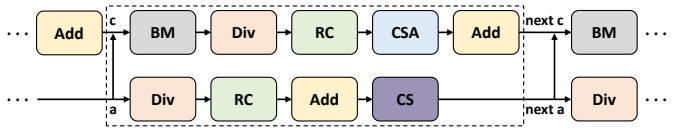

Fig. 6. Pipelined modular multiplication.

removed. On the other hand, the conversions can be optimized with the packing strategy proposed in [22]. The plaintext space (3072 bits) is too large for the values in the models (within 64 bits), and decryption in Paillier costs much more than encryption. Thus the space can be divided into small-scale buckets, with a smaller value in each bucket. Computation results in each bucket will not disturb others if the bucket size is large enough to ensure that there is no overflow in the following computation. So different ciphertexts can be packed into one, and there will be fewer ciphertexts needing to be sent and decrypted, which reduces both communication and computation.

The optimal bucket size depends on the computation under encrypted values in different protocols. In hybrid schemes, the optimal bucket size roughly equals $(m+1)l + log(a+1) + \sigma$ to ensure that there is no overflow in each bucket [22], where l is share length, usually equaling 64. m and a represent the numbers of multiplication with plaintext shares and addition with other shares. σ denotes the statistic security parameter of the scheme, equaling 40 in default. For example, AHE handles matrix multiplication in [5], where one multiplication and multiple additions are processed with each ciphertext. And the bucket size can be set to 180 as default, where each ciphertext contains around 17 buckets. 180-bit bucket size keeps valid unless the number of additions in matrix multiplication exceeds 2^{12}. Then the decryption overhead can be reduced at the expense of several ciphertext additions and plaintext multiplications for packing.

IV. Implementation and Evaluation

We implement SHAPER on a Xilinx 16nm VU13P FPGA with Xilinx Vivado toolchain. Several optimizations are applied to our implementation to improve performance and FPGA resource efficiency.

A. Parallel Modular Operations

1) Parallel Modular Operations: We observe that there are a large amount of matrix computations in real-world PPML scenarios [5], [10], consisting of multiple AHE operations without data dependency. Meanwhile, a single Paillier encryption can also benefit from parallelization, as fixed-base pre-computation is involved to improve efficiency. Thus, we provide support for vectorized modular operations in our design.

The pipeline implementation of the MM engine has five stages for each iteration, as shown in Fig. 6. Each stage takes 5 execution cycles. Block Multiplication (BM) stage calculates $b_i a$ in Alg. 1. Division (Div) stage computes the Barrett division in Alg. 2 to obtain γ. Reduction Computation (RC) stage multiplies γ with $-m$. CSA stage involves the carry-save adders to merge $c + b_i a - km$ into single addition, and Add stage includes an optimized ripple-carry adder. CS stage handles the conditional subtraction of a. Pipelining brings 5 times performance improvement to the MM engine. Besides pipeline, we also deploy multiple MM engines on FPGA to improve parallelism.

2) Optimal Pre-computation Window: The ME in Paillier encryption is under fixed bases depending on the public keys. An optimization using this insight is to store all ME of short powers (*e.g.* window) in the offline stage. Large integer ME is converted to multiple MM Operations in the online stage [3]. Enlarging the pre-computation window can reduce the ME latency. Nevertheless, the maximum window size is restricted by on-chip memory size, as a larger window has a larger enumeration space. If the window size is 4, the required memory size for the pre-computed table is around 34 MB. When the window size grows to 8, the size grows to around 287 MB. In this case, SHAPER provides the interface **AHE.init** to reload the pre-computed table with the default window size of 4.

B. FPGA Resource Utilization

Table II shows the resource usage of SHAPER. The maximum clock frequency is 300 MHz under the default setting. As the most expensive module, 10 MM engines are deployed considering performance and LUT consumption. 32 Integer engines are deployed to support vectorized SS operations, contributing little to the overall consumption. Only one CSPRNG is deployed as its throughput is over 2.6 Gbps, which is enough for existing hybrid schemes. 75% of URAM is occupied for the pre-computed table, which is balanced between performance and area.

TABLE II
Resource utilization on the Xilinx FPGA.

Module	LUT	FF	BRAM	URAM	DSP	Num
MM Engine	99446	136795	0	0	624	10
Pre-Computed Table	38912	49152	1024	960	0	1
CSPRNG	3447	3234	29	0	0	1
FIFO	228	1924	45	0	0	1
Integer Engine	311	3234	0	0	12	32
ScratchPad	2595	3637	512	0	0	1
Misc.	1613	3880	290	0	0	1
Total*	61%	45%	71%	75%	54%	-

* Measured by percentage in terms of Xilinx VU13P FPGA.

C. Function-Level Comparisons

We evaluate the latency of general functions, including modular operations, Paillier functions, and several MPC-level functions. The latency of these micro-benchmarks is shown in Table III. We test the performance of SHAPER in different settings. The results indicate that SHAPER has a great advantage in cycles compared with [2]. The MM engine of SHAPER greatly reduces the cycles, which leads to an order-of-magnitude efficiency improvement.

Table III shows the HW speedup of 10xMM SHAPER with 2048-bit key compared with 12xCP (cryptography processor) [2], and SW speedup compared with the solutions published in [5], [8], [16]. The results show that MM and ME latency of SHAPER is reduced by 25 times compared with [2]. Paillier encryption in SHAPER has a significant advantage according to the results. It is reasonable as our encryption benefits a lot from exclusive optimizations including DJN and pre-computation, which makes the speedup increase to more than 10 times higher than the decryption speedup. The applied CRT optimization contributes $2\times$ speedup on decryption, MM and ME. Hardware solutions for higher-level functions are not provided in [2]. So we only compare them with software solutions. In this case, SHAPER performs 9.03-657 times better than the well-optimized software solutions [5].

D. End-to-End PPML Comparisons

Actually, we can hardly find a hardware competitor since most existing accelerators only consider small datasets, which is far less than real-world business-to-business applications. We build a simulator to get the performance of SHAPER when computing the CAESAR

TABLE III
COMPARISON BETWEEN THE HARDWARE PERFORMANCE AND SPEED OF FUNCTION-LEVEL MICRO-BENCHMARKS.

Design	Key Length	FPGA	Freq.(MHz)	Function Latency (μs)									
				MM	ME	Enc	Dec	CCAdd*	PCAdd	PCMult	S2H	H2S	TriGen
MK20 [2](1xCP)	2048	Xilinx Artix-7	122	16.42	50490	180450	180720	54.42	-	-	-	-	-
SHAPER **(1xMM)**	2048	Xilinx Artix-7	130	1.24	3795	632	7590	6.44	6.69	618	-	-	-
SHAPER **(10xMM)**	2048	Xilinx UltraScale+	300	0.054	164	27.4	329	0.28	0.29	26.8	27.7	30.8	139
SHAPER **(10xMM)**	3072	Xilinx UltraScale+	300	0.080	370	61.6	739	0.42	0.43	40.1	62.0	44.8	249
HW Speedup	-	-	-	25.7×	25.7×	549×	45.8×	16.2×	-	-	-	-	-
CPU Speedup	-	-	-	-	-	657×	54.7×	28.7×	27.6×	28.7×	9.03×	19.5×	10.6×

* **CCAdd** - Ciphertext-ciphertext addition; **PCAdd** - Plaintext-ciphertext addition; **PCMult** - Plaintext(64bit)-ciphertext multiplication;
S2H / H2S - Transformation from SS / HE to HE / SS; **TriGen** - Beaver triple generation.

TABLE IV
COMPARISON BETWEEN THE SIMULATED PERFORMANCE OF END-TO-END LOGISTIC REGRESSION TRAINING.

Design	Platform	Primitive	Security (bit)	Latency (min)	Speedup*
SHAPER	FPGA	Hybrid	128	42	**73.8**×
SHAPER	FPGA	Hybrid	112	24	**129**×
CAESAR [5](Paillier)	CPU	Hybrid	112	3101	1×
CAESAR [5](OU)	CPU	Hybrid	112	333	9.31×
CraterLake [23]	ASIC	HE	128	3100	1.00×
SecureML [18]	CPU	SS	128	14729	0.211×

* Take CAESAR (Paillier) as the baseline.

scheme [5], and compare the simulated results with the results on the CPU. Other solutions using pure HE or SS are also introduced for comparison. The result of [23] is estimated based on scaling their results over small sets, as their scheme has computation overhead linear to the size of the dataset. [23], which requires expensive bootstrapping, is designed for non-interactive outsourcing computing scenarios, where communication latency is ignored.

We evaluate real-world PPML applications with practical network settings, *i.e.* 40 Mbps network bandwidth and 40ms latency. Table IV shows the performance on sparse logistic regression with 1M samples, 100K features with 0.02% sparsity. SHAPER performs $129\times$ faster than CAESAR executed on the CPU when both using 2048-bit Paillier (112-bit security). The acceleration is mainly contributed by fast Paillier encryption and pipeline execution. Even when the key is lengthened for 128-bit security (3072-bit key), SHAPER still performs $7.9\times$ better than OU-based CAESAR with 112-bit security. Besides, most solutions of hybrid schemes show significant efficiency gains compared with SS or HE-based solutions, confirming the performance advantage of hybrid PPML schemes. SecureML performs the worst among the solutions since SS-based solutions mask sparse features to dense data shares.

V. CONCLUSION

In this paper, we propose SHAPER to accelerate hybrid SS-AHE PPML protocols. The algorithm-protocol-hardware co-design methodology explores the full-stack techniques to minimize the end-to-end latency in various network settings. SHAPER further supports secure domain computing acceleration and the conversion between mainstream privacy-preserving primitives, making it ready for general and distinctive data characteristics. We provide a prototype of SHAPER on an off-the-shelf FPGA with several hardware optimizations. Our evaluation shows that SHAPER brings significant speed-up over CPU clusters on a large-scale logistic regression training task.

REFERENCES

[1] M. Al-Rubaie *et al.*, "Privacy-preserving machine learning: Threats and solutions," *IEEE S& P*, vol. 17, no. 2, pp. 49–58, 2019.
[2] M. Bahadori *et al.*, "A programmable soc-based accelerator for privacy-enhancing technologies and functional encryption," *IEEE VLSI*, 2020.
[3] E. F. Brickell *et al.*, "Fast exponentiation with precomputation," in *Advances in Cryptology-Eurocrypt' 92*. Springer, 1992, pp. 200–207.
[4] D. Catalano *et al.*, "Paillier's cryptosystem revisited," in *ACM CCS*, 2001, pp. 206–214.
[5] C. Chen *et al.*, "When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control," in *ACM SIGKDD*, 2021, pp. 2652–2662.
[6] V. Costan *et al.*, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.
[7] I. Damgård *et al.*, "Efficient and secure comparison for on-line auctions," in *ACISP*. Springer, 2007, pp. 416–430.
[8] D. Demmler *et al.*, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.
[9] J. Fan *et al.*, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.
[10] W. Fang *et al.*, "Large-scale secure xgb for vertical federated learning," in *CIKM*, 2021, pp. 443–452.
[11] R. Gilad-Bachrach *et al.*, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *ICML*, 2016.
[12] O. Goldreich *et al.*, "How to play any mental game or A completeness theorem for protocols with honest majority," in *STOC*. ACM, 1987.
[13] S. Hardy *et al.*, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *arXiv preprint arXiv:1711.10677*, 2017.
[14] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," in *ACM CCS*, 2020, pp. 1575–1590.
[15] B. Knott *et al.*, "Crypten: Secure multi-party computation meets machine learning," *Advances in Neural Information Processing Systems*, 2021.
[16] Y. Li *et al.*, "Shift-sub modular multiplication algorithm and hardware implementation for RSA cryptography," in *HIS*. Springer, 2021.
[17] P. Mishra *et al.*, "Delphi: A cryptographic inference service for neural networks," in *USENIX*, 2020, pp. 2505–2522.
[18] P. Mohassel *et al.*, "Secureml: A system for scalable privacy-preserving machine learning," in *IEEE S&P*. IEEE, 2017, pp. 19–38.
[19] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.
[20] T. Okamoto *et al.*, "A new public-key cryptosystem as secure as factoring," in *Eurocrypt*. Springer, 1998, pp. 308–318.
[21] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Eurocrypt*. Springer, 1999, pp. 223–238.
[22] P. Pullonen *et al.*, "Actively secure two-party computation: Efficient beaver triple generation," *Instructor*, 2013.
[23] N. Samardzic *et al.*, "Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data," in *ISCA'22*. ACM, 2022.
[24] G. Xin *et al.*, "Vpqc: A domain-specific vector processor for post-quantum cryptography based on risc-v architecture," *IEEE TCAS-I*, 2020.
[25] R. Xu *et al.*, "Privacy-preserving machine learning: Methods, challenges and directions," *arXiv preprint arXiv:2108.04417*, 2021.
[26] A. C. Yao, "Protocols for secure computations (extended abstract)," in *FOCS*. IEEE Computer Society, 1982, pp. 160–164.
[27] X. Zhou *et al.*, "Ppmlac: high performance chipset architecture for secure multi-party computation," in *ISCA*, 2022, pp. 87–101.