DeepSecure: Scalable Provably-Secure Deep Learning

Bita Darvish Rouhani
UC San Diego
bita@ucsd.edu

M. Sadegh Riazi
UC San Diego
mriazi@ucsd.edu

Farinaz Koushanfar
UC San Diego
farinaz@ucsd.edu

ABSTRACT

This paper presents DeepSecure, the an scalable and provably secure Deep Learning (DL) framework that is built upon automated design, efficient logic synthesis, and optimization methodologies. DeepSecure targets scenarios in which neither of the involved parties including the cloud servers that hold the DL model parameters or the delegating clients who own the data is willing to reveal their information. Our framework is the first to empower *accurate* and *scalable* DL analysis of data generated by distributed clients without sacrificing the security to maintain efficiency. The secure DL computation in DeepSecure is performed using *Yao's Garbled Circuit* (GC) protocol. We devise GC-optimized realization of various components used in DL. Our optimized implementation achieves up to *58-fold* higher throughput per sample compared with the best prior solution. In addition to the optimized GC realization, we introduce a set of novel low-overhead pre-processing techniques which further reduce the GC overall runtime in the context of DL. Our extensive evaluations demonstrate up to *two orders-of-magnitude* additional runtime improvement achieved as a result of our pre-processing methodology.

KEYWORDS

Privacy-Preserving Deep Learning, Secure Function, Evaluation, Garbled Circuit, Automated Design, Logic Synthesis

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 human brain [1]. Technology leaders such as Microsoft, Google, IBM, and Facebook are devoting millions of dollars to devise accurate DL models for various artificial intelligence and data inference applications ranging from social networks and transportations to environmental monitoring and health care [2–4]. The applicability of DL models, however, is hindered in settings where the risk of data leakage raises serious privacy concerns. Examples of such applications include cloud-based applications where clients hold sensitive private information, e.g., medical records, financial data, or location.

To employ DL models in a privacy-preserving setting, it is imperative to devise computing frameworks in which neither of the involving parties is required to reveal their information. Several research works have been developed to address privacy-preserving computing for DL networks, e.g., [5, 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 [5, 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 trade-off 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 real-world settings.

This paper introduces DeepSecure, the first provably-secure framework for *scalable* DL-based analysis of data collected by distributed clients. DeepSecure enables applying the state-of-the-art DL models on sensitive data without sacrificing the accuracy to obtain security. Consistent with the literature, we assume an *honest-but-curious* adversary model for a generic case where both DL parameters and input data must be kept private. DeepSecure proposes the use of Yao's Garbled Circuit (GC) protocol to securely perform DL execution. We empirically corroborate that our framework is significantly efficient for scenarios in which the number of samples collected by each distributed client is less than 2600 samples; the clients send the data to the server for processing with the least possible delay. Our approach is well-suited for streaming 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. In DeepSecure framework, we further provide mechanisms based on secret sharing to securely delegate GC computations to a third party for constrained embedded devices.

The function to be securely evaluated in GC should be represented as a list of Boolean logic gates (a.k.a., *netlist*). We generate the netlists required for deep learning using logic synthesis tools with GC-optimized custom libraries as suggested in [7]. The computation and communication workload of GC protocol in DeepSecure framework is explicitly governed by the number of neurons in the target DL network and input data size. We further introduce a set of novel low-overhead pre-processing techniques to reduce data and DL network footprint without sacrificing neither the accuracy nor the data confidentiality. Our pre-processing approach is developed based on two sets of innovations: (i) transformation of input data to an ensemble of lower-dimensional subspaces, and (ii) avoiding the execution of neutral (inactive) neurons by leveraging the sparsity structure inherent in DL models.

The explicit contributions of this paper are as follows:

- Proposing DeepSecure, the first provably-secure framework that simultaneously enables *accurate* and *scalable* privacy-preserving DL execution for distributed clients.
- Devising new custom libraries to generate *GC-optimized netlists* for required DL network computations. Our custom library is built upon automated design, efficient logic synthesis, and optimization methodologies.
- Incepting the idea of *data and DL network* pre-processing in the secure function evaluation settings. Our approach leverages the fine- and coarse-grained data and DL network

DAC '18, June 24–29, 2018, San Francisco, CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196023

parallelism to avoid unnecessary computation/communication in the execution of Yao's GC protocol.
- Providing proof-of-concept evaluations of various visual, audio, and smart-sensing benchmarks. Our evaluations corroborate DeepSecure's scalability and practicability for distributed users compared with the HE-based solution.

2 PRELIMINARIES

2.1 Deep Learning Networks

DL 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).

A DL model is formed by stacking different hidden layers on top of each other. As we discuss in Section 3.3, common hidden layers (e.g., non-linearities, matrix-multiplication, convolution) used in the DL can be effectively represented as a Boolean circuit. Note that many of the computations involved in DL inference, such as non-linearity layers, cannot be accurately represented by polynomials used in HE. E.g., approximating a Rectified Linear unit (ReLu) in HE requires using high-order polynomials; whereas a ReLu can be accurately represented by a Multiplexer in Boolean circuit.

2.2 Cryptographic Protocols

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

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 potentially many pieces of information provided by S. In a 1-out-of-n OT protocol, the sender S has n messages $(x_1, ..., x_n)$ 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 [8].

Garbled Circuit. Yao's garbled circuit protocol [9] 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. The function f should be represented as a Boolean circuit with 2-input gates (e.g., XOR, AND, etc.). Alice (a.k.a., Garbler) garbles the circuit by assigning two random keys to each wire in the circuit. For each gate, a garbled table is computed that allows decryption of the gate's output key given its two input keys. Alice then sends the encrypted circuit along with her corresponding input keys to Bob (a.k.a., Evaluator). To evaluate f, Bob needs to acquire his input keys without revealing his actual input data to Alice. To do so, Bob obtains his input keys obliviously through a 1-out-of-2 OT protocol and uses them to evaluate the garbled circuit gate by gate. Finally, Bob shares the encrypted output with Alice. The encrypted output (e.g., inference label) is then decrypted by Alice using the mapping keys.

There are several optimizations for the GC protocol including Free-XOR [10], Row Reduction [11], Half Gates [12], and Fixed-Key Cipher [13]. 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. We have leveraged all the aforementioned optimization techniques to provide an efficient GC-based deployment of DeepSecure framework. Following the approach presented in [7], DeepSecure uses Hardware Description Language (HDL) to describe the pertinent function f which is then compiled with a logic synthesis tool to generate the netlist.

3 DeepSecure FRAMEWORK

Figure 1 demonstrates the overall flow of DeepSecure framework. DeepSecure consists of two main components to securely perform data inference in the context of deep learning: (i) GC-optimized execution of the target DL model (Section 3.1), and (ii) data and DL network transformation (Section 3.2).

3.1 DeepSecure GC Core Structure

In a DL-based cloud service, a cloud server (Bob) holds the DL model parameters trained for a particular application, and a delegated client (Alice) owns a data sample for which she wants to securely find the corresponding classified label (a.k.a., inference label).

DeepSecure 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, the netlist of the publicly known DL architecture[1] should be generated prior to the execution of the GC protocol. The execution of the GC protocol involves four main steps: (i) the client (data owner) garbles the Boolean circuit of the DL architecture. (ii) The client sends the computed garbled tables from the first step to the cloud server along with her input wire labels. Both client and the cloud server then engage in a 1-out-of-2 Oblivious Transfer [8] protocol to obliviously transfer the wire labels associated with cloud server's inputs. (iii) The cloud server evaluates the garbled circuit and computes the corresponding encrypted data inference. (iv) The encrypted result is sent back to the client to be decrypted using the garbled keys so that the true inference label is revealed.

GC Communication and Computation Overhead. Table 1 details the computation and communication cost for execution of a fully-connected DNN. A similar setup applies to the CNN models in which a set of convolutions are performed per layer. The total communication needed between client and the server is proportional to the number of non-XOR gates since only the garbled tables for non-XOR gates need to be transferred. In DeepSecure framework, garbling/evaluating each non-XOR and XOR gate requires 164 and 62 CPU clock cycles (*clks*) on average, respectively.

As shown in Table 1, the computation and communication overhead of DL execution using GC protocol is explicitly governed by the number of neurons (units) per DL layer. As such, we suggest a set of data and DL network transformation as an arbitrarily pre-processing step to reduce the computation and communication overhead of GC protocol for DL inference.

3.2 Data and DL Network Pre-processing

DeepSecure pre-processing consists of two main steps: (i) data projection (Section 3.2.1), and (ii) DL network distillation (Section 3.2.2).

3.2.1 Data Projection

The input layer size of a neural network is conventionally dictated by the feature space size of the input data samples. Many complex modern data matrices that are not inherently low-rank can be modeled by a composition of multiple lower-rank subspaces. This type of composition of high-dimensional dense (non-sparse but structured) data as an ensemble of lower dimensional subspaces has been used earlier by data scientists and engineers to facilitate knowledge extraction or achieve resource efficiency [14–16]. DeepSecure, for

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

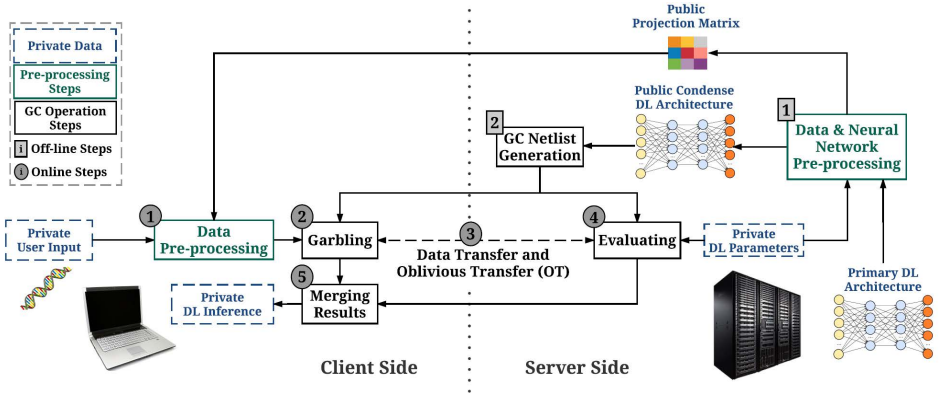

Figure 1: Global flow of DeepSecure framework including both off-line (indexed by rectangular icons) and online (indexed by oval icons) steps. The operations shown in the left hand side of the figure are executed by the client (Alice) while the operations on the right hand side are performed by the server (Bob).

the first time, introduces, implements, and automates the idea of data pre-processing as a way to achieve performance optimization for GC execution of a DL model.

As we empirically demonstrate in Section 4, the main bottleneck in the execution of GC protocol is the communication overhead between the server (Bob) and client (Alice). Therefore, we focus our pre-processing optimization to minimize the GC communication workload (T_{comm}) customized to the application data and DL model (see Table 1 for the characterization of communication overhead in accordance with the data and DL network size). In order to perform data pre-processing in DeepSecure framework, the server needs to re-train its private DL parameters according to the following objective function:

$$\begin{aligned} &\underset{D,\ \tilde{DL}_{param}}{Minimize} \quad (T_{comm}) \ s.t., \ ||A - DC||_F \leq \epsilon ||A||_F \\ &\qquad\qquad \delta(\tilde{DL}_{param}) \leq \delta(DL_{param}) \\ &\qquad\qquad l \leq m, \end{aligned} \tag{1}$$

where $A_{m\times n}$ is the raw input training data (owned by the server) that we want to factorize into a *dictionary matrix* $D_{m\times l}$ and a low-dimensional *data embedding* $C_{l\times n}$ that is used to re-train the DL model. Here, $\delta(.)$ is the partial validation error corresponding to the pertinent DL parameters. We use DL_{param} to denote the initial DL parameters acquired by training the target DL model using raw data features (A). Whereas, $\tilde{DL}_{param}$ indicates the updated parameters after re-training the underlying DL model using the projected data embedding C. $||\cdot||_F$ denotes the Frobenius norm and ϵ is an intermediate approximation error that casts the rank of the input data. We leveraged the streaming-based data sketching methodology suggested in [14] to solve the optimization objective outlined in Equation (1).

Once the DL model is re-trained using the projected data embedding C, we define a projection matrix ($W = DD^+$)[2] which is publicly released to be used by the clients during DL execution phase. As we will discuss in Section 3.4, W does not reveal any information regarding the training data nor the DL parameters. We emphasize that re-training of conventional DL model is a *one-time off-line* process performed by the server. As such, the data pre-processing overhead during GC execution only involves a matrix-vector multiplication, $Y_i = WX_i$, that is performed prior to garbling on the client side. Here, X_i indicates the raw data owned by the client (Alice), W denotes the projection matrix, and Y_i is the projected data in the space spanned by columns of matrix W.

[2] D^+ indicates the pseudo-inverse of the matrix D.

Table 1: GC Computation and Communication Costs for realization of a DNN model.

Computation and Communication Costs
$T_{comp} = \beta_{mult}\sum_{l=1}^{n_l-1} n^{(l)}n^{(l+1)} + \beta_{add}\sum_{l=2}^{n_l} n^{(l)} + \beta_{act}\sum_{l=2}^{n_l} n^{(l)}$ $\beta_{opr} = \frac{N_{opr}^{XOR}\times C_{opr}^{XOR} + N_{opr}^{Non\_XOR}\times C_{opr}^{Non\_XOR}}{f_{CPU}}$ n_l: *total number of DL layers* β_{mult}: *computational cost of a multiply operation in GC protocol* β_{add}: *computational cost of an add operation in GC protocol* β_{act}: *computational cost of a non-linearity operation in GC protocol* N^{XOR}: *number of XOR gates* N^{Non_XOR}: *number of non_XOR gates* C^{XOR}: *garbling/evaluating cost of a XOR gate* C^{Non_XOR}: *garbling/evaluating cost of a Non_XOR gate* f_{CPU}: *CPU clock frequency*
$T_{comm} = \frac{\alpha_{mult}\sum_{l=1}^{n_l-1} n^{(l)}n^{(l+1)} + \alpha_{add}\sum_{l=2}^{n_l} n^{(l)} + \alpha_{act}\sum_{l=2}^{n_l} n^{(l)}}{BW_{net}}$ $\alpha_{opr} = N_{opr}^{Non\_XOR}\times 2\times N_{bits}$ BW_{net}: *operational communication bandwidth* α_{mult}: *communication cost of a multiply operation in GC protocol* α_{add}: *communication cost of an add operation in GC protocol* α_{act}: *communication cost of a non-linearity operation in GC protocol* N_{bits}: *GC security parameter*

3.2.2 DL Network Pre-processing

Recent theoretical and empirical advances in DL has demonstrated the importance of sparsity in training DL models, e.g., [17]. Sparsity inducing techniques such as rectifying non-linearities and $\mathcal{L}_1$ penalty are key techniques used to boost the accuracy in training neural networks with millions of parameters.

To eliminate the unnecessary garbling/evaluation of non-contributing neurons in a DL model, we suggest pruning the underlying DL network prior to netlist generation for the GC protocol. In our DL network pre-processing, the connections with a weight below a certain threshold are removed from the network. The condensed network is re-trained as suggested in [17] to retrieve the accuracy of the initial DL model. DeepSecure network pre-processing step is a one-time off-line process performed by the server. Our approach is built upon the fact that DL models are

usually over-parameterized and can be effectively represented with a sparse structure without a noticeable drop in the accuracy.

Note that using conventional GC protocol, it is not feasible to skip the multiplication/addition in evaluating a particular neuron in a DL model. Our network pre-processing cuts out the non-contributing connections/neurons per layer of a DL network. It, in turn, enables using the sparse nature of DL models to significantly reduce the computation and communication workload of executing the GC protocol. The off-line step 1 indicated in Figure 1 corresponds to both data pre-processing and neural network pruning.

3.3 GC-Optimized Circuit Components Library

As we explained earlier, the GC protocol requires the function of interest being represented as a Boolean circuit. Following the Free-XOR optimization [10], the XOR gates are almost free of cost and the garbled table needs to be generated and transferred only for the non-XOR gates. Therefore, to optimize the computation and communication costs, one needs to minimize the number of non-XOR gates. We leverage the industrial synthesis tool to optimize the resulting netlist by setting the area overhead for XOR gates to zero and for all the other non-XOR gates to one. We design optimized fundamental blocks e.g., multiplexer (MUX), comparator (CMP), adder (ADD), and multiplier (MULT) such that they incur the least possible non-XOR gates. We add our optimized blocks to the library of the synthesis tool.

Table 2: Number of XOR and non-XOR gates for each element of DL networks.

Name	#XOR	#Non-XOR
$\text{Tanh}_{\text{CORDIC}}$	8415	3900
$\text{Sigmoid}_{\text{CORDIC}}$	8447	3932
$\text{Softmax}_{\text{n}}$	$(n-1)\cdot 48$	$(n-1)\cdot 32$
ReLu	30	15
ADD	16	16
MULT	381	212
DIV	545	361
$A_{1\times m}\cdot B_{m\times n}$	$397\cdot m\cdot n-16\cdot n$	$228\cdot m\cdot n-16\cdot n$

Our custom synthesis library includes the GC-optimized realization of all the necessary computation modules in a neural network. The results of our synthesized circuits in terms of the number of XOR and non-XOR are summarized in Table 2. To compute DL non-linearity functions, we evaluate COordinate Rotation DIgital Computer (CORDIC) circuit. Each iteration of computing CORDIC improves the final accuracy by one bit. For instance, in order to achieve 12 bit accuracy, we need to iteratively evaluate the circuit 12 times. To operate CORDIC in hyperbolic mode, one needs to evaluate iterations $(3\times i+1)$ twice, which in turn, results in an overall 14 iterations per instance computation. CORDIC outputs Cosine-Hyperbolic (Cosh) and Sine-Hyperbolic (Sinh). The synthesized result provided in Table 2 shows the total number of gates for 14 iterations of evaluation plus one DIV operation for $Tanh_{\text{CORDIC}}$ with an additional two ADD operations for Sigmoid computation.

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). As such, we use optimized CMP and MUX blocks to implement Softmax in DeepSecure framework.

3.4 Security Proof

In this section, we provide a comprehensive security proof of DeepSecure in the Honest-but-Curious (HbC) adversary model. Our core secure function evaluation engine is the GC protocol. GC is proven to be secure in HbC adversary model [18]; thereby, any input from either client or server(s) to GC will be kept private during the protocol execution. The Garbled circuit optimization techniques (Section 3.3) that we leverage in DeepSecure to reduce the number of non-XOR gates of circuit components do not affect the security of our framework. This is because the security of the GC protocol is independent of the topology of the underlying Boolean circuit. As such, we only need to provide the security proof of the three modifications that are performed outside of the GC protocol:

(i) Data Pre-processing. In step one of off-line processing depicted in Figure 1, the projection matrix (W) is computed and released publicly. Projection Matrix (W) reveals nothing but the subspace of dictionary matrix (D) from which the matrix D cannot be reconstructed. **Proof:** Let $D_{m\times l}=U_{m\times r}\times\Sigma_{r\times r}\times V^T_{r\times l}$ denote the Singular Value Decomposition (SVD) of the dictionary matrix D, where r is the rank of D ($r = min(m,l)$). As such,

$$\begin{aligned} W &= DD^{+} \\ &= D(D^TD)^{-1}D^T \\ &= U\Sigma V^T(V\Sigma^TU^TU\Sigma V^T)^{-1}V\Sigma^TU^T \\ &= U\Sigma(V^TV^{T^{-1}})(\Sigma^TI_r\Sigma)^{-1}(V^{-1}V)\Sigma^TU^T \\ &= U\Sigma I_r(\Sigma^T\Sigma)^{-1}I_r\Sigma^TU^T \\ &= UU^T, \end{aligned} \tag{2}$$

where I_r indicates the identity matrix of size $r\times r$. As such, the projection matrix W does not reveal any information regarding the actual values of the dictionary matrix D but the subspace spanned by its column space U (a.k.a., *left-singular vectors*). Note that for a given set of left-singular vectors U, there exist infinite possible data matrices that reside in the same column space. As such, the dictionary matrix D cannot be reconstructed without having access to corresponding right-singular vectors V and the singular value set Σ. If revealing the subspace spanned by U is not tolerable by the server, this pre-processing step can be skipped.

(ii) DL Network Pre-processing. In this pre-processing stage, the inherent sparsity of the neural network is utilized to produce a new model which requires less computation. In order to avoid garbling/evaluating unnecessary components in the Boolean circuit, the server needs to modify the netlist used in the GC protocol. Therefore, the sparsity map of the network is considered as a public knowledge and will be revealed to the Garbler as well (Garbler needs to garble the circuit based on the netlist). However, the sparsity map only contains information regarding which part of the network does not contribute to the output and reveals nothing about the private network parameters. Just like the data pre-processing step, if revealing the sparsity map is not acceptable by the server (DL model owner), this step can be skipped.

4 EVALUATIONS

We use Synopsys Design Compiler 2010.03-SP4 to generate the Boolean circuits. The timing analysis is performed by two threads on an Intel Core i7-2600 CPU working at $3.4GHz$ with $12GB$ RAM and Ubuntu 14.04 operating system. In all of the experiments, the GC security parameter is set to 128-bit. In our target DL setting, DeepSecure achieves an effective throughput of $2.56M$ and $5.11M$ gates per second for non-XOR and XOR gates, respectively.

Table 3 details DeepSecure performance in the realization of four different DL benchmarks without including the data and DL network pre-processing. Our benchmarks include both DNN and CNN models for analyzing visual, audio, and smart-sensing datasets.

Table 3: Number of XOR and non-XOR gates, communication, computation time, and overall execution time for our benchmarks without involving the data and DL network pre-processing.

Name	Data	Network Architecture	#XOR	#Non-XOR	Comm. (MB)	Comp. (s)	Execution (s)
Benchmark 1	MNIST [19]	28×28-5C2-ReLu-100FC-ReLu-10FC-Softmax	4.31E7	2.47E7	7.91E2	1.98	9.67
Benchmark 2	MNIST [19]	28×28-300FC-Sigmoid-100FC-Sigmoid-10FC-Softmax	1.09E8	6.23E7	1.99E3	4.99	24.37
Benchmark 3	Audio [20]	617-50FC-Tanh-26FC-Softmax	1.32E7	7.54E6	2.41E2	0.60	2.95
Benchmark 4	Smart-sensing [21]	5625-2000FC-Tanh-500FC-Tanh-19FC-Softmax	4.89E9	2.81E9	8.98E4	224.50	1098.3

Table 4: Number of XOR and non-XOR gates, communication, computation time, and overall execution time for our benchmarks after considering the pre-processing steps. The last column of the table denotes the improvement achieved as a result of applying our pre-processing methodology.

Name	Data and Network Compaction	#XOR	#Non-XOR	Comm. (MB)	Comp. (s)	Execution (s)	Improvement
Benchmark 1	9-fold	4.81E6	2.76E6	8.82E1	0.22	1.08	8.95×
Benchmark 2	12-fold	1.21E7	6.57E6	2.10E2	0.54	2.57	9.48×
Benchmark 3	6-fold	2.51E6	1.40E6	4.47E1	0.11	0.56	5.27×
Benchmark 4	120-fold	6.28E7	3.39E7	1.08E3	2.78	13.26	82.83×

The topology of each benchmark is outlined in the third column of the Table 3. For instance, the first benchmark is a five-layer CNN model to classify MNIST data including: (i) a convolutional layer with a kernel of size 5×5, a stride of $(2,\ 2)$, and a map-count of 5 (indicated as layer $5C2$ in Table 3). 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 (indicated as layer $100FC$ in Table 3). (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. Same notation is used to describe the architecture of other DL benchmarks.

DeepSecure Pre-processing Effect. Table 4 shows DeepSecure performance for each benchmark after including the data and DL network pre-processing. Our pre-processing customization is an arbitrary step that can be used to minimize the number of required XOR and non-XOR gates for the realization of a particular DL model. As illustrated, our pre-processing approach reduces the execution time of GC protocol by up to 82-fold without any drop in the underlying DL accuracy.

Comparison with Prior Art Framework. Table 5 details the computation and communication overhead per sample in DeepSecure framework compared with the prior art privacy-preserving DL system [5]. Our result shows more than 58-fold improvement in terms of overall execution time per sample even without considering the pre-processing steps. For instance, it takes 570.11 seconds to run a single instance on the pertinent MNIST network using [5] while DeepSecure reduces this time to 9.67 seconds with no data and network pre-processing. Our data and DL network pre-processing further reduces the processing time per sample to only 1.08 seconds with no drop in the target accuracy. Authors in [5] have only reported one benchmark (which we used as *benchmark 1*) for proof-of-concept evaluation. As such, we did not include the comparison for the other three benchmarks used in this paper.

Table 5: Communication and computation overhead per sample in DeepSecure vs. CryptoNet [5] for benchmark 1.

Framework	Comm.	Comp. (s)	Execution (s)	Improvement
DeepSecure without pre-processing	791MB	1.98	9.67	**58.96** ×
DeepSecure with pre-processing	88.2MB	0.22	1.08	**527.88**×
CryptoNets	74KB	570.11	570.11	-

Figure 2 shows the expected processing time as a function of data batch size from the client's point of view. The reported runtime for CryptoNets corresponds to implementing benchmark 1 using 5-10 bit precision on a Xeon E5-1620 CPU running at 3.5GHz as presented in [5]. Whereas, DeepSecure is prototyped using 16 bit number representation on an intel Core-i7 processor that has a slightly less computing power compared to the Xeon processor [22].

As illustrated in Figure 2, DeepSecure's computational cost scales linearly with respect to the number of samples. As such, DeepSecure is particularly ideal for scenarios in which *distributed clients stream* small batches of data (e.g., $N_{client} \leq 2590$) and send them to the server to find the corresponding inference label with *minimal delay*. However, CryptoNet is better-suited for settings where one client has a large batch of data (e.g., $N_{client} = 8192$) to process at once. This is because CryptoNets incurs a constant computational cost up to a certain number of samples depending on the choice of the polynomial degree. To mitigate the cost, authors in [5] suggest processing data in batches as opposed to individual samples using scalar encoding. The data batch size, in turn, is dictated by the polynomial degree used in the realization of a particular DL model. Therefore, to acquire a higher security level one might need to use larger data batch sizes in the CryptoNet framework.

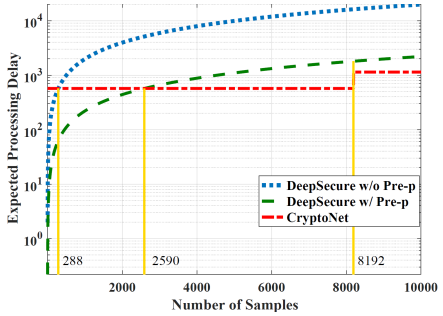

Figure 2: Expected processing time from client's point of view as a function of data batch size. In this figure, the y axis is illustrated in logarithmic scale.

5 RELATED WORK

Authors in [23] 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 [23]: (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, [24] and [25] enhance the protocol initially proposed in [23] to obscure the weights. However, even these works [24, 25] 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 DeepSecure.

Perhaps the closest work to DeepSecure is [5] in which homomorphic encryption is used as the primary tool for privacy-preserving DL computation. Unlike DeepSecure, the inherent noise in HE yields a trade-off between privacy and accuracy in evaluating the DL model which in turn translates to a lower accuracy level for obtaining a higher degree of privacy. In addition, the relatively high computation overhead of HE bounds the applicability of such approach to the use of low-degree polynomials and limited-precision numbers (e.g., 5-10 bits). SecureML [6] is another secure DL framework that proposes to use additive secret-sharing and the GC protocol. Similar to CryptoNets [5], they approximate the non-linear activation functions with polynomials. The MiniONN framework [26] is also based on additive secret-sharing and GC but has lower latency compared to SecureML approach.

A number of earlier works have shown the usability of data projection and sparsity regularization techniques to facilitate feature extraction and accelerate the execution of particular DL models [14, 17, 27]. These set of works have been mainly focused on the functionality of DL models in terms of the accuracy and physical performance (e.g., energy consumption, memory footprint, etc.) with no attention to the data privacy. To the best of our knowledge, DeepSecure is the first framework that introduces, implements, and automates the idea of data and DL network transformation as a way to minimize the number of required non-XOR gates for the privacy-preserving realization of DL models using GC.

6 CONCLUSION

We present DeepSecure, a novel practical and provably-secure DL framework that enables distributed clients and cloud servers, jointly evaluate a DL network on their private data. DeepSecure leverages automated design, efficient logic synthesis tools, and optimization methodologies to provide scalable realization of functions required for DL evaluation optimized for Yao's GC protocol. Our GC-optimized realization of hierarchical non-linear DL models demonstrates up to 58 times higher throughput per sample compared with the prior art privacy-preserving DL solution. We further propose a set of data and DL network transformation techniques as a pre-processing step to explicitly optimize the computation and communication overhead of GC protocol in the context of deep learning. Proof-of-concept evaluations using different DL benchmarks shows up to two orders-of-magnitude additional improvements achieved as a result of our pre-processing methodology.

ACKNOWLEDGMENT

This work was supported in parts by the ONR (N00014-17-1-2500) and NSF (CNS-1619261) grants.

REFERENCES

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553), 2015.
[2] Nicola Jones et al. The learning machines. *Nature*, 505(7482):146–148, 2014.
[3] Jeremy Kirk. Ibm join forces to build a brain-like computer. http://www.pcworld.com/article/2051501/universities-join-ibm-in-cognitive-computing-researchproject.html, 2016.
[4] Amir Efrati. How "deep learning" works at apple, beyond. https://www.theinformation.com/How-Deep-Learning-Works-at-Apple-Beyond, 2017.
[5] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 201–210, 2016.
[6] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. *IACR Cryptology ePrint Archive*, 2017:396, 2017.
[7] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy*, pages 411–428. IEEE, 2015.
[8] Moni Naor and Benny Pinkas. Computationally secure oblivious transfer. *Journal of Cryptology*, 18(1):1–35, 2005.
[9] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.
[10] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.
[11] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.
[12] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 220–250. Springer, 2015.
[13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 478–492. IEEE, 2013.
[14] Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. Deep3: Leveraging three levels of parallelism for efficient deep learning. In *Proceedings of the 54rd Annual Design Automation Conference*. ACM, 2017.
[15] Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. Delight: Adding energy dimension to deep neural networks. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pages 112–117. ACM, 2016.
[16] Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. Tinydl: Just-in-time deep learning solution for constrained embedded systems. In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pages 1–4. IEEE, 2017.
[17] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.
[18] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796. ACM, 2012.
[19] Yann LeCun, Corinna Cortes, and Christopher Burges. Mnist dataset. http://yann.lecun.com/exdb/mnist/, 2017.
[20] UCI machine learning repository. https://archive.ics.uci.edu/ml/datasets/isolet, 2017.
[21] UCI machine learning repository. https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities, 2017.
[22] Intel Processors. http://www.velocitymicro.com/blog/xeon-vs-i7i5-whats-difference/, 2017.
[23] Mauro Barni, Claudio Orlandi, and Alessandro Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia and security*, pages 146–151. ACM, 2006.
[24] Claudio Orlandi, Alessandro Piva, and Mauro Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security*, 2007:18, 2007.
[25] Alessandro Piva, Claudio Orlandi, M Caini, Tiziano Bianchi, and Mauro Barni. Enhancing privacy in remote data classification. In *IFIP International Information Security Conference*, pages 33–46. Springer, 2008.
[26] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. Oblivious neural network predictions via MiniONN transformations. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.
[27] Mohammad Samragh, Mohammad Ghasemzadeh, and Farinaz Koushanfar. Customizing neural networks for efficient fpga implementation. In *IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017.