYs to communicate with the off-chip memory.

B. A 512-bit EO CU

We build a 512-bit EO CU featured by an NTT unit, a modular add/mult unit, an automorphism unit, and a TRNG unit. Its most important component is the 512-bit EO NTT unit, which has an arithmetic and inversion unit, an address generation unit, and two butterfly units. A 512-bit EO NTT unit also supports the kernel of iNTT working in a different data flow. We also use EO adders and multipliers to construct the other CU components.

An EO NTT Unit. We present an EO NTT unit for the CU. Matrix transpositions are done by TUs in SPM banks, but the other three steps of the NTT on a large polynomial are computed by an NTT unit. PriML aims to support 64K-element NTTs, so a NTT unit supports 256-element NTT operations. The details are summarized as follows.

- **Data flow**: Like prior FHE accelerators [12], [13], we adopt the Cooley-Tukey data flow [20] for (i)NTTs.
- **A butterfly unit**: We propose an EO butterfly unit (BU) to accelerate radix-2 NTT butterflies, as shown in Figure 4(a). A BU consists of an EO pipelined integer array multiplier, an EO Montgomery modular reduction unit, and two EO modular adders. The EO multiplier computes the multiplication between the input and a twiddle factor ω. The EO Montgomery modular reduction unit performs modular reduction on the multiplication result. By an EO adder and a comparator, the EO modular adder performs modular additions and subtractions. Two EO modular adders can generate the radix-2 butterfly outputs concurrently.
- **A pipelined EO array multiplier**: Through EO ripple-carry adders [22], we propose an EO pipelined integer array multiplier. We show the example of a 4-bit $\times$ 4-bit pipelined array multiplier in Figure 4(b). An m-bit $\times$ m-bit pipelined

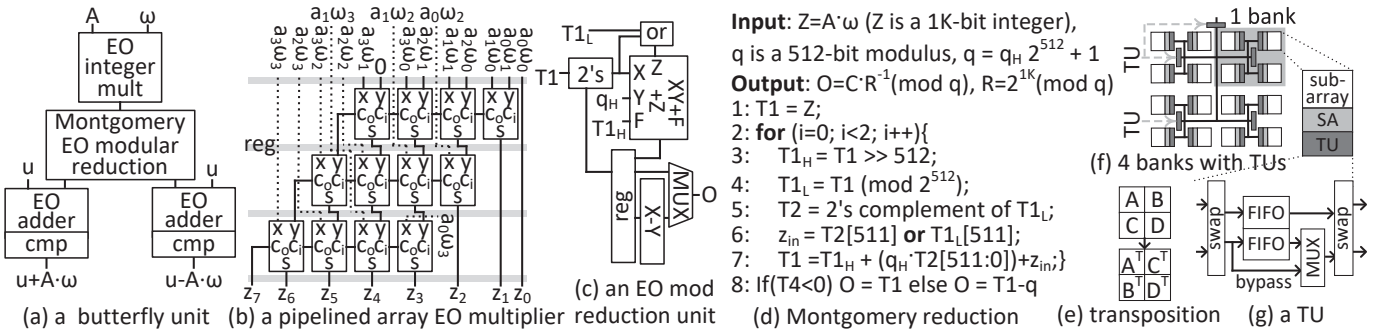


Fig. 4: An NTT unit in an Electro-Optical computation unit, and a transpose unit (TU) in a SPM.

array multiplier consists of m stages, each of which is an m-bit EO ripple-carry adder. Between two stages, there is an m-bit register file to buffer the intermediate result. The inputs of the first pipeline stage of the multiplier are the results of AND operations between the corresponding bits of the multiplier inputs. The input of the other multiplier pipeline stage is the shifted output of the previous multiplier pipeline stage. Although an m-bit $\times$ m-bit multiplication takes m cycles, *the pipelined array multiplier generates an output per cycle*. The wide datapaths and in-SPM TUs of PriML make the (i)NTT kernels memory-friendly, so the (i)NTT kernels become more computationally intensive. Since each stage, i.e., a 512-bit EO ripple-carry adder, can operate at 3GHz [22], the pipeline of a 512-bit $\times$ 512-bit EO array multiplier also works at 3GHz. Since only the EO devices on its critical path work at such high frequency, based on our estimation, the power consumption of the 512-bit $\times$ 512-bit EO array multiplier is similar to that of its CMOS counterpart operating at $\sim$800MHz.

- **A Montgomery modular reduction unit**: As Figure 4(c) shows, we build an EO Montgomery modular reduction unit (MMRU) in a BU to perform modular reduction operations. The MMRU implements the modular reduction algorithm [20] shown in Figure 4(d). Besides some logic operations, and 2's complement conversions, the most intensive operation in a modular reduction operation is the multiply-add operation (i.e., $T1_H + (q_H \cdot T2) + z_{in}$), which can be computed by EO adders and an EO multiplier. The output of each iteration of the loop can be cached in a register file and used as the input for the next iteration. A MUX selects one between the outputs of the register file and an EO adder as the modular reduction output.

A modular add & mult unit. In a CU, we group two modular adders and two modular multipliers to construct a modular add/mult unit. A modular adder consists of an EO 512-bit integer adder and a comparator, while a modular multiplier is composed of an EO 512-bit $\times$ 512-bit integer pipelined array multiplier and a Montgomery modular reduction unit.

An automorphism unit. In an automorphism unit, we use 128 CMOS 32-bit $\times$ 32-bit integer multipliers, 128 binary shifters, and several binary logic gates to compute the index permutations. But matrix transpositions are done by TUs.

A TRNG unit. Since key generation and encryption rely on true random number generation (TRNG), in a CU, we adopt a CMOS TRNG generator [24], which combines the the entropy of multiple independent sources to generate a TRNG bit-stream. It produces $162.5M$ random bits per second and consumes only $\sim 1.09mW$.

C. An eDRAM-based SPM System with TUs

Scratchpad Memory. To avoid the large power of SRAM [25], we present a 512MB eDRAM-based scratchpad memory (SPM) system consisting of 512 banks, which is large enough to store ciphertexts and KS hint matrices shared by FHE operations. To reduce the refresh power of the eDRAM SPM, we skip refreshes on the rows which are accessed by a TU or a CU in each refresh period. All CUs access SPM banks via two optical crossbar NoCs.

Transposing a Matrix. Matrix transpositions are heavily used in the (i)NTT and automorphism kernels of FHE operations. As Figure 4(e) shows, we implement the recursive algorithm [12] to transpose a large matrix. An $E \times E$ matrix is stored in the eDRAM in a row-major order, i.e., the element$[n, m]$ $(0 \leq n, m \leq E-1)$ is stored at the address of $E \times n + m$. After the transposition, the address of the element is $E \times m + n$. For the transposition of a large $E \times E$ matrix, we divide the matrix into four $\frac{E}{2} \times \frac{E}{2}$ matrices, i.e., A, B, C, and D, at the top level. Instead of transposing the matrix directly, we compute A^T, C^T, B^T, and D^T. By repeating this process recursively, the $E \times E$ matrix can be transposed.

In-SPM Transpose Unit. We create an in-SPM transpose unit (TU) to transpose a matrix inside SPM banks without sending it to CUs via NoCs. As Figure 4(f) shows, along the H-tree of all SPM banks, we hierarchically deploy TUs in all SPM banks. We assume a SPM bank has four sub-arrays. Each sub-array in a SPM bank has a TU. One bank has a level-2 TU to enable data movements inside the bank. All banks share a level-3 TU for inter-bank communication. The level-3 and level-2 TUs perform the recursive matrix transposition algorithm. When the recursive process reaches the 2×2 matrix level, a TU attached to a sub-array swaps each element into its new position. The index of an element in the 2×2 matrix is $0 \leq i \leq 4$. The new index (j) after the transposition can be computed by as: if $i = 3$, $j = 3$; otherwise, $j = (2i \mod 3)$.

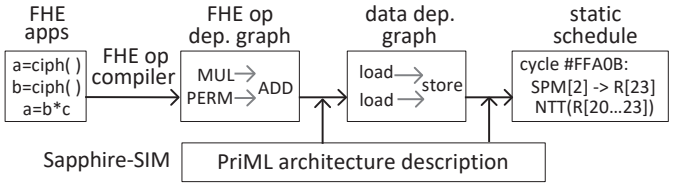


Fig. 5: Compilation on PriML.

A TUs share a similar structure shown in Figure 4(g) with different numbers of FIFOs.

D. Compiling FHE Operations on PriML

We build a compilation framework to compile C/C++ source code to executable binaries for PriML. Our compilation framework is shown in Figure 5.

- First, a C/C++-based application can be translated to FHE primitives, e.g., FMUL, FROT and FBOT, by an FHE operation compiler Cingulata [26]. The compiler reorders FHE operations to reuse each KS hint as much as possible to minimize the DRAM traffic.
- Second, we modified the state-of-the-art FHE accelerator simulator, Sapphire-Sim [27], by adding an Operation-to-Function-Unit (OtFU) mapper. The OtFU mapper decomposes each FHE operation to function kernels, e.g., (i)NTT and SIMD add/mult, converts these function kernels to a data dependency graph, and then maps the graph to function units of PriML (specified by the architecture description). Each function unit can be an NTT unit, a modular add/mult unit, an auto unit, a TRNG unit, or a TU. To simplify the mapping, the mapper considers the SPM and DRAM as function units, so the graph includes accesses to on-/off-chip memories. The mapper statically schedules every node of the graph to a function unit to maximize the operation concurrency and the utilization of each function unit using the worst-case latency of function units and DRAM.
- Third, we modified the cycle-level scheduler of Sapphire-Sim to further refine the data dependency graph. The scheduler divides a function kernel into multiple sub-kernels, schedules sub-kernels across all components of a function unit, and allocates data in the register files of the function unit to maximize the data locality. This step determines the exact cycles of all sub-kernels, and produces the binaries all function units. The scheduler also minimizes the off-chip data movement in multiple passes.

E. Design Overhead

The power and area overhead of PriML is shown in Table I. All CMOS logic units are synthesized by Synopsys design compiler with 7nm PTM process. The eDRAM SPM and register files are modeled by CACTI. To simulate photonic microdisk-based computing components, we used Lumerical FDTD [28] and INTERCONNECT. To model the electro-optical full adder, we adopted optical splitters & combiners, photo-detectors, and microdisks from [22]. We set the frequency of electro-optical adders and multipliers to 3GHz. We use two 32×16 optical crossbars to connect all CUs and SPM banks. We used two HBM2 PHYs to access off-chip DRAMs. Totally, PriML occupies $290.1mm^2$ and consumes $146.2W$.

TABLE I: The power and area of PriML (7nm).

Name	Component	Spec	Power	Area
CU	(i)NTT	$\times$2 BUs	$7.98mW$	$28{,}503\mu m^2$
	Add/Mult	$\times$2 mod adds/mults	$5.94mW$	$21{,}852\mu m^2$
	auto	$\times$128 mults/shifts	$2.92mW$	$14{,}459\mu m^2$
	TRNG	$\times$1	$1.09mW$	$1{,}501\mu m^2$
	reg. files	512KB	$2.15mW$	$12{,}479\mu m^2$
Sub-Total			$20.09mW$	$78{,}794\mu m^2$
PriML	CU	$\times$2048	$161.4W$	$41.14mm^2$
	SPM	512MB, 512-bank	$8.1W$	$58.7mm^2$
	optical NoC	2 32×16 bit-sliced	$28.35W$	$16.2mm^2$
	HBM2 PHY	$\times$2	$29.6W$	$31.76mm^2$
	TU	$\times$ 2,569	$24.2W$	$36.8mm^2$
Total			$146.2W$	$290.1mm^2$

TABLE II: Our accelerator baselines (norm. to 7nm).

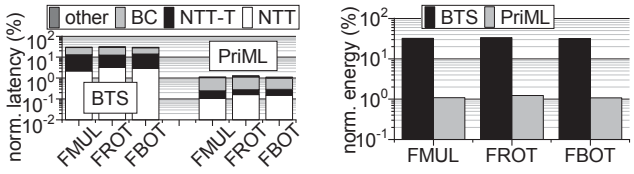
Name	Bit	Description	SPM	Area	Power
Lake [12]	32	1GHz, 1.4K PEs, 6 HBMs	256MB	$276mm^2$	$151W$
BTS [13]	64	1.2GHz, 2K PEs, 2 HBMs	512MB	$373mm^2$	$163W$

IV. Experimental Methodology

Simulation. We modeled PriML by a cycle-accurate FHE accelerator simulator, Sapphire-Sim [27], which is validated against several crypto-processor chips. The input of Sapphire-Sim is the FHE operation schedule generated by our FHE compilation framework. Based on the architectural description of PriML, Sapphire-Sim simulates the cycle-level execution and data movement for each FHE operation. At last, Sapphire-Sim generates the total latency and energy consumption of an FHE application.

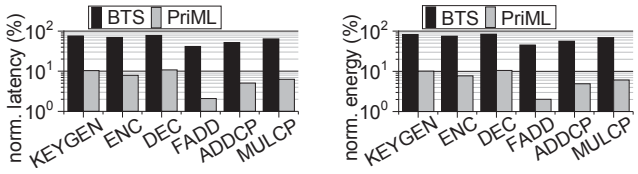
Schemes. We compared PriML against the state-of-the-art ASIC-based hardware accelerators, CrateLake (Lake) [12] and BTS [13]. Their configurations are shown in Table II. Lake has a 32-bit datapath operating at 1GHz and consisting of 1.4K NTT processing elements (PEs), six HBM memory controllers, and a 256MB scratchpad memory (SPM) system. It occupies $276mm^2$ at 7nm and consumes 151 Watt. In contrast, BTS is a 64-bit 1.2GHz FHE accelerator composed of 2K NTT PEs, two HBM memory controllers, and a 512MB SPM system. Its chip size is $373mm^2$ at 7nm, and it consumes 163 Watt. Lake uses a larger off-chip DRAM bandwidth to make up for its smaller SPM capacity.

PriML Benchmarks and Parameters. We first studied the performance and energy of CKKS FMUL, FROT, and FBOT operations on ciphertexts. We also investigated the performance and energy of other CKKS operations including public/private/permutation key generation (KEYGEN), encryption/decryption (ENC/DEC), addition between two ciphertexts (FADD), addition between a ciphertext and a plaintext (ADDCP), and multiplication between a ciphertext and a plaintext (MULTCP). To systematically study FHE performance, we selected two state-of-the-art CKKS-based applications, i.e.,



(a) Latency. (b) Energy.

Fig. 6: The latency and energy comparison of FMUL, FROT, and FBOT between three accelerators (norm. to Lake).



(a) Latency. (b) Energy.

Fig. 7: The latency and energy comparison of other CKKS operations between three accelerators (norm. to Lake).

encrypted logistic regression [19] (LR) and encrypted neural network inference [29] (Lola).

- **LR**. LR is trained with 2K features. It has 4K samples per batch. Nonlinear operations in LR are approximated by polynomials. And LR heavily depends on FMUL, FROT, and FBOT.
- **Lola**. LoLa is a six-layer convolutional network inferring on the CIFAR-10 dataset. The weights of Lola is not encrypted, and all ReLU activations are approximated by square functions. So the most intensive operations of Lola are FROT, MULTCP, and FMUL.

We evaluated the FHE parameters that can support the multiplicative depth of 40 and maintain the 128-bit security, i.e., $N = 2^{16}$, and Q is 1536-bit, since this multiplicative depth is large enough for both FHE applications. Under this set of FHE parameters, a ciphertext costs 24MB.

V. Evaluation

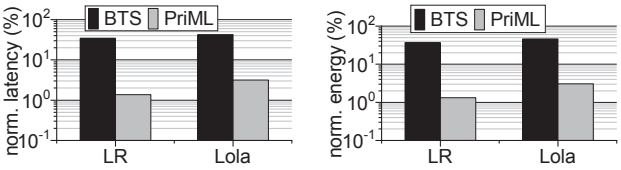
In this section, we first report the CKKS operation performance and energy consumption of PriML, and then discuss the FHE application performance and power efficiency of PriML. At last, we performed the design space exploration for PriML.

A. FHE Operations

FMUL/FROT/FBOT latency. The latency comparison between PriML and various accelerator baselines is shown in Figure 6(a). Compared to Lake, BTS decreases the latency of FMUL, FROT and FBOT by 69% on average, due to its larger bit-width datapath and larger SPM system. The 32-bit datapath of Lake greatly increases the computational overhead of each key-switching (KS) primitive in FMUL, FROT and FBOT operations, while the small SPM cannot hold ciphertexts and KS hint matrices simultaneously, thereby significantly increasing off-chip memory traffic. Because of the 512-bit EO datapath and the TUs in the SPM, PriML reduces the latency of FMUL, FROT and FBOT by 96% on average over BTS. Particularly, the in-SPM TUs of PriML minimize the matrix transpositions (NTT-T) during (i)NTTs, while the 512-bit datapath of PriML evenly reduces the latency of the other kernels in each KS primitive.

FMUL/FROT/FBOT energy. The energy comparison between PriML and various accelerator baselines is shown in Figure 6(b). Compared to Lake, BTS decreases the energy consumption of FMUL, FROT and FBOT by 67.5% on average, although it consumes slightly larger power than Lake, since it uses a much smaller latency to compute the same operation. This shows that building a 64-bit datapath and a 512MB SPM actually improves the energy efficiency for FHE accelerators. Compared to BTS, PriML reduces the energy consumption of FMUL, FROT and FBOT by 98.8% on average. Instead of power hungry SRAM, PriML uses low-power eDRAM to construct its SPM. Although operating at higher frequency, the photonic 512-bit datapath of PriML consumes similar power consumption to the CMOS 64-bit datapath of BTS. In this way, PriML uses only 89% of the power consumption of BTS to complete the same FHE operation.

Other FHE operations. We also studied the latency and energy improvement of other FHE operations including KEYGEN, ENC, DEC, FADD, ADDCP, and MULTCP on PriML in Figure 7. Since these FHE operations do not involve KS, the latency improvement of these FHE operations achieved by PriML is not as significant as those of FMUL, FROT and FBOT. On average, PriML reduces the latency of these FHE operations by 88% over BTS. ENC, ADDCP, and MULCP still invoke (i)NTT kernels, so their latency improvement is still high, as shown in Figure 7(a). Moreover, the 512-bit photonic datapath also helps the FADD operation performing a modular addition between two polynomials. Again, because other FHE operations has no KS primitive, compared to BTS, as Figure 7(b) highlights, PriML reduces the energy consumption of these FHE operations by 91% on average.



(a) Latency. (b) Energy.

Fig. 8: The latency and energy comparison of FHE applications between three accelerators (norm. to Lake).

B. FHE-based PriML Performance

Latency. The latency comparison of FHE applications between PriML and various accelerator baselines is shown in Figure 8(a). Since the performance bottleneck in both LR and Lola is FROT, their latency improvement is heavily influenced by the speedup of FROT. Compared to Lake, BTS reduces the latency of LR and Lola by 65.5% and 57.4% respectively, due to its faster FROT operations. PriML reduces the latency of LR and Lola by 96% and 93% respectively, compared to

BTS. Particularly, the latency improvement of PriML on Lola is smaller. This is because only square activation functions on Lola require FMULs, while the summed products between weights and inputs involve just MULTCPs.

Energy. The energy comparison of FHE applications between PriML and various accelerator baselines is shown in Figure 8(b). The energy reduction of these two applications share a similar trend to the latency reduction. Compared to Lake, BTS decreases the energy consumption of LR and Lola by 62.7% and 54% respectively. PriML reduces the energy consumption of LR and Lola by 96.4% and 93.3% respectively over BTS.

C. Design Space Exploration

We tried 256-, 512-, and 1K-bit datapaths on our PriML. Compared to 256-bit, the 512-bit PriML reduces the latency of FMUL, FROT and FBOT by 39% on average. However, when the datapath reaches 1K, the latency of these operations increases by 3%. This is because a longer datapath decreases the operating frequency of PriML and imposes a larger burden on the small register file in each CU. We also tried 256MB, 512MB, and 1GB SPMs. Compared to 256MB, a 512MB SPM reduces the latency of FMUL, FROT and FBOT by 27%. However, a 1GB SPM does not reduce their latency obviously.

VI. Conclusion

We propose an EO privacy-preserving machine learning accelerator, PriML, which has fast EO 512-bit CUs, a 512MB eDRAM SPM, and in-SPM TUs. PriML greatly reduces the key-switch cost, while the TUs process matrix transpositions in SPM banks. On average, for various machine learning applications, PriML reduces their latency by $> 94.4\%$ and their energy consumption by $> 95\%$ over prior FHE accelerators.

Acknowledgment

This work was supported in part by NSF awards CCF-1908992, CCF-1909509, and CCF-2105972.

References

[1] H. Tuinhof, C. Pirker, and M. Haltmeier, "Image-based fashion product recommendation with deep learning," in *International conference on machine learning, optimization, and data science*. Springer, 2018, pp. 472–481.

[2] G. Koch, R. Zemel, R. Salakhutdinov *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2. Lille, 2015, p. 0.

[3] D. Dong, H. Wu, W. He, D. Yu, and H. Wang, "Multi-task learning for multiple language translation," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2015, pp. 1723–1732.

[4] M. Neethu and R. Rajasree, "Sentiment analysis in twitter using machine learning techniques," in *2013 fourth international conference on computing, communications and networking technologies (ICCCNT)*. IEEE, 2013, pp. 1–5.

[5] C. J. Hoofnagle, B. van der Sloot, and F. Z. Borgesius, "The European Union General Data Protection Regulation: What It Is and What It Means," *Information & Communications Technology Law*, vol. 28, no. 1, 2019.

[6] Q. Lou and L. Jiang, "She: A fast and accurate deep neural network for encrypted data," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[7] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the aes circuit," in *Annual Cryptology Conference*. Springer, 2012, pp. 850–867.

[8] Q. Lou, W.-j. Lu, C. Hong, and L. Jiang, "Falcon: fast spectral inference on encrypted data," *Advances in Neural Information Processing Systems*, vol. 33, pp. 2364–2374, 2020.

[9] B. Feng, Q. Lou, L. Jiang, and G. Fox, "Cryptogru: Low latency privacy-preserving text analysis with gru," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

[10] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer, "Varys: Protecting SGX enclaves from practical Side-Channel attacks," in *USENIX Annual Technical Conference*, Jul. 2018, pp. 227–240.

[11] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "DELPHI: A cryptographic inference service for neural networks," in *USENIX Security Symposium*, 2020.

[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 *IEEE/ACM International Symposium on Computer Architecture*, 2022.

[13] 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 *ACM International Symposium on Computer Architecture*, 2022.

[14] Q. Lou, Y. Shen, H. Jin, and L. Jiang, "Safenet: A secure, accurate and fast neural network inference," in *International Conference on Learning Representations*, 2020.

[15] M. Zheng, L. Ju, and L. Jiang, "Cofhe: Software and hardware co-design for fhe-based machine learning as a service," *Frontiers in Electronics*, vol. 3, 2023.

[16] M. Han, Y. Zhu, Q. Lou, Z. Zhou, S. Guo, and L. Ju, "coxhe: A software-hardware co-design framework for fpga acceleration of homomorphic computation," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 1353–1358.

[17] L. Jiang, Q. Lou, and N. Joshi, "Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 235–240.

[18] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

[19] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang, "Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation," *JMIR Medical Informatics*, vol. 6, no. 2, p. e19, Apr 2018.

[20] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: An Architecture for Computing on Encrypted Data," in *Architectural Support for Programming Languages and Operating Systems*, 2020.

[21] Palisade, "PALISADE Operations using CUDA," 2020.

[22] Z. Ying, S. Dhar, Z. Zhao, C. Feng, R. Mital, C.-J. Chung, D. Z. Pan, R. A. Soref, and R. T. Chen, "Electro-Optic Ripple-Carry Adder in Integrated Silicon Photonics for Optical Computing," *Journal of Selected Topics in Quantum Electronics*, 2018.

[23] F. de Dinechin, H. D. Nguyen, and B. Pasca, "Pipelined FPGA Adders," in *International Conference on Field-Programmable Logic and Applications*, 2010.

[24] S. K. Mathew, D. Johnston, S. Satpathy, V. Suresh, P. Newman, M. A. Anders, H. Kaul, A. Agarwal, S. K. Hsu, G. Chen, and R. K. Krishnamurthy, "μRNG: A 300-950 mV, 323 Gbps/W All-Digital Full-Entropy True Random Number Generator in 14nm FinFET CMOS," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 7, pp. 1695–1704, 2016.

[25] L. Jiang, B. Zhao, Y. Zhang, and J. Yang, "Constructing large and fast multi-level cell stt-mram based cache for embedded processors," in *ACM/IEEE Design Automation Conference*, 2012.

[26] S. Carpov, P. Dubrulle, and R. Sirdey, "Armadillo: A compilation chain for privacy preserving applications," in *3rd International Workshop on Security in Cloud Computing*, 2015.

[27] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, no. 4, p. 17–61, Aug. 2019.

[28] Lumerical, "Fdtd solutions," http://www.lumerical.com/tcad-products/.

[29] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low Latency Privacy Preserving Inference," in *International Conference on Machine Learning*, 2019, pp. 812–821.