ReDCrypt: Real-Time Privacy-Preserving Deep Learning Inference in Clouds Using FPGAs

BITA DARVISH ROUHANI and SIAM UMAR HUSSAIN, University of California San Diego
KRISTIN LAUTER, Microsoft Research
FARINAZ KOUSHANFAR, University of California San Diego

Artificial Intelligence (AI) is increasingly incorporated into the *cloud business* in order to improve the functionality (e.g., accuracy) of the service. The adoption of AI as a cloud service raises serious privacy concerns in applications where the *risk of data leakage* is not acceptable. Examples of such applications include scenarios where clients hold potentially sensitive private information such as medical records, financial data, and/or location. This article proposes ReDCrypt, the first *reconfigurable* hardware-accelerated framework that empowers privacy-preserving inference of deep learning models in cloud servers. ReDCrypt is well-suited for *streaming (a.k.a., real-time AI)* settings where clients need to dynamically analyze their data as it is collected over time without having to queue the samples to meet a certain batch size. Unlike prior work, ReDCrypt neither requires to change how AI models are trained nor relies on two non-colluding servers to perform. The privacy-preserving computation in ReDCrypt is executed using *Yao's Garbled Circuit* (GC) protocol. We break down the deep learning inference task into two phases: (i) privacy-insensitive (local) computation, and (ii) privacy-sensitive (interactive) computation. We devise a high-throughput and power-efficient implementation of GC protocol on FPGA for the privacy-sensitive phase. ReDCrypt's accompanying API provides support for seamless integration of ReDCrypt into any deep learning framework. Proof-of-concept evaluations for different DL applications demonstrate up to 57-fold higher throughput per core compared to the best prior solution with no drop in the accuracy.

CCS Concepts: • **Security and privacy → Management and querying of encrypted data**;

Additional Key Words and Phrases: Secure machine learning, deep learning, data mining, privacy-preserving computation, garbled Circuit

ACM Reference format:
Bita Darvish Rouhani, Siam Umar Hussain, Kristin Lauter, and Farinaz Koushanfar. 2018. ReDCrypt: Real-Time Privacy-Preserving Deep Learning Inference in Clouds Using FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* 11, 3, Article 21 (December 2018), 21 pages.
https://doi.org/10.1145/3242899

This research is supported in parts by an Office of Naval Research (ONR-R17460), NSF Trust Hub (CNS-1649423), National Science Foundation (CNS-1619261), and MURI (FA9550-14-1-0351) grants.
Authors' addresses: B. D. Rouhani, S. U. Hussain, and F. Koushanfar, University of California San Diego, 9500 Gilman Dr, La Jolla, CA 92093; emails: {bdarvish, siamumar, farinaz}@ucsd.edu; K. Lauter, Microsoft Research; email: klauter@microsoft.com.

1936-7406/2018/12-ART21 $15.00
https://doi.org/10.1145/3242899

1 INTRODUCTION

Deep Learning (DL) has provided a paradigm shift in our ability to comprehend raw data by showing superb inference accuracy resembling the learning capability of the human brain [1, 2]. Technology leaders such as Microsoft, Google, IBM, and Facebook are devoting millions of dollars to devise accurate DL models for various artificial intelligence (AI) applications and provide cloud-based learning/prediction services to clients [3, 4]. Examples of such AI applications range from social networks and autonomous transportation to natural language processing and health care.

A complicating factor in the rush to adopt AI as a cloud service is the data and model privacy. On the one hand, AI models are usually trained by allocating significant computational resources to process massive amounts of training data. As such, the trained models are considered an Intellectual Property (IP) of companies, which require confidentiality to preserve the competitive advantage and ensure receiving continuous query requests by the clients. On the other hand, clients do not desire to send their private data (e.g., location or financial input) in plain text to cloud servers due to the risk of information leakage.

To incorporate deep learning into the cloud services, it is highly desired to devise privacy-preserving frameworks in which neither of the involving parties is required to reveal their private information. Several research works have been developed to address privacy-preserving computing for DL networks, e.g., Refs [5] and [6]. The existing solutions, however, either: (i) Rely on the modification of DL layers (such as non-linear activation functions) to efficiently compute the specific cryptographic protocols. For instance, authors in Refs [5] and [6] have suggested the use of polynomial-based Homomorphic encryption to make the client's data oblivious to the server. Their approach requires changing the non-linear activation functions to some polynomial approximation (e.g., square) during training. Such modification, in turn, can reduce the ultimate accuracy of the model and poses a tradeoff between the model accuracy and execution cost of the privacy-preserving protocol. Or (ii) fall in the two-server settings in which data owners distribute their private data among two non-colluding servers to perform a particular DL inference. The two non-colluding server assumption is not ideal as it requires the existence of a trusted third-party, which is not always an option in practical scenarios.

We propose ReDCrypt, the first provably-secure framework for *scalable* inference of DL models in cloud servers using FPGA platforms. ReDCrypt employs Yao's Garbled Circuit (GC) protocol to securely perform DL inference. GC allows any two-party function to be computed without revealing the respective inputs. In GC protocol, the underlying function shall be represented as a Boolean circuit, called a *netlist*. The truth tables of the netlist are encrypted to ensure privacy. In contrast with the prior work, e.g., Refs [5] and [6], the methodology of ReDCrypt neither requires any modification to the existing DL architecture nor relies on two non-colluding servers to ensure privacy. We designed a customized FPGA implementation for efficient execution of GC protocol on cloud servers. Our implementation provides support for the privacy-preserving realization of the computational layers used in the context of deep learning. To the best of our knowledge, no prior hardware acceleration has been reported in the literature for high-throughput and power-efficient inference of deep learning in a privacy-preserving setting.

We demonstrate that in various deep learning networks, the majority of the privacy-sensitive computations boils down to matrix-vector multiplications (Section 4). To achieve the highest efficiency for privacy-preserving DL computation, we devise a customized hardware architecture for this operation via GC protocol. Our custom solution is designed to maximize system throughput by incorporation of both algorithmic and hardware parallelism. Our design explicitly benefits from the precise control in synchronization with the system clock supported by the FPGA platform as opposed to a generic processor.

To leverage ReDCrypt framework, the cloud server that owns the DL model topology and weights needs to reformat the convolution operations into matrix multiplication by inserting a custom reshaping layer that unrolls the patches as discussed in Section 4. This pre-processing is a one-time step before generating the corresponding Boolean circuit (netlist). DL inference workload in ReDCrypt is divided into local privacy-insensitive and interactive privacy-sensitive computations. The input sample and all the intermediate features (activations) of the pertinent neural network are computed in a privacy-sensitive setting. The encoded output layer's activation is then sent to the client. The client *locally* applies a softmax to the final encoded features to obtain the pertinent data label. The Softmax execution is considered a privacy-insensitive local computation.

Given the importance of GC protocol for secure function evaluation, it is not surprising that several GC realizations on FPGA have been reported in literature [7–9]. The existing works, however, are devised for a generic realization of GC protocol and are not particularly tailored for deep learning computations. By carefully designing the control flow of our customized architecture for matrix-vector multiplications through GC, we ensure minimal idle cycle due to dependency issues. In particular, we demonstrate a 57-fold improvement in throughput per core compared to the best prior-art solution. This improvement directly translates to providing support for 57 times more clients using the same number of computational cores.

The architecture of the GC accelerator embraces two recent approaches in the literature: (1) the TinyGarble [10] framework introduced sequential GC, where the same netlist is garbled for multiple rounds with updated encryption keys; and (2) the work in Ref. [11] performed static analysis on the underlying function (given the control path is independent of the data path) to determine the most optimized netlist to garble in every round. In our design, there are two levels of nested loops: one outer loop and multiple inner loops. The outer loop garbles the netlist of the smallest unit of the matrix-vector multiplication operation in every round, analogous to the sequential GC of Ref. [10]. The inner loop breaks the operation of the unit into segments such that in every clock cycle, we can ensure optimal utilization of the implemented encryption units. Unlike the typical GC approach, the underlying netlist is embedded in a finite state machine (FSM) that governs the transfer of the keys between gates. This approach allows us to employ a parallel architecture for the multiplication operation as we can precisely control the garbling operation in every clock cycle and ensure accurate synchronization among the gates being garbled in parallel. Unlike software parallelization, our approach does not incur any synchronization overhead. As a result, we can ensure the minimal idle cycle of the encryption units.

The explicit contributions of this article are:

- —Introducing ReDCrypt, the first reconfigurable and provably-secure framework that simultaneously enables *accurate* and *scalable* privacy-preserving DL execution. ReDCrypt supports secure streaming (real-time) DL computation in cloud servers using efficient FPGA platforms.
- —Designing customized hardware architecture for *GC-optimized* DL computations (e.g., matrix-vector multiplication and non-linearities) to maximize parallelism. Our architecture seamlessly supports scalability in dimension and data bit-width of the matrices.
- —Presenting the first GC architecture with precise gate level control per clock cycle. Instead of conventional netlist-based GC execution, we design our custom hardware accelerator as an FSM that controls the operation and communication among parallel GC cores, ensuring minimal (highest 2) idle cycles.
- —Providing up to 57-fold improvement in garbling operation compared to the state-of-the-art software GC framework. This translates to the capability of the cloud to support 57 times more clients simultaneously.

—Providing proof-of-concept evaluations of various visual, audio, and smart-sensing benchmarks. Our evaluations corroborate ReDCrypt's scalability and practicability for distributed users compared to the prior-art solution.

The rest of this article is organized as follows. In the next section, we provide a brief background on deep learning and the cryptographic protocols leveraged by ReDCrypt framework. In Section 3, the global flow of ReDCrypt is outlined along with the security model. We then describe the details of ReDCrypt architecture and the operation of the various components of the system in Section 4. Next, we delineate the configuration of the parallel cores in the hardware accelerator in Section 5. In Section 6, we present the details of the implementation and report the timing results and resource usage. We provide a number of practical design experiments in Section 7 to benchmark several DL applications. In Section 8, we discuss in detail the earlier prior-art solutions, limitations, and how ReDCrypt overcomes them. Finally, Section 9 concludes the article.

2 PRELIMINARIES

2.1 Deep Learning Networks

Deep learning refers to learning a hierarchical non-linear model that consists of several processing layers stacked on top of one the other. To perform data inference, the raw values of data features are fed into the first layer of the DL network known as the input layer. These raw features are gradually mapped to higher-level abstractions through the intermediate (hidden) layers of the DL model. The acquired data abstractions are then used to predict the label in the last layer of the DL network (a.k.a., the output layer).

Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) are the two main categories of neuron networks widely used in deep learning domain [1]. These two types of neural networks share many architectural similarities: CNNs are composed of additional convolution layers on top of fully-connected networks that form the foundation of DNNs. The use of convolutional layers in a CNN model makes them better-suited for interpreting data measurements with strong local connectivity (e.g., visual data), whereas DNNs pose a more generic architecture to model datasets that may not show a solid local dependency pattern (e.g., audio data).

The state of each neuron (unit) in a DL model is determined in response to the states of the units in the prior layer after applying a non-linear activation function. Commonly used activation functions for hidden layers include logistic sigmoid, Tangent-hyperbolic (Tanh), and Rectified Linear Unit (ReLu). The output layer is an exception for which a Softmax regression is typically used to determine the final inference. Softmax regression (or multinomial logistic regression) is a generalization of logistic regression that maps a $\mathcal{P}$-dimensional vector of arbitrary real values to a $\mathcal{P}$-dimensional vector of real values in the range of $[0, 1)$. The final inference for each input sample can be determined by the output unit that has the largest conditional probability value [1].

2.2 Cryptographic Protocols

In the following, we provide a brief description of the cryptographic protocols used in this article.

2.2.1 Oblivious Transfer. Oblivious Transfer (OT) is a cryptographic protocol that runs between a Sender (S) and a Receiver (R). The receiver R obliviously selects one of the potentially many pieces of information provided by S. Particularly, in a 1-out-of-n OT protocol, the sender S has n messages $(x_1, \ldots, x_n)$ and the receiver R has an index r where $1 \leq r \leq n$. In this setting, R wants to receive x_r among the sender's messages without the sender learning the index r, while the receiver only obtains one of the n possible messages [12].

2.2.2 Garbled Circuit. Yao's garbled circuit protocol [13] is a cryptographic protocol in which two parties, Alice and Bob, jointly compute a function $f(x, y)$ on their inputs while keeping the

inputs fully private. In GC, the function f should be represented as a Boolean circuit with 2-input gates (e.g., XOR, AND, etc.). The input from each party is represented as input wires to the circuit. All gates in the circuit have to be topologically sorted, which creates a list of gates called *netlist*. GC has four different stages: (i) garbling, which is only performed by Alice (a.k.a., Garbler); (ii) data transfer and OT, which involves both parties, Alice and Bob; (iii) evaluating, only performed by Bob (a.k.a., Evaluator); and finally, (iv) merging the results of the first two steps by either of the parties.

(i) Garbling. Alice garbles the circuit by assigning two random k-bit[1] labels to each wire in the circuit corresponding to semantic values one and zero. For instance, for input wire number 5, Alice creates 128-bit random string l_5^0 as a label for semantic value zero and l_5^1 for semantic value one. For each gate, a garbled table is computed. The very first realization of the garbled table required four different rows, each corresponding to one of the four possible combinations of inputs labels. Each row is the encryption of the correct output key using two input labels as the encryption key [14]. As an example, assume wire 5 (w_5) and 6 (w_6) are input to an XOR gate and the output is wire 7 (w_7). Then, the second row of the garbled table, which corresponds to ($w_5 = 0$) and ($w_6 = 1$) is equivalent to $Enc_{(l_5^0, l_6^1)}(l_7^1)$. To decrypt any garbled table, one needs to possess the associated two input labels. Once Garbler creates all garbled tables, the protocol is ready for the second step.

(ii) Transferring Data and OT. In this step, Alice sends all the garbled tables along with the correct labels corresponding to her actual input to Bob. For instance, if the input wire 8 belongs to her and her actual input for that wire is zero, she sends l_8^0 to Bob. In order for Bob to be able to decrypt and evaluate the garbled tables (step 3), he needs the correct labels for his input wires as well. This task is not trivial nor easy. On the one hand, Bob cannot send his actual input to Alice to avoid undermining his input privacy. On the other hand, Alice cannot simply send both input labels to Bob since Bob can then infer more information in step 3. To effectively perform this task, OT protocol is utilized. For each input wire that belongs to Bob, both parties engage in a 1-out-of-2 OT protocol where the selection bit is Bob's input and two messages are two labels from Alice. After all required information is received by Bob, he can start evaluating the garbled circuit.

(iii) Evaluating. To evaluate the garbled circuit, Bob starts from the first garbled table and uses two input labels to decrypt the correct output key. All gates and their associated dependencies are topologically sorted in the netlist. As such, Bob can perform the evaluation one gate at a time until reaching the output wires without any halts in the process. In order to create the actual plain-text output, both the output mapping (owned by Alice) and final output labels (owned by Bob) are required; thereby, one of the parties, say Bob, needs to send his share to the other party (Alice).

(iv) Merging Results. At this point, Alice can easily compute the final results. To do so, she uses the mapping from output labels to the semantic value for each output wire. The protocol can be considered finished after merging the results (as in ReDCrypt) or Alice can also share the final results with Bob.

2.3 Garbled Circuit Optimizations

During the past decade, several optimization methodologies have been suggested in the literature to minimize the overhead of executing GC protocol. In the following, we summarize the most important cryptographic optimizations techniques that we also leverage for deployment of ReDCrypt framework.

Point and Permute: According to this optimization [15], the label of each wire is appended by a select bit, such that the select bits for the two labels of the same wire are inverse of each other.

[1] k is a security parameter; its value is chosen as 128 in recent works.

Even though the select bits are public, the association between select bits and semantic value of the wire is random and private to the garbler. Besides allowing the use of more efficient encryption, it also makes the evaluation simpler since the evaluator can simply decrypt the appropriate row based on the public select bits of the wire labels.

Row-Reduction: As we discussed earlier, the initial garbled table consists of four rows. Authors in Ref. [16] proposed a technique to reduce the number of rows in the garbled table to three. Instead of randomly generating the label for the output wire of a gate, it is computed as a function of the labels of the inputs such that the first row of the garbled table becomes all 0s and no longer needs to be sent to the evaluator. This technique results in 25% reduction in communication.

Free-XOR: Perhaps one of the most important optimizations of GC is Free-XOR [17]. The Free-XOR methodology enables the evaluation of XOR, XNOR, and NOT gates without costly cryptographic encryption. Therefore, it is highly desirable to minimize the number of non-XOR gates in the deployment of the underlying circuit. In this method, Alice generates a random $(k-1)$-bit value R, which is known only to her. For each wire c, she generates the label X_c^0 and sets $X_c^1 = X_c^0 \oplus (R \parallel 1)$.[2] With this convention, the label for the output wire r of an XOR gate with input wires p, q can be simply computed as $X_r = X_p \oplus X_q$. Note that R is appended by 1 to be compatible with the Point and Permute optimization.

Half-Gates: This technique that is proposed in Ref. [18] further reduces the number of rows for AND gates from three to two, resulting in 33% less communication on top of the Row-reduction optimization.

Garbling with Fixed-Key Block Cipher: This methodology [14] introduces an encryption mechanism for garbled tables based on fixed-key block ciphers (e.g., Advanced Encryption Standard (AES)). Many of the modern processors have AES-specific instructions in their Instruction Set Architecture (ISA) which, in turn, makes the garbling and evaluating process significantly faster using the fixed-key cipher.

Sequential Garbled Circuit: For years the GC protocol could have only been used for Combinational circuits (a.k.a., acyclic graphs of gates). Authors in Ref. [10] suggested a new approach that enables garbling/evaluating sequential circuits (cyclic graphs). In their framework, one can garble/evaluate a sequential circuit iteratively for multiple clock cycles.

3 REDCRYPT GLOBAL FLOW

Figure 1 illustrates the global flow of ReDCrypt framework. In ReDCrypt, a cloud server (Alice) holds the DL model parameters trained for a particular inference application, and a delegated client (Bob) owns a data sample for which he wants to securely find the corresponding classified label (a.k.a., inference result). ReDCrypt empowers the cloud server (Alice) to provide a power-efficient and high-throughput concurrent service to multiple clients. The evaluation of the encrypted circuit is then performed offline on the client side. Note that to ensure privacy, the server requires to garble a new encrypted model for each client and/or input sample.

DL models have become a well-established machine learning technique. Commonly employed DL topologies and cost functions are well-known to the public. Indeed, what needs to be kept private from the cloud server's perspective is the DL model parameters that have been tuned for a particular task using massive statistical databases and devoting large amounts of computing resources for several weeks/months. Data owners, on the other hand, tend to leverage off-the-shelf models to figure out the inference label for their private data while keeping their data fully private.

[2] $\parallel$ denotes the concatenation operator.

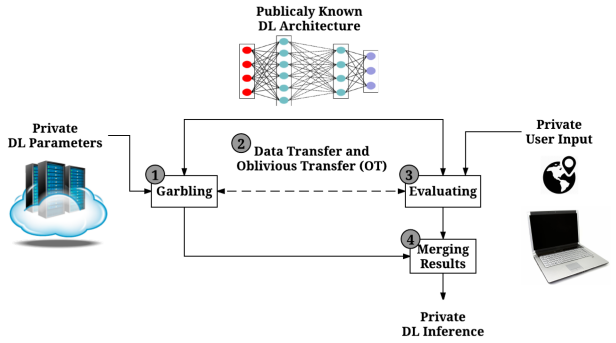

Fig. 1. Global flow of ReDCrypt framework.

ReDCrypt enables computing the pertinent data inference label in a provably-secure setting while keeping both the DL model's parameters and data sample private. To perform a particular data inference in ReDCrypt framework, the netlist of the publicly known DL architecture[3] should be generated prior to the execution of the GC protocol. As shown in Figure 1, the execution of the GC protocol involves four main steps: (i) The server garbles the Boolean circuit of the DL architecture. (ii) The server then sends the computed garbled tables from the first step to the client along with her input wire labels. Both client and the cloud server then engage in a 1-out-of-2 OT [12] protocol to obliviously transfer the wire labels associated with the cloud server's inputs. (iii) The client evaluates (executes) the garbled circuit and computes the corresponding encrypted data inference. (iv) The server shares the mapping for the output labels with the client so that he can learn the inference results.

To provide service to a large number of distributed clients simultaneously, ReDCrypt provides a high-throughput and power-efficient realization of the garbling (step 1) on a field-programmable gate array (FPGA). The garbled tabled are then sent out to the clients through multiple communication channels, which is a rational assumption for cloud servers. The cloud server in ReDCrypt acts as the garbler while the client acts as the evaluator. The motivations behind this setting are as the following:

—The server possesses the deep learning model parameters (the set of weight matrices) and the client holds the input data (a single vector). In GC, the evaluator receives his inputs through OT. Thus, it is more efficient to have the client, who has less private data, as the evaluator. Even though it is possible to send all the inputs at once through OT extension [19], a resource-constrained client may not have enough memory to store all the labels together. With the recent development of sequential GC [10], it is feasible to perform OT every round and store only the labels required for that round, making our approach amenable to a wide range of clients including those who are using resource-constrained embedded systems such as smartphones.

—The garbling operation does not require any input from any party. It is only during the evaluation that the weight matrices and the client data are required. The garbling accelerator keeps generating the garbled tables independently and sends them to the host CPU. The host, in the meantime, dynamically updates her model if required, and, when requested by the client, simply sends one of the stored garbled circuits along with the input labels. Note that even if the model does not change, new labels are required for every garbling operation to ensure the security of the pertinent DL execution.

[3]DL architecture refers to the number and type of layers and not the values of the pertinent private DL parameters.

Table 1. Commonly Used Layers in DNN and CNN Models

DL Layer	Description	Computation	
2D Convolution (*C*)	Multiplying the filter parameters (W_{ij}) with the post-nonlinearity values in the preceding layer ($x_{ij}^{(l-1)}$) and summing up the results	$z_{ij}^{(l)} = \sum_{s_1=1}^{k} \sum_{s_2=1}^{k} W_{s_1 s_2}^{(l-1)} \times x_{(i+s_1)(j+s_2)}^{(l-1)}$	*Weighted Sum*
Max Pooling (M_1P)	Computing the maximum value of $k \times k$ overlapping regions of the preceding layer	$z_{ij}^{(l)} = Max(x_{(i+s_1)(j+s_2)}^{(l-1)})$ $s_1 \in \{1, 2, \ldots, k\}$ $s_2 \in \{1, 2, \ldots, k\}$	*Maximum*
Mean Pooling (M_2P)	Computing the mean value of $k \times k$ non-overlapping regions of the preceding layer	$z_{ij}^{(l)} = Mean(x_{(i+s_1)(j+s_2)}^{(l-1)})$ $s_1 \in \{1, 2, \ldots, k\}$ $s_2 \in \{1, 2, \ldots, k\}$	*Mean*
Fully-Connected (*FC*)	Multiplying the parameters of the l^{th} layer (W_{ij}^{l}) with the post-nonlinearity values in the preceding layer (x_i^{l-1})	$z_i^{(l)} = \sum_{j=1}^{n_{l-1}} W_{ij}^{(l-1)} \times x_j^{(l-1)}$	*Matrix-Vector Multiplication*
Non-Linearity (*NL*)	*Softmax*	$x_i^{(l)} = \frac{e^{z_i^{(l)}}}{\sum_{j=0}^{n_l} e^{z_j^{(l)}}}$	*CORDIC*
	Sigmoid	$x_i^{(l)} = \frac{1}{1+e^{-z_i^{(l)}}}$	*CORDIC*
	Tangent-Hyperbolic (Tanh)	$x_i^{(l)} = \frac{Sinh(z_i^{(l)})}{Cosh(z_i^{(l)})}$	*CORDIC*
	Rectified Linear unit (ReLu)	$x_i^{(l)} = Max(0, z_i^{(l)})$	*Maximum*

3.1 Security Model

We assume an *Honest-but-Curious* (HbC) security model in which the participating parties follow the protocol they agreed on, but they may want to deduce more information from the data at hand. We focus our evaluations on this security model because of the following reasons: (i) HbC is a standard security model in the literature [14, 20] and is the first step toward security against malicious adversaries. Our solution can be modified to support *malicious* models following the methods presented in Refs [21–24]. Note that stronger security models rely on multiple rounds of HbC with varying parameters; thereby, the efficiency of ReDCrypt is carried out to those models as well. (ii) Many privacy-preserving DL execution settings naturally fit well in HbC security, for instance, when all parties have the incentive to produce the correct result (perhaps when the DL inference task is a paid service). In these settings, both data provider and the server that holds the DL model will follow the protocol in order to produce the correct outcome.

ReDCrypt's core secure function evaluation engine is the GC protocol. GC is proven to be secure in the HbC adversary model [25]; thereby, any input from either client or server(s) to GC will be kept private during the protocol execution. Our hardware accelerator does not alter the protocol execution and thus is as secure as any GC realization. In the rest of the article, we will describe in details the ReDCrypt's architecture for enabling efficient privacy-preserving DL execution in cloud servers that support FPGA platforms.

4 SYSTEM ARCHITECTURE

Table 1 summarizes hidden layers commonly used in different DL networks. Each of these layers can be effectively represented as a *Boolean circuit* used in GC. An end-to-end DL model is formed by stacking different layers on top of each other. Note that many of the computations involved in DL inference, such as non-linearity layers, cannot be accurately represented by polynomials used in Homomorphic encryption. For instance, approximating a Rectified Linear unit (ReLu) using Homomorphic encryption [5] requires leveraging high-order polynomials, whereas a ReLu can be accurately represented by a Multiplexer in Boolean circuits.

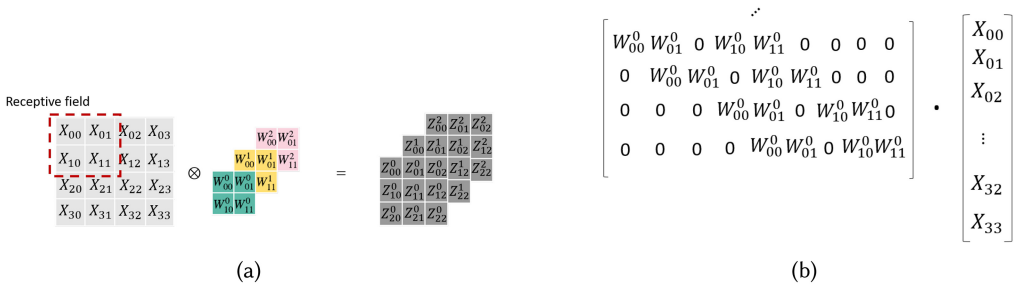

Fig. 2. (a) Convolution operation can be mapped into (b) matrix multiplication. Here, W represents weight kernels, X is the input tensor, and Z indicates the results of convolution. In this figure, the padding and stride are set to 0 and 1 with one channel.

The main computational workload of DL models is related to the realization of convolutional and fully-connected layers [26]. Fully-connected layers are essentially built based on matrix-vector multiplication (see Table 1). The convolutional layers can also be evaluated in the form of a matrix multiplication [27]. Figure 2 demonstrates how a convolution operation can be effectively transformed into a matrix multiplication. As such, it is highly required to design our hardware-accelerated GC engine such that it supports the efficient realization of this operation in a privacy-preserving setting. The non-linearity layers in a neural network, including Tanh, Sigmoid, and ReLu, can be easily implemented as a Boolean circuit using COordinate Rotation DIgital Computer (CORDIC) and the multiplexer (MUX).

DL inference workload in ReDCrypt framework is divided into two phases: (i) interactive privacy-sensitive, and (ii) local privacy-insensitive computations. The interactive privacy-sensitive phase includes computing the intermediate activation sets that rely on both the input data and DL model weights. In the very last layer of a neural network, a Softmax layer is typically used to convert the last layer activations into a probability vector and find the ultimate class. Evaluating the Softmax function in ReDCrypt framework is considered a local privacy-insensitive operation as the client is in charge of evaluating the garbled circuit and should ultimately know his corresponding data label. Nevertheless, we want to emphasize that Softmax is a monotonically increasing function. Therefore, applying this function to a given input vector does not change the index of the maximum value (inference label index). Therefore, one can also use logic building blocks such as MUX and comparator for the realization of the Softmax functionality in a privacy-preserving setting if required by another application.

The architecture of ReDCrypt is presented in Figure 3. The cloud server includes a Central Processing Unit (CPU) as the host and an FPGA-based accelerator to perform the garbling operation. The GC accelerator on FPGA generates the garbled tables along with the labels for both garbler (server) and evaluator (client) and sends them to the host CPU. The CPU stores them in a buffer (not shown in the figure) and reads back when requested for an inference task by a client. Note that ReDCrypt only accelerates the GC computation on the server side and is independent of the GC realization on the client side. Moreover, the operation on the client side is transparent to the presence of ReDCrypt on the server. In general, the bottleneck of GC protocol evaluation is communication as also shown in the prior works [10]. As such, acceleration of GC evaluation on the client side is not effectual to reduce the overall latency. However, in the cloud server setting, where a single server is simultaneously communicating with a large number of clients via multiple channels, generating the garbled tables becomes the bottleneck. Therefore, accelerating this process is beneficial on the server side, but not on the client side. In the following sections, we describe in details the operation of the host CPU, followed by the GC accelerator on FPGA.

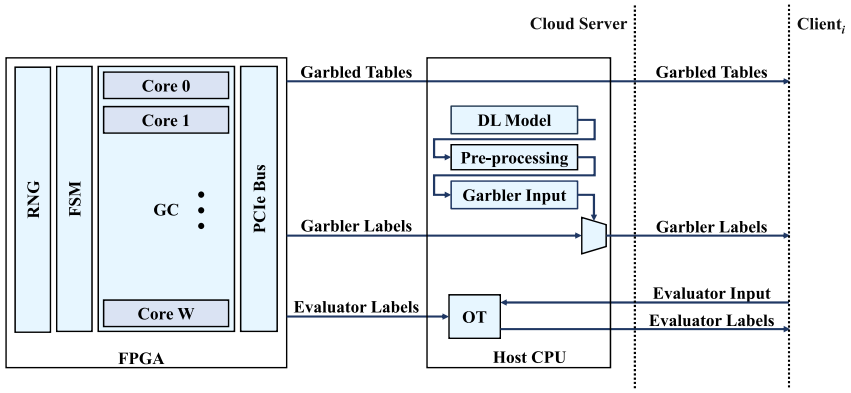

Fig. 3. ReDCrypt system architecture on the server side.

4.1 Host CPU

The host CPU in ReDCrypt framework is in charge of two main tasks:

(i) Transforming the traditional convolution layers into a matrix multiplication. To do so, ReDCrypt inserts a set of reshaping layers at the input of each convolution to unroll the input activation set as shown in Figure 2. It then reformats the thre-dimensional convolution weight tensors into a two-dimensional matrix based on the corresponding stride and kernel size in the target convolution layer. ReDCrypt provides a set of software codes based on TensorFlow to facilitate the conversion of convolution layers into matrix multiplication format and ensure ease of use by data scientists and engineers. The transformation of the underlying neural network into a set of subsequent matrix multiplication is a one-time pre-processing step before generating the corresponding netlist. The server only needs to repeat this step if and only if it decided to update its DL model topology/weights.

(ii) Sending the garbled labels to the client. The host CPU also gets involved in an oblivious transfer to communicate the evaluator labels/input with the client (Bob).

4.2 FPGA Accelerator

The garbling accelerator consists of the following components:

- —Multiple garbling cores to generate the garbled tables in parallel. Each core incorporates a GC engine and a memory block to store the generated garbled tables.
- —Label generator to create the wire labels required by the garbling operation. It incorporates a hardware Random Number Generator (RNG) to generate the random bit stream.
- —An FSM to govern the operation of the garbling cores. The FSM replaces the netlist in the conventional GC. This approach allows us to precisely control the garbling operation customized for the matrix-vector multiplication. Note that the netlist is embedded in the FSM. Therefore, the hardware acceleration is transparent to the evaluator (client) except for the speedup in service.
- —A PCI Bus to transfer the generated garbled tables to the CPU.

Motivation behind the Hardware Accelerator. In ReDCrypt, we chose to perform the parallel garbling operation on an FPGA-based accelerator rather than on a processor with multiple cores. There are several advantages of this approach. In a processor, the threads communicate among themselves through shared memory. To ensure that there are no race conditions or the threads do

not read stale variables, barriers are created both before and after a thread accessing that memory. The time overhead of the creation and release of the barriers is so high as compared to the time of generating one garbling table that eventually parallelizing the GC operation does not result in an improved timing. Parallelization of garbling operation on GPU is presented in Refs [28] and [29], but these works precede the row reduction optimization described in Section 2.2. Therefore, they do not manage the dependency among gates. FPGA allows us to precisely control the operation in sync with the clock. In the conventional GC, the underlying function (in this case, the matrix-vector multiplication) is described as a Boolean circuit (the netlist) and stored in the memory of both the garbler and the evaluator. In our implementation, instead of storing the circuit description in a file, we designed an FSM to generate the garbled tables according to the netlist. Thus, we can precisely schedule the operations to make sure that all the variables (in this case, the labels) are written and read in order *without the use of a barrier*.

5 CONFIGURATION OF THE GC CORES

We now present the design of the GC cores generating the garbled tables for privacy-preserving matrix-vector multiplication on the hardware accelerator. The product z of the weight matrix $W_{M\times N}$ and an input data vector $x_{N\times 1}$ is:

$$z[m] = \sum_{n=0}^{N-1} W[m,n] \times x[n],\ m = 0\ to\ M-1 \tag{1}$$

As such, the smallest unit of the matrix-vector multiplication comprises of a multiplier followed by an accumulator, i.e., an multiply-accumulate (MAC). The control flow of the FPGA accelerator comprises two nested loops. Following the methodology presented in Ref. [10], we design the MAC unit and garble (and evaluate) this unit sequentially for N rounds to compute one element of z. This forms the outer loop of the control flow. As described above, multiple parallel garbling cores are employed to generate the garbled tables for the MAC. The number of cores to be used in parallel depends on the input bit-width and available resource provisioning on the FPGA platform. In the inner loop, we break down the operation of the MAC unit such that (1) only one non-XOR operation is performed per core per clock cycle, and (2) at each cycle, no core is idle due to dependency issues. Note that the cores also contain 1 to 4 XOR gates at every cycle. However, due to the free-XOR optimization, they do not need costly encryption operations.

For the addition operation, we employ the implementation with the minimum number of non-XOR gates (1 AND gate per input bit) provided in Ref. [10]. However, the implementation of the multiplication operation in Ref. [10] follows a serial nature that does not allow parallelism. We leverage a tree-based structure for multiplication to maximize parallelism. Figure 4 shows the multiplication operation of two unsigned numbers with bit-width $b = 8$. The operation for signed numbers is discussed later in this section. The bits of x (i.e., their corresponding labels in GC operation) are constant over time for one multiplication, and the bits of W (i.e., their corresponding labels) are input to the system serially over time. The addition operations in Figure 4 represent one-bit full adder where the carry is transferred internally for the next cycle. In the following, we first describe the configuration of the parallel GC cores in the two segments marked in Figure 4 to coherently generate the necessary garbled tables. We then describe the accumulation operation and how we handle signed numbers.

5.1 Segment 1: MUX_ADD

The configuration of the parallel GC cores in segment 1 is displayed in Figure 5. Each block in the figure represents the logic operations performed by one core in every three clock cycles as shown

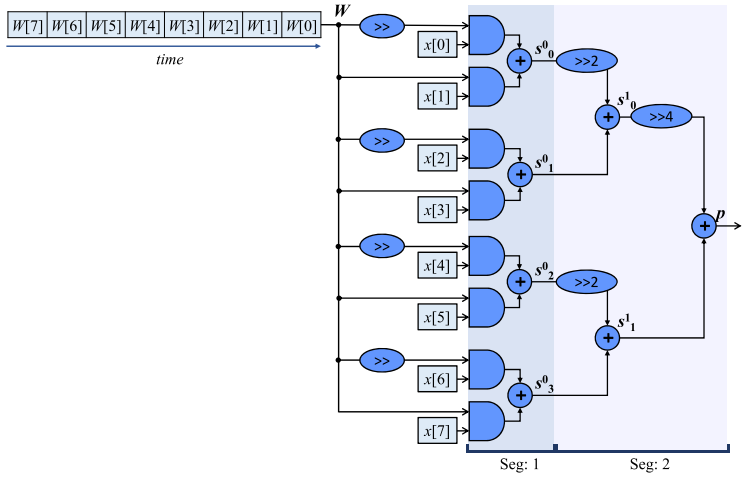

Fig. 4. Schematic depiction of the tree-base multiplication.

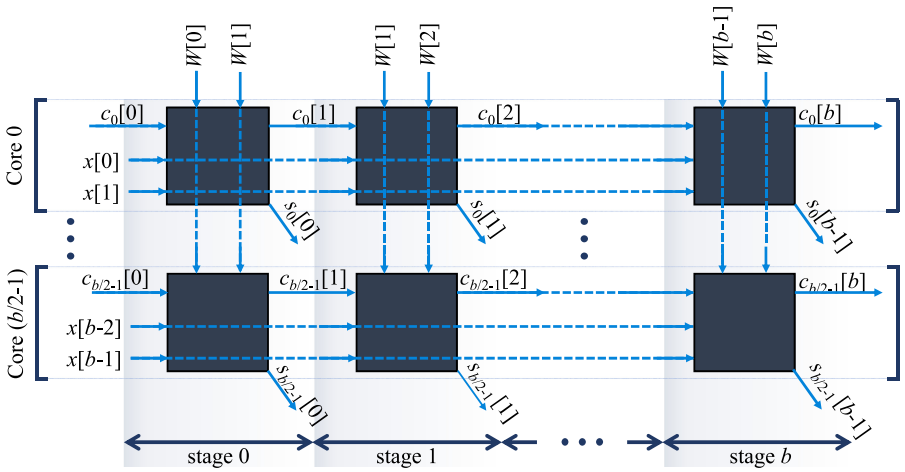

Fig. 5. The high-level configuration and functionality of the parallel GC cores in segment 1: MUX_ADD.

in Figure 6. Henceforth, we refer to every three clock cycles as one *stage*. Each GC core in this segment handles two AND gates and one adder per stage. The adder itself contains one AND and four XOR gates. The garbling engine of ReDCrypt can generate one garbled table per clock cycle, as described later in Section 6.1. Thus, generating the three garbled tables requires three clock cycles, i.e., one stage. Note that the XOR gates do not require the generation of garbled tables thanks to the free-XOR optimization [17]. Instead, the output labels of an XOR gate are generated by simply XORing the input labels.

Each core m receives the labels for the two corresponding bits of x: $x[m]$ and $x[m+1]$. These labels remain unchanged for the computation of one product. All the cores receive the labels of two bits of W: $W[n]$ and $W[n+1]$ at each stage n. However, since the garbled table for only one gate is generated every clock cycle, each core needs to import one label per cycle; thus, one k-bit input port is sufficient for getting the labels. The label for one bit of the W input is required for

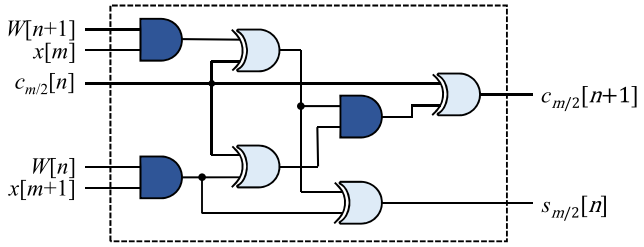

Fig. 6. Logic operations performed in one GC core.

two consecutive stages; thereby, at each stage after the first, one label is ported and the other one is shifted internally.

5.2 Segment 2: TREE

At each stage n, a single GC core in segment 1 generates the garbled table (and wire labels) for one bit of the sums: $s_0[n], \ldots, s_{b/2-1}[n]$. At the next stage, these sums are added up in segment 2 according to the tree structure. The shift operations in Figure 4 translate to delay operations as all the cores in segment 1 perform in parallel. The number of additions performed in this segment per stage is $b/2 + 1$. The GC cores in this segment are designed to perform three additions per core (three garbled tables, one per each addition) so that they are synchronized with segment 1. As such, there are $\lceil((b/2 - 1)/3)\rceil$ GC cores in this segment.

5.3 Accumulator, Support for Signed Inputs, and Relu

The final step of the MAC is the accumulator, which requires one addition per cycle. To support signed inputs, two MUX-2's complement pairs are placed at both input and output of the multiplier. Each pair incorporates two AND gates. Finally, the ReLu operation (if needed) is basically one MUX where the selector bit is connected to the sign-bit. Each MUX contains one AND gate. Our garbling accelerator operates in a pipelined fashion, which allows us to integrate these nine AND operations (10 with the ReLu) into segment 2. Even though this approach results in an increased number of the shift registers, it ensures the minimum number of idle cycles for the GC cores. Our implementation results (see Section 6) show that the bottleneck of the resources is the number of Look-Up Tables (LUT), not the number of registers (Flip-Flops). Therefore, our approach results in the most optimized design.

5.4 Scalability Analysis.

Scalability in terms of bit-width *b*. For bit-width b, the accelerator requires $b/2 + \lceil(b/2 + 8)/3\rceil$ cores per MAC. Thus, the maximum number of idle cores per MAC is 2. The computation of the output of one MAC takes $b + log(b) + 2$ stages. However, since the operations are pipelined, the throughput is 1 MAC per b stages per $b/2 + \lceil(b/2 + 8)/3\rceil$ cores.

Scalability in terms of the weight matrix dimensions $M \times N$ and depth of the network. The number of MAC operations increases linearly with the matrix dimension or the number of layers in the DL model (which dictates the number of matrix multiplication operations). For a constant number of cores, the computation time for one matrix-vector product increases linearly with either of M or N or the number of layers. For a constant computation time, the number of required cores increases linearly with either of M or N or the number of layers. Note that the configuration of the parallel cores dictates that the smallest increment in the number of cores is $b/2 + \lceil(b/2 + 8)/3\rceil$.

Unlike Homomorphic encryption-based frameworks, ReDCrypt is not sensitive to the depth of the underlying neural networks. HE protocol has a privacy/utility tradeoff. In other words, to obtain a higher privacy level, the utility of the system can decrease significantly. In particular, in the HE protocol, the noise introduced to the DL model as a result of securely encrypting data samples gets accumulated in each layer toward the output activations [5]. As such, the accuracy might degrade in deep neural networks with a large number of layers. GC protocol, on the other hand, does not induce such noise in the system due to the exact representation of the model as a Boolean circuit.

6 HARDWARE SETTING AND RESULTS

We implement the prototype of ReDCrypt on a Virtex UltraSCALE VCU108 (XCVU095) FPGA. A system with Ubuntu 16.04, 128 GB memory, and Intel Xeon E5-2600 CPU @ 2.2GHz is employed as the general purpose processor hosting the FPGA. The software realization for comparison purposes is executed on the same CPU. We leverage PCIe library provided by Ref. [30] to interconnect the host and FPGA platforms. Vivado 2017.3 is used to synthesize our GC units.

6.1 GC Engine

Each GC core incorporates one GC engine that generates one garbled table per clock cycle. The GC engine adopts all the optimizations described in Section 2.3. Our implementation involves only two logic gates: AND and XOR. The GC engine takes as its input the labels for the two input wires of the AND gate and outputs the output label and the two rows of the corresponding garbled tables. According to the methodology presented in Ref. [14], the encryption is performed by fixed-key block cipher instantiated with AES. We employ four instantiations of a single stage AES implementation to perform the four required AES encryptions in parallel. The s-boxes [31] inside the AES algorithm are implemented efficiently by utilizing the LUTRAMs on the Virtex UltraSCALE FPGA. The unique gate identifier T is generated by concatenating n (see Equation (1)), core ID, stage index, and and gate ID (see Figure 5). Due to the free XOR optimization, XOR gates just require XORing the two input labels and are handled outside while the GC engine is designed to generate garbled tables only for the AND gates. This approach ensures that there is no mismatch in the timing for executing different gates as in Ref. [8] and, therefore, no stalling caused by dependency issues.

The labels and garbled tables are stored in the on-chip memory of the FPGA. The memory is divided into blocks with one input port per block and one output port for the entire memory. The output port is used by the PCIe Bus to transfer the generated input labels and garbled tables to the general purpose processor hosting the FPGA. Since each core has its own block in the memory with an individual input port, logically, it can be visualized as each core having its own memory block.

6.2 Label Generator

To generate the wire labels for GC, we implement on-chip hardware RNG. We adopt the Ring Oscillator (RO) based RNG suggested in Ref. [32]. Each RO contains three inverters and a single RNG XORs the output of 16 ROs. The entropy of the implemented RNG on our evaluation platform is thoroughly evaluated by National Institute of Standards and Technology (NIST) battery of randomness tests [33]. In the worst-case scenario, the GC accelerator requires $k \times (b/2)$ random bits/cycle. However, for a large portion of the operation, it requires only k bits/cycle on average. The label generator incorporates $k \times (b/2)$ RNGs such that it can support the worst-case setting. The FSM that synchronizes the garbling operation, fully or partially turns off the operation of the RNGs to conserve energy, when possible.

Table 2. Resource Usage of One MAC Unit

Bit-width (b)	8	16	32
LUT	2.95E+04	5.91E+04	1.11E+05
LUTRAM	1.28E+02	3.84E+02	6.40E+02
Flip-Flop	2.44E+04	4.88E+04	8.40E+04

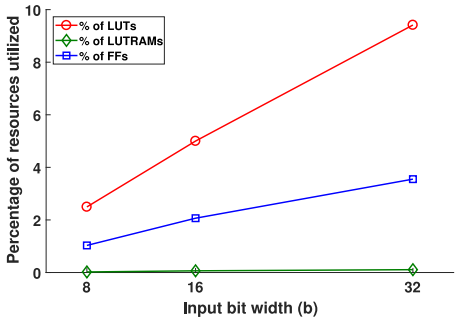

Fig. 7. Percentage resource utilization per MAC for different bit-widths.

6.3 Resource Utilization

The FPGA resource utilization of one MAC unit is shown in Table 2 for different bit-widths b. It can be seen from the table that the underlying resource utilization of our design increases linearly with b. We do not compare the resource utilization with the prior-art GC implementation on FPGA [9] for two reasons: (i) [9] being a generic GC implementation, it is difficult to estimate the resource it would require only to perform the MAC operation in a similar number of clock cycles as this work; (ii) it employs SHA-1 for encryption (the most resource consuming part of the implementation), while we employ AES. SHA-1 is not considered secure anymore and all the current GC realizations in both software and hardware employ AES.

We plot the percentage of resources utilized per bit-width for one MAC unit in Figure 7. It can be seen from the plot that the bottleneck of the design on this platform is the number of LUTs, which reaches a peak of around 10% for $b = 32$. The maximum clock frequency supported by this implementation is 200MHz on the Virtex UltraSCALE.

6.4 Performance Comparison with the Prior-art GC Implementation

To the best of our knowledge, ReDCrypt is the first FPGA implementation of privacy-preserving deep learning. Table 3 compares the throughput of ReDCrypt against the fastest available software GC framework TinyGarble [10] and the FPGA GC solution presented in Ref. [9]. Both ReDCrypt and Ref. [9] employ parallel GC cores to accelerate the operation. In ReDCrypt, the maximum number of parallel cores depends on the available resources in the FPGA, while in Ref. [9], it depends on the latency of garbling one AND gate and available BRAMs on FPGA. Considering all these, we believe that reporting the overall throughput would be ambiguous and somewhat unfair to the software framework [10]. Therefore, we report the throughput of all the frameworks per core.

As shown in Table 3, ReDCrypt accelerates the garbling operation by up to 57 times compared to Ref. [10] and at least 985 times compared to Ref. [9]. Another recent GC realization on FPGA, GarbledCPU [8] do not report timing results for multiplication and addition. However, they report

Table 3. Throughput Comparison of ReDCrypt with State-of-the-art GC Frameworks

	TinyGarble [10] on CPU			FPGA Overlay Architecture [9]			ReDCrypt on FPGA		
Bit-width	8	16	32	8	16†	32	8	16	32
Clock Cycle per MAC	1.44E+05	5.45E+05	2.24E+06	4.40E+03	1.20E+04	3.60E+04	24	48	96
Time per MAC (μs)	42.29	160.35	657.65	22.00	60.00	180.00	0.12	0.24	0.48
Throughput (MAC per sec)	2.36E+04	6.24E+03	1.52E+03	4.55E+04	1.67E+04	5.56E+03	8.33E+06	4.17E+06	2.08E+06
No. of cores	1	1	1	43	43	43	8	14	24
Throughput per core (MAC per sec)	2.36E+04	6.24E+03	1.52E+03	1.06E+03	3.88E+02	1.29E+02	1.04E+06	2.98E+05	8.68E+04
× Throughput of ReDCrypt per core	1/44	1/48	1/57	1/985	1/768	1/672	-	-	-

†Interpolated from the results provided in Ref. [9] for 8, 32, and 64 bits.

Table 4. Number of XOR and Non-XOR Gates, Amount of Communication, and Computation Time for Each Benchmark

		#non-XOR		#XOR		Comm. (GB)		Comp. (ms)	
Id	DL Architecture	b = 8	b = 16	b = 8	b = 16	b = 8	b = 16	b = 8	b = 16
1	28×28-5C2-ReLu-100FC-ReLu-10FC-Softmax	2.09E+07	6.95E+07	5.56E+07	1.67E+08	0.68	2.25	13.04	26.07
2	28×28-300FC-ReLu-100FC-ReLu-10FC-Softmax	5.11E+07	1.70E+08	1.36E+08	4.09E+08	1.67	5.52	31.94	63.89
3	617-50FC-ReLu-26FC-Softmax	6.17E+06	2.06E+07	1.65E+07	4.94E+07	0.20	0.67	3.86	7.72
4	5625-2000FC-ReLu-500FC-ReLu-19FC-Softmax	2.35E+09	7.85E+09	6.28E+09	1.88E+10	76.90	254.22	1471.14	2942.28

2× improvement in throughput compared to JustGarbled [14] (which is the back-end of Ref. [10]) on an Intel Core i7-2600 CPU @ 3.4GHz. We estimate at least 37 times improvement over Ref. [8] in throughput per core (this work does not attempt parallelization). Due to pipeline stalls caused by dependency issues, the throughput of Ref. [10] is likely to go down further while garbling a complete netlist.

To be fair, we should state that a major factor behind the lower throughput of Refs [9] and [10] is their focus on general purpose GC computing while ReDCrypt is custom made for performing DL inference only. However, the large enhancement in throughput establishes the practicality of the custom solution.

7 PRACTICAL DESIGN EXPERIMENTS

In this section, we evaluate ReDCrypt framework for realization of both deep learning (Section 7.1) and generic matrix-based machine learning applications (Section 7.2).

7.1 Deep Learning Benchmarks

We evaluate ReDCrypt performance for the realization of four different DL benchmarks. Our benchmarks include both DNN and CNN models for analyzing visual, audio, and smart-sensing datasets. Table 4 details the computation on the server side and the transferred Bytes for each client in each benchmark. The topology of our benchmarks is outlined in the following.

Benchmark 1. Detecting objects in an image is a key enabler in devising various artificial intelligence and learning tasks. We evaluate ReDCrypt practicability in analyzing the Mixed-NIST (MNIST) dataset [34] using two different DL architectures. This data contains hand-written digits represented as 28×28 pixel grids, where each pixel is denoted by a gray level value in the range of 0–255. In this experiment, we train and use a five-layer CNN for document classification as suggested in Ref. [5]. The five layers include: (i) a convolutional layer with a kernel of size 5×5, a stride of $(2, 2)$, and a map-count of 5. This layer outputs a matrix of size $5 \times 13 \times 13$.

(ii) A ReLu layer as the non-linearity activation function. (iii) A fully-connected layer that maps the ($5 \times 13 \times 13 = 865$) units computed in the previous layers to a 100-dimensional vector. (iv) Another ReLu non-linearity layer, followed by (v) a final fully-connected layer of size 10 to compute the probability of each inference class.

Benchmark 2. We train and use LeNet-300-100 as described in Ref. [35] for the MNIST dataset [34]. LeNet-300-100 is a classical feed-forward neural network consisting of three fully-connected layers interleaved with two non-linearity layers (ReLu) with a total of $267K$ DL parameters.

Benchmark 3. Processing audio data is an important step in devising different voice activated learning tasks that appear in mobile sensing, robotics, and autonomous applications. Our audio data collection consists of approximately 1.25 hours of speech collected by 150 speakers [36]. In this experiment, we train and use a three-layer fully-connected DNN of size ($617 \times 50 \times 26$) with ReLu as the non-linear activation function to analyze data within 5% inference error.

Benchmark 4. Analyzing smart-sensing data collected by embedded sensors such as accelerometers and gyroscopes is a common step in the realization of various learning tasks. In our smart-sensing data analysis, we train and use a four-layer fully-connected DNN of size ($5625 \times 2000 \times 500 \times 19$) with ReLu as the non-linear activation function to classify 19 different activities [37] within 5% inference error.

7.2 Generic ML Applications

Even though ReDCrypt is designed to accelerate deep learning inference, it can greatly enhance the performance of many other machine learning applications. In this section, we analyze a number of well-known ML applications to assess the speedup provided by the custom FPGA realization of a GC-based MAC operation. We assume a 32-bit fixed point system with 24 cores on ReDCrypt. Note that the throughput can be increased linearly by adding more GC cores to the FPGA. For example, about 25 times more GC cores can fit in our current implementation platform.

Recommendation System. The movie recommendation system in Ref. [38] presents an efficient implementation of privacy-preserving matrix factorization, which has been widely adopted by many other works such as Refs [39] and [40]. More than two-thirds of the execution time in Ref. [38] is spent on matrix-vector multiplication for gradient computations. The complexity of the proposed matrix factorization is $O(Mlog^2M)$ where M is the total number of ratings, while the complexity of the pertinent MAC operations in each operation is $O(Sd)$ where S is summation of the total number of ratings and total number of movies, and d is the dimension of user/item profile. On the MovieLens dataset, each iteration of Ref. [38] takes 2.9hr. Incorporating our hardware accelerated MAC into the approach of Ref. [38] significantly reduces the gradient computation time, decreasing the total runtime per iteration from 2.9hr to 1hr (69% improvement).

Ridge Regression. This method is used to find the best-fit curve through a set of data points. The work in Ref. [41] combines both homomorphic encryption and Yao garbled circuits to efficiently perform privacy-preserving ridge regression. Their approach has $O(d^3)$ MACs, $O(d)$ square roots, and $O(d^2)$ divisions in the first phase and $O(d^2)$ MAC operations in the second phase. As such, accelerating the MAC operations would significantly improve the runtime as shown in Table 5 for selected datasets used in Ref. [41]. n and d are the number of samples and feature size, respectively.

Portfolio Analysis. To calculate the risk to return ratio based on the stock portfolio of the investor, the client stock weight vector w (which contains relative weight of stocks in the investor's portfolio) and the financial institution stock covariance matrix cov (which is the result of a

Table 5. Ridge Regression Runtime Improvement

Name	n	d	Time (s) ([41])	Time (s) (Ours)	Runtime Impr.
	28.4 ×				
forestFires	517	12	46	1.8	24.5 ×
winequality-red	1599	11	39	1.7	22.6 ×
autompg	398	9	21	1.1	18.7 ×
concreteStrength	1030	8	17	1.0	16.8 ×

financial institution's research on the market) are required. The risk to return ratio is then obtained by performing $w \times cov \times w'$ where w' is the transpose of w [42]. In Ref. [43], the authors reported 20μs to perform 252 rounds of risk to return analysis for a portfolio of size 2 on an Nvidia-k80 GPU. According to our evaluation, the same computation with privacy-preserving would take 1.33secs using TinyGarble and 15.23ms using ReDCrypt.

In the above analysis, we assumed that the cloud server has a sufficient number of communication channels and bandwidth. However, after a certain threshold, communication capability of the server may become the bottleneck of the operation. Note that ReDCrypt does not affect the pertinent accuracy of the model in any of the benchmarks described above.

8 RELATED WORK

Authors in Ref. [44] have suggested the use of secure function evaluation protocols to securely evaluate a DL model. Their proposed approach, however, is an interactive protocol in which the data owner needs to first encrypt the data and send it to the cloud. Then, the cloud server should multiply the encrypted data with the weights of the first layer, and send back the results to the data owner. The data owner decrypts, applies the pertinent non-linearity, and encrypts the result again to send it back to the cloud server for the evaluation of the second layer of the DL model. This process continues until all the layers are computed. There are several limitations with this work [44]: (i) it leaks partial information embedded in the weights of the DL model to the data owner. (ii) It requires the data owner to have a constant connection with the cloud server while evaluating the DL network. To address the issue of information leakage as a result of sharing the intermediate results, Refs [45] and [46] enhance the protocol initially proposed in Ref. [44] to obscure the weights. However, even these works [45, 46] still need to establish a constant connection with the client to delegate the non-linearity computations after each hidden layer to the data owner and do not provide the same level of security provided by ReDCrypt.

Most recently, several Cryptographic frameworks have been reported in the literature to enable DL computations in a privacy-preserving setting, e.g., Refs [5, 6, 47, 48]. The existing frameworks, however, are not amenable to be used for real-time DL cloud services. In particular, the existing secure DL frameworks either rely on large batch sizes for an optimized performance [5], or require a second non-colluding server to execute their security primitives [6]. More importantly, to the best of our knowledge, none of the prior works have provided a hardware accelerated solution to optimize system performance for secure execution of DL models.

A number of recent works [8, 9] have presented an implementation of GC on FPGA. However, their primary focus is on the versatility of the framework rather than computational efficiency. In Ref. [8], the underlying netlist is always that of a MIPS processor where the secure function is loaded as a set of instructions. They report two times improvement in throughput (by employing a single core) over the fastest software realization at that time, while our ReDCrypt achieves 57 times increase in throughput per core. The work in Ref. [9], which targets privacy-preserving data

mining applications, presents an FPGA overlay architecture [49] where the overlay circuit contains implementations of garbled components (logic gates) upon which the netlist of the secure function is loaded. Overlay architectures, in general, require 40× to 100× more LUTs compared to the conventional design approach [49]. Even though Ref. [9] achieves increased throughput by employing parallel cores, its throughput per core is about 17 times smaller compared to the software realization of Ref. [10]. As explained above, in the majority of the DL computations, the privacy-sensitive computation boils down to a matrix-vector multiplication. Therefore, our customized architecture on FPGA results in orders of magnitude higher throughput compared to the generic GC realizations.

A preliminary version of this work has appeared in Ref. [50] in which we had designed a standalone hardware accelerator for the privacy-preserving MAC. In this article, we extend our work in Ref. [50] in several ways.

—Providing an end-to-end architecture supporting DL inference.
—Devising automated customization tools to transform conventional DL models into matrix-vector multiplication suitable for privacy-preserving GC acceleration.
—Implementing custom privacy-preserving non-linearity layers typically used in DL models.
—Benchmarking different neural network topologies for analysis of visual, audio, and smart-sensing data.

9 CONCLUSION

We present ReDCrypt, a novel practical and provably-secure DL framework that enables the cloud servers to provide high-throughput and power-efficient service to distributed clients. The security primitive of our framework does not involve any tradeoff between accuracy and privacy. ReDCrypt targets streaming settings where the DL inference results should be computed in real time (batch size = 1). We devise custom hardware acceleration on FPGA that provides 57-fold higher throughput compared to the state-of-the-art GC solution for DL inference.

REFERENCES

[1] Li Deng and Dong Yu. 2014. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing* 7, 3–4 (2014).
[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436.
[3] Nicola Joneset al. 2014. The learning machines. *Nature* 505, 7482 (2014), 146–148.
[4] Jeremy Kirk. 2016. IBM join forces to build a brain-like computer. Retrieved from http://www.pcworld.com/article/2051501/universities-join-ibm-in-cognitive-computing-researchproject.html.
[5] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on Machine Learning*. 201–210.
[6] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A system for scalable privacy-preserving machine learning. *IACR Cryptology ePrint Archive* 2017 (2017), 396.
[7] Kimmo Järvinen, Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. 2010. Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs. In *CHES*, Vol. 10. Springer, 383–397.
[8] Ebrahim M. Songhori, Shaza Zeitouni, Ghada Dessouky, Thomas Schneider, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. 2016. GarbledCPU: A MIPS processor for secure computation in hardware. In *Proceedings of the 53rd Annual Design Automation Conference (DAC)*. ACM, 73.
[9] Xin Fang, Stratis Ioannidis, and Miriam Leeser. 2017. Secure function evaluation using an FPGA overlay architecture. In *FPGA*. 257–266.
[10] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. 2015. TinyGarble: Highly compressed and scalable sequential garbled circuits. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*. IEEE, 411–428.

[11] Xiao Wang, S. Dov Gordon, Allen McIntosh, and Jonathan Katz. 2016. Secure computation of MIPS machine code. In *ESORICS*. Springer.
[12] Moni Naor and Benny Pinkas. 2005. Computationally secure oblivious transfer. *Journal of Cryptology* 18, 1 (2005), 1–35.
[13] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science, 1986*. IEEE, 162–167.
[14] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. 2013. Efficient garbling from a fixed-key blockcipher. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP)*. IEEE, 478–492.
[15] Donald Beaver, Silvio Micali, and Phillip Rogaway. 1990. The round complexity of secure protocols. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*. ACM, 503–513.
[16] Moni Naor, Benny Pinkas, and Reuban Sumner. 1999. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*. ACM, 129–139.
[17] Vladimir Kolesnikov and Thomas Schneider. 2008. Improved garbled circuit: Free XOR gates and applications. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*. Springer, 486–498.
[18] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two halves make a whole. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 220–250.
[19] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *Crypto*, Vol. 2729. Springer, 145–161.
[20] Ben Kreuter, Abhi Shelat, Benjamin Mood, and Kevin Butler. 2013. PCF: A portable circuit format for scalable two-party secure computation. In *Presented as Part of the 22nd USENIX Security Symposium (USENIX Security 13)*. 321–336.
[21] Yehuda Lindell and Benny Pinkas. 2007. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT'07 (LNCS)*, Vol. 4515. Springer, 52–78.
[22] Jesper Buus Nielsen and Claudio Orlandi. 2009. LEGO for two-party secure computation. In *TCC'09 (LNCS)*, Vol. 5444. Springer, 368–386.
[23] Abhi Shelat and Chih-hao Shen. 2011. Two-output secure computation with malicious adversaries. In *EUROCRYPT'11 (LNCS)*, Vol. 6632. Springer, 386–405.
[24] Yehuda Lindell and Benny Pinkas. 2012. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology* 25, 4 (2012), 680–722.
[25] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. ACM, 784–796.
[26] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, and Jason Cong. 2016. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*. ACM, 326–331.
[27] Zhourui Song, Zhenyu Liu, Chunlu Wang, and Dongsheng Wang. 2017. Computation error analysis of block floating point arithmetic oriented convolution neural network accelerator design. *arXiv preprint arXiv:1709.07776* (2017).
[28] Shi Pu, Pu Duan, and Jyh-Charn Liu. 2011. Fastplay—A parallelization model and implementation of SMC on CUDA based GPU cluster architecture. *IACR Cryptology ePrint Archive* (2011).
[29] Nathaniel Husted, Steven Myers, Abhi Shelat, and Paul Grubbs. 2013. GPU and CPU parallelization of honest-but-curious secure two-party computation. In *Proceedings of the Computer Security Applications Conference*. ACM.
[30] XILLYBUS. 2017. Retrieved from http://xillybus.com/.
[31] Joan Daemen and Vincent Rijmen. 2013. The Rijndael Block Cipher.
[32] Knut Wold and Chik How Tan. 2009. Analysis and enhancement of random number generator in FPGA based on oscillator rings. *International Journal of Reconfigurable Computing* 2009 (2009), 4.
[33] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, and Elaine Barker. 2001. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. Technical Report 800-22. NIST.
[34] Yann LeCun, Corinna Cortes, and Christopher Burges. 2017. MNIST dataset. Retrieved from http://yann.lecun.com/exdb/mnist/.
[35] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[36] UCI machine learning repository. 2017. UCI machine learning repository: ISOLET data set. Retrieved from https://archive.ics.uci.edu/ml/datasets/isolet.
[37] UCI machine learning repository. 2017. UCI machine learning repository: Daily and sports activities data set. Retrieved from https://archive.ics.uci.edu/ml/datasets/Daily+and+ Sports+Activities.
[38] Valeria Nikolaenko, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. 2013. Privacy-preserving matrix factorization. In *Proceedings of the Conference on Computer & Communications Security*. ACM.
[39] Florian Kerschbaum, Thomas Schneider, and Axel Schröpfer. 2014. Automatic protocol selection in secure two-party computations. In *Proceedings of the International Conference on Applied Cryptography and Network Security*. Springer.

[40] Xiao Shaun Wang, Yan Huang, T. H. Hubert Chan, Abhi Shelat, and Elaine Shi. 2014. SCORAM: Oblivious RAM for secure computation. In *CCS*. ACM.
[41] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. 2013. Privacy-preserving ridge regression on hundreds of millions of records. In *Proceedings of the Symposium on S & P*. IEEE.
[42] Gregory Connor, Lisa R. Goldberg, and Robert A. Korajczyk. 2010. *Portfolio Risk Analysis*. Princeton University Press.
[43] Javier Alejandro Varela and Norbert Wehn. 2017. Near real-time risk simulation of complex portfolios on heterogeneous computing systems with OpenCL. In *Proceedings of the International Workshop on OpenCL*. ACM.
[44] Mauro Barni, Claudio Orlandi, and Alessandro Piva. 2006. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th Workshop on Multimedia and Security*. ACM, 146–151.
[45] Claudio Orlandi, Alessandro Piva, and Mauro Barni. 2007. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security* 2007 (2007), 18.
[46] Alessandro Piva, Claudio Orlandi, M. Caini, Tiziano Bianchi, and Mauro Barni. 2008. Enhancing privacy in remote data classification. In *Proceedings of the IFIP International Information Security Conference*. Springer, 33–46.
[47] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. 2017. Oblivious neural network predictions via MiniONN transformations. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.
[48] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. 2017. DeepSecure: Scalable provably-secure deep learning. *arXiv preprint arXiv:1705.08963* (2017).
[49] Alexander Brant and Guy G. F. Lemieux. 2012. ZUMA: An open FPGA overlay architecture. In *Field-Programmable Custom Computing Machines*. IEEE, 93–96.
[50] Siam U. Hussain, Bita Darvish Rouhani, Mohammad Ghasemzadeh, and Farinaz Koushanfar. 2018. MAXelerator: FPGA accelerator for privacy preserving multiply-accumulate (MAC) on cloud servers. In *Proceedings of the 55th Annual Design Automation Conference (DAC)*. ACM.

Received December 2017; revised April 2018; accepted July 2018