Privacy-Preserving Deep Learning Based on Multiparty Secure Computation: A Survey

Qiao Zhang, Chunsheng Xin, *Senior Member, IEEE*, and Hongyi Wu, *Fellow, IEEE*

***Abstract*—Deep learning (DL) has demonstrated superior success in various of applications, such as image classification, speech recognition, and anomalous detection. The unprecedented performance gain of DL largely depends on tremendous training data, high-performance computation resources, and well-designed model structures. However, privacy concerns raise from such necessities. First, as the training data are usually distributed among multiple parties, directly exposing and collecting such large amount of data could violate the laws especially for private information, such as personal identities, medical records, and financial profiles. Second, locally deploying advantageous computation resources is costly for individual party having partial data. Third, direct release of well-trained model parameters threatens the information about training data or the intellectual property of model owners. Therefore, individual party prefers outsourcing computation (data) in a secure way to powerful cloud servers such as Microsoft Azure, and how to enable the cloud servers to perform DL algorithms without revealing data owners' private information and model owners' valuable parameters is emerging as an urgent task, which is termed as privacy-preserving (outsourcing) DL. In this article, we review the state-of-the-art researches in privacy-preserving DL based on multiparty secure computation with data encryption and summarize these techniques in both training phase and inference phase. Specifically, we categorize the techniques with respect to the linear and nonlinear computations, which are the two basic building blocks in DL. Following a comprehensive overview of each research scheme, we present primary technical hurdles needed to be addressed and discuss several promising directions for future research.**

***Index Terms*—Deep learning (DL), linear and nonlinear computations, privacy preserving.**

I. Introduction

LARGE-VOLUME data are generated in our daily life due to the rapid evolution and utilization of the Internet of Things (IoT) [1], [2]. Compared to 0.9 ZB in 2013, the amount of usable data is estimated to be over 15 ZB by 2020, [3], which has become a driving force for the development of deep learning (DL) technologies [4]–[7]. DL has shown its unrivalled performance in many applications, such as image classification [8], [9], speech recognition [10]–[12], anomalous detection [13]–[15], and business analytics [16], [17]. The success of DL largely depends on three key ingredients, i.e., massive amount of high-quality training data, high-performance computation resources, and well-designed model structures [18], [19]. These ingredients are often owned by different entities. For instance, end users and enterprises possess enormous data, while the DL talents and computing power are mostly gathered in technology giants, such as Google, Facebook, and Microsoft. The data owners are motivated to outsource their data, along with the computationally intensive tasks, to the clouds (e.g., Microsoft Azure and Google Cloud) in order to leverage clouds' abundant resources and DL talents for developing cost-effective DL solutions.

DL includes the training phase and the inference phase, where the former utilizes the training data to produce a well-trained model, based on which the latter conducts the prediction for newly input data. For example, assume Alice owns her data, while Bob owns the cloud platform. In training phase, Alice would like to have her data processed by Bob to create a well-trained DL model. In the inference phase, Bob, the cloud server with the trained model and computational resources, performs predictions for clients, which is often known as machine learning as a service (MLaaS) [20], [21].

However, both phases require trust between the cloud servers (i.e., service providers) and clients (i.e., data owners). Due to different trust domains, privacy issues arise from the exposure to the cloud servers of private information in the outsourced data and from the intermediate data through the interactions between data owners and cloud servers. Both of them involve data breach corresponding to either the original data or model parameters. There are various instances, where the original data or model parameters should not be public [22]–[25]. As the data breach becomes a major concern, more and more governments have established regulations for protecting users' data, such as general data protection regulation (GDPR) in European Union [26], personal data protection Act (PDPA) in Singapore [27], California consumer privacy act (CCPA) [28], and health insurance portability and accountability act (HIPAA) [29] in the U.S. The cost of data breach is high. For instance, in the breach of 600 000 drivers' personal data in 2016, Uber had paid \$148 million to settle the investigation [30]; SingHealth was fined about \$750 000 by the Singapore government for a breach of PDPA data [31]; Google was fined about \$57 million for a breach of GDPR

Manuscript received October 22, 2020; revised December 26, 2020; accepted January 27, 2021. Date of publication February 11, 2021; date of current version June 23, 2021. This work was supported in part by the National Science Foundation under Grant CNS-1828593, Grant OAC-1829771, Grant EEC-1840458, and Grant CNS-1950704; in part by the Office of Naval Research under Grant N00014-20-1-2065; and in part by the Commonwealth Cyber Initiative, an Investment in the Advancement of Cyber Research and Development, Innovation and Workforce Development. For more information about CCI, visit cyberinitiative.org. *(Corresponding author: Hongyi Wu.)*

The authors are with the School of Cybersecurity and the Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, VA 23529 USA (e-mail: qzhan002@odu.edu; cxin@odu.edu; h1wu@odu.edu).

Digital Object Identifier 10.1109/JIOT.2021.3058638

data [32], which is the largest penalty as of March 18, 2020 under the European Union privacy law.

In addition to the processed (usually personal) data from clients, cloud servers that created/computed the DL models in MLaaS, also wish not to make these highly valued model parameters publicly available. By releasing the parameters of their DL models, cloud servers are worried about losing their intellectual property. Furthermore, DL models are shown to memorize information about their training data [33]. In particular, the parameters of DL models could lead to exposure of training data [34], which are considered confidential in many cases.

Privacy-preserving DL takes ethical and legal privacy concerns into account. Specifically, in privacy-preserving training, it is demanded that neither the cloud servers, nor any other involved parties, learn anything about clients' outsourced data other than its size, which we denote as *data privacy*. Meanwhile, the trained models are supposed to be, respectively, blind to data owners and cloud servers [35], which we denote as *model privacy*. In privacy-preserving MLaaS, besides the data privacy, it also needs model privacy such that neither the clients nor any other parties learn anything about the model parameters at cloud servers, other than the final predictions.

Given the importance of both data and model privacy for privacy-preserving DL, in this article, we review the state-of-the-art schemes that achieve privacy-preserving DL based on multiparty secure computation that relies on data-encryption techniques. Compared with the surveys that directly classify the schemes based on privacy-preserving primitives [36], [37], specific functions (applications) [38], or learning phases [39], [40] (i.e., training phase and inference phase), we categorize these data-encryption-based privacy-preserving techniques with respect to (w.r.t.) linear and nonlinear computations, which are the two building blocks in DL frameworks. This novel classification offers deep insights into data-encryption-based privacy-preserving techniques used in DL. While different DL frameworks are developed for different applications or functions, the general pattern that repeats linear and nonlinear computations keeps unchanged. Thus, this survey can serve as a basic reference for privacy-preserving DL via data encryption. Specifically, we first review five basic privacy-preserving primitives used in linear and nonlinear computations of DL[1] and present a comprehensive overview of the correspondingly state-of-the-art research schemes. Then, we analyze the fundamental technical hurdles needed to be addressed and discuss several promising directions for future research to close the gap between efficiency and accuracy in data-encryption-based privacy-preserving DL.

The remainder of this article is organized as follows. Section II introduces the basic structure of DL frameworks as well as the five basic primitives used in data-encryption-based privacy-preserving computation. The threat model is described in Section III. Sections IV and V, respectively, review the state-of-the-art privacy-preserving schemes via data encryption for linear and nonlinear computations in DL. Section VI gives a combinational analysis for the secure computations in DL. Several technical hurdles needed to be addressed and some promising directions for future research are presented in Section VII. Finally, Section VIII concludes this article.

[1]There is another branch of research for privacy-preserving DL with differential privacy [41]–[44] which deals with user data in plaintext and we make it orthogonal to our survey scope as we mainly focus on privacy-preserving DL with various data encryption techniques.

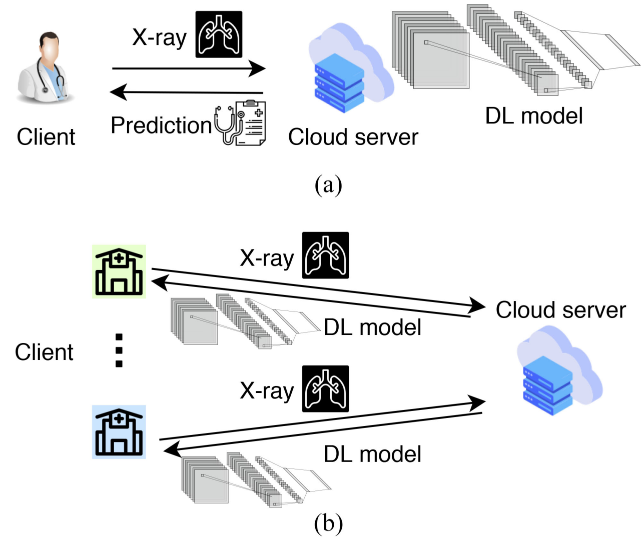


Fig. 1. Data-encryption-based privacy-preserving DL models. (a) Privacy-preserving MLaaS. (b) Privacy-preserving training.

II. Preliminaries

In this section, we give basic background about DL networks and the five basic cryptographic tools used in most privacy-preserving DL systems based on multiparty secure computation.

A. System Models

In this article, we consider two dominant privacy-preserving DL scenarios in multiparty secure computation, i.e., data-encryption-based privacy-preserving MLaaS and training. In data-encryption-based privacy-preserving MLaaS as shown in Fig. 1(a), the data owner is the client and the cloud server has a well-trained DL model. The cloud server provides the inference service based on the private data from the client. For example, an encrypted medical image (such as a chest X-ray) is sent by a doctor to the cloud server, which runs the DL model and returns the encrypted prediction to the doctor. The prediction is then decrypted by the doctor into a plaintext result to assist diagnosis and healthcare planning. As for data-encryption-based privacy-preserving training shown in Fig. 1(b), multiple clients securely outsource their partial data and jointly communicate with the cloud server to obtain a well-trained model. The trained model is supposed to be blind to individual data owner and cloud server [35]. For instance, multiple healthcare providers (such as hospitals) send their encrypted medical data (such as the chest X-ray) to the cloud server and the final model is obtained by interactions among them. The healthcare providers can utilize the more comprehensive model to improve the diagnosis accuracy while each of them is supposed to be blind to the final model.

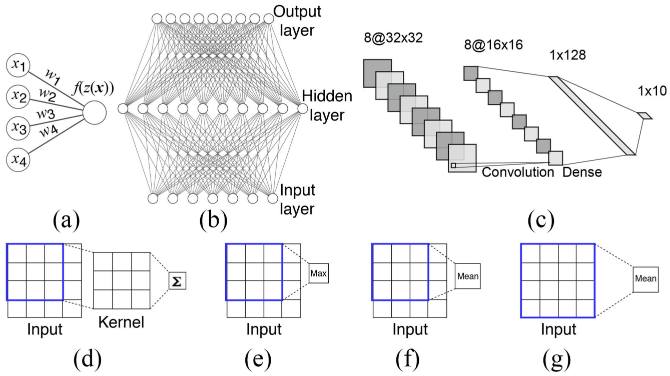


Fig. 2. Neural network structures. (a) Single neuron. (b) MLP. (c) CNN. (d) Convolution. (e) Max pooling. (f) Mean pooling. (g) Global pooling.

B. Deep Learning Models

In this article, we focus on two typical DL models [45], i.e., multilayer perceptron (MLP) and convolutional neural network (CNN). A neuron is the basic unit of a DL model, which receives signals from the connected preneurons, sequentially conducts a linear and a nonlinear transformation, and then produces an output signal that multicasts to postneurons. Each connection between two neurons has a weight value, assume the preneuron signal is $\boldsymbol{x}$, then the linear transformation is $z(\boldsymbol{x})$ and the output signal with nonlinear transformation is $f(z(\boldsymbol{x}))$. The neurons with same preneurons and postneurons form one layer and the "deep" in DL indicates the neural network with many hidden layers (i.e., the layers except the input layer that has no preneurons and the output layer that has no postneurons). We show MLP and CNN with only a few hidden layers in Fig. 2 for clarity. Meanwhile, the last a few layers in many CNNs are composed of MLP [8], [9], [18], [19]. We provide the model details as follows.

Linear Transformation: There are mainly two forms for $z(\boldsymbol{x})$ in DL models, i.e., dot product and convolution. As for dot product, it is simply matrix–vector multiplication denoted as $z(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{w}+\boldsymbol{b}$ where $\boldsymbol{w}$ is the weight matrix between two adjacent layers and $\boldsymbol{b}$ is a constant vector termed bias. Meanwhile, the process of convolution can be visualized as placing the kernel at different locations of the input data. At each location, a sum of elementwise product is computed between the kernel and corresponding input values within the kernel window [46]. For example, as shown in Fig. 2(d), by first conducting elementwise multiplication between the nine-element kernel and the nine elements of input that are within the kernel window, one convolution value is then obtained by summing the nine multiplied values. Other convolution values are calculated in a similar way with the kernel placed at different locations of input. Moreover, the convolution can be converted to dot product for efficient calculation [47]. Specifically, both of the kernel and corresponding input values within that kernel window can be, respectively, flattened as vectors w' and x', which enables matrix–vector multiplication $w'x'$.

Nonlinear Transformation: The output of linear transformation, i.e., $z(\boldsymbol{x})$, is fed into different nonlinear functions, which generally have three types in DL models, i.e., activation functions, pooling functions, derivatives of activation functions, and derivatives of pooling functions. Activation functions are applied to the linear result, i.e., dot product or convolution, in an elementwise manner. The commonly used activation functions include ReLU, $f(z) = \max\{0, z\}$, sigmoid, $f(z) = [1/(1 + e^{-z})]$, and tanh, $f(z) = [(e^{2x} - 1)/(e^{2x} + 1)]$. The output layer adopts softmax, $f(z_j) = [e^{z_j}/(\sum_j e^{z_j})]$, to normalize each linear value z_j. Meanwhile, there are generally three types of pooling functions, i.e., mean pooling, max pooling, and global pooling. The mean pooling and max pooling, respectively, take the mean and maximum of the values within the designated pooling window [see Fig. 2(e) and (f)], while the global pooling returns the mean of the whole input [19] [see Fig. 2(g)].

Forward and Backpropagation: Given the input data, the forward propagation repeats the linear transformation, i.e., the dot product or convolution, and nonlinear transformation, i.e., activation functions or pooling functions, until the last layer, where the nonlinear transformation is softmax and the output is the prediction result. In data-encryption-based privacy-preserving MLaaS, the cloud server receives encrypted data from the client and then completes the forward propagation such that the prediction result is finally returned back to the client. Meanwhile, the softmax can be ignored in data-encryption-based privacy-preserving MLaaS as it is monotone and, thus, does not affect the prediction result. In order to do training, the output of forward propagation is fed into the backpropagation to update the model parameters, i.e., the weight matrix, bias and kernels, through gradient decent, where the derivatives of activation functions and pooling functions are involved [8]. The goal of data-encryption-based privacy-preserving training is to make the process of parameter update blind to all involved parties, given that all the clients send their encrypted data to the cloud servers, which obliviously update the model parameters by multiparty interactions [35], [48].

C. Privacy-Preserving Tools

1) Homomorphic Encryption: Homomorphic encryption (HE) is a kind of data encryption mechanism that supports arithmetic operations, i.e., addition and multiplication, on encrypted data without the need of decryption [49]. Generally, let $\mathcal{M}$ and $\mathcal{E}$ denote the spaces of plaintexts and ciphertexts, respectively, an HE scheme Π = (KeyGen, Enc, Dec, Eval) is a quadruple of algorithms that proceeds as follows [50].

1) *KeyGen(1^λ):* Given the security parameter λ, this algorithm outputs a public key *pk*, a public evaluation key *evk*, and a secret key *sk*.
2) *Enc$_{pk}$($\boldsymbol{m}$):* Using the public key *pk*, the encryption algorithm encrypts a message $\boldsymbol{m} \in \mathcal{M}$ into a ciphertext $ct \in \mathcal{E}$.
3) *Dec$_{sk}$(ct):* For the secret key *sk* and a ciphertext *ct*, the decryption algorithm returns a message $\boldsymbol{m} \in \mathcal{M}$.
4) *Eval$_{evk}$($f; ct_1, \ldots, ct_k$):* Using the evaluation key *evk*, for a circuit $f : \mathcal{M}^k \rightarrow \mathcal{M}$, which consists of addition and multiplication gates, and a tuple of ciphertexts $(ct_1, \ldots, ct_k)$, the evaluation algorithm outputs a ciphertext $ct_0 \in \mathcal{E}$.

An HE scheme Π is called correct if the following statements are satisfied with an overwhelming probability.

TABLE I
HE LIBRARIES

Libraries	Language	HE schemes
SEAL [62]	C++	BFV & CKKS
HElib [63]	C++	BGV [64] & CKKS
TFHE [65]	C++	GSW [66]
HEANN [67]	C++	CKKS
PALISADE [68]	C++	BGV [64] & BFV & StSt [69]
$\wedge \circ \lambda$ [70]	Haskell	BGV [64]
Cingulata [71]	C++	BFV
FV-NFLlib [72]	C++	FV [56]
Lattigo [73]	Go	BFV
NuFHE [74]	Python	GSW [66]
Marble [75]	C++	BGV [64] & FV [56]

Protocol 1 1-Out-of-2 OT

1: The sender has a set of two strings, m_0 and m_1.
2: The chooser selects $b \in \{0, 1\}$ corresponding to whether she wishes to learn m_0 or m_1.
3: The chooser generates a public / private key pair (k^{pub}, k^{pri}), along with a second value $k^{\perp}$ that is indistinguishable from a public key, but for which chooser has no corresponding private key to decrypt with.
4: The chooser advertises values as public keys (k_0^{pub}, k_1^{pub}) and sets $k_b^{pub} = k^{pub}$ and $k_{1-b}^{pub} = k^{\perp}$.
5: The sender encrypts two messages $c_0 = E_{k_0^{pub}}(m_0)$ and $c_1 = E_{k_1^{pub}}(m_1)$ and sends c_0 and c_1 to chooser.
6: The chooser decrypts the chosen string by $m_i = D_{k^{pri}}(c_i)$.

1) $\mathrm{Dec}_{sk}(ct) = \boldsymbol{m}$ for any $\boldsymbol{m} \in \mathcal{M}$ and $ct \leftarrow \mathrm{Enc}_{pk}(\boldsymbol{m})$.
2) $\mathrm{Dec}_{sk}(ct_0) = f(\boldsymbol{m}_1, \ldots, \boldsymbol{m}_k)$ with an overwhelming probability if $ct_0 \leftarrow \mathrm{Eval}_{evk}(f; ct_1, \ldots, ct_k)$ for an arithmetic circuit $f: \mathcal{M}^k \rightarrow \mathcal{M}$ and for some ciphertexts $ct_1, \ldots, ct_k \in \mathcal{E}$ such that $\mathrm{Dec}_{sk}(ct_i) = \boldsymbol{m}_i$.

The first set of HE schemes, such as Paillier [51] and ElGamal [52], are partially homomorphic encryption (PHE) in that they support exactly one homomorphic operation: addition or multiplication. The Boneh–Goh–Nissim (BGN) cryptosystem [53] is the first scheme to support arbitrary number of additions and a single multiplication, which is called somewhat homomorphic encryption (SHE). Gentry's work [54] presents the first fully homomorphic encryption (FHE) scheme in that it supports the evaluation of an arbitrary number of both additions and multiplications. Recently optimized HE schemes, e.g., Brakerski/Fan–Vercauteren (BFV) scheme [55], [56] and Cheon–Kim–Kim–Song (CKKS) scheme [57], apply optimizations such as the Smart-Vercauteren batching technique [58] to pack multiple plaintexts into one ciphertext, so that computations can be performed in a single instruction multiple data (SIMD) manner. Such schemes belong to leveled homomorphic encryption (LHE) in that the amount of both additions and multiplications is limited, as the noise in a ciphertext is accumulated after each HE operation. Furthermore, the multikey homomorphic encryption (MKHE) [59] enables $ct_0 = \mathrm{Eval}_{\{evk_i\}_{i=1}^{k}}(f; ct_1, \ldots, ct_k)$ on a set of evaluation keys, $\{evk_i\}$, from k parties while the ciphertext ct_i is encrypted by party i's public key. The decryption of ct_0 needs collaboration from all the k parties such that ct_0 remains partially decrypted to each party and the fully decrypted ciphertext is obtained by merging these partially decrypted ct_0s from all involved parties. Finally, many libraries [60], [61] are available for HE-related applications and implementations, as shown in Table I.

Remark: HE supports addition and multiplication over encrypted ciphertexts, which indicates its linearity for secure computation. As for efficiency, the batching-based schemes, such as BFV and CKKS, are more advantageous to process large-size data than the nonbatching-based schemes, e.g., Paillier, that deal with single-value ciphertexts [76]. Meanwhile, MKHE requires all participants to be involved to complete the evaluation functions as well as decryption, which limits its scalability to massive clients [59].

2) Garbled Circuits: In Yao's garbled circuits (GC) [77], a garbled version of a Boolean circuit is interactively evaluated by two parties. One party, called garbler, creates the garbled circuit and encodes its inputs based on the garbled circuit. The other party, called evaluator, receives the garbled circuit and obtains encodings of its inputs via oblivious transfer (OT) [78]. The evaluator then evaluates the circuit gate by gate to finally compute the encoding of the output, which is decoded by garbler. Formally, A garbling scheme is a tuple of algorithms GC = (Garble, Eval) with the following syntax [79].

1) *GC.Garble(C)*$\rightarrow$*(*$\tilde{C}$*, {label*$_{i,0}$*, label*$_{i,1}$*}):* On inputting a boolean circuit C, Garble outputs a garbled circuit $\tilde{C}$ and a set of labels $\{\mathrm{label}_{i,0}, \mathrm{label}_{i,1}\}$. Here $\mathrm{label}_{i,b}$ represents assigning the value $b \in \{0, 1\}$ to the ith input label.
2) *GC.Eval(*$\tilde{C}$*,{label*$_{i,x_i}$*}*$_{i\in[n]}$*)*$\rightarrow$ *y:* On inputting a garbled circuit $\tilde{C}$ and labels $\{\mathrm{label}_{i,x_i}\}_{i\in[n]}$ corresponding to an input $x = \{x_i\} \in \{0, 1\}^n$, Eval outputs a string $y = C(x)$.

The GC tuple must be complete: the output of Eval must equal $C(x)$. Second, it must be private: given $\tilde{C}$ and $\{\mathrm{label}_{i,x_i}\}$, the evaluator should not learn anything about C or x except the size of $|C|$ (i.e., the number of gates in C denoted by $1^{|C|}$) and the output $C(x)$.

Yao's protocol has a constant number of communication rounds and the main overhead stems from the total number of AND gates in the circuit, as XOR gates can be evaluated for free [80]. Other state-of-the-art GC optimizations are point-and-permute [81], fixed-key AES garbling [82], and half-gates [83].

Remark: Theoretically, GC supports linear and nonlinear computations. Nevertheless, it may lead to high computation and communication overheads for a DL model since there are usually thousands of input values in each layer. Each of these values costs, for example, one comparison circuit to get the ReLU result [76].

3) Oblivious Transfer: OT is a cryptographic primitive which involves two parties, i.e., a chooser and a sender [84]. Specifically, the sender holds two messages, m_0 and m_1, while the chooser holds a Boolean value $b \in \{0, 1\}$. At the end of the OT execution, the chooser learns m_b, i.e., the sender's message corresponding to chooser's Boolean value. Here, an OT protocol guarantees that: 1) the chooser learns nothing about m_{1-b}, and 2) the sender learns nothing about b. Protocol 1 shows a basic OT scheme [85].

In fact, any function can be evaluated securely using only an OT protocol [86] and OT is a crucial component of Yao's GC.

Furthermore, a remarkable advancement in the practicality of OT protocols was the discovery of OT extension [87].

Remark: OT protocols usually have a considerable communication complexity, i.e., the amount of exchanged data is highly related to the bit length of input data. Especially, the communication overhead alone can introduce significant delays over wide-area networks (WAN).

4) Secret Sharing: There are mainly three types of secret sharing (SS), i.e., arithmetic sharing, Shamir's sharing [88] and the Goldreich–Micali–Wigderson (GMW) protocol [86], [89]. Arithmetic sharing uses modular addition to share arithmetic values in $\mathbb{Z}_{2^l}$ for a bit length l. The first party, who owns secret x, generates a random number $x_1 \in \mathbb{Z}_{2^l}$ and subtracts it from the secret to get $x_2 = x - x_1 \bmod 2^l$. x_1 is the first party's share, while x_2 sent to the second party is her share. Addition can be done for free, while multiplication requires: 1) one round of interaction and 2) arithmetic multiplication triples that can be efficiently precomputed using OTs [90]. As for Shamir's sharing, a $(t-1)$-degree polynomial within a moduli is constructed, where the secret is embedded as the constant term. Then, each of the total n parties is assigned a point for the polynomial such that t out of n parties can recover the coefficients of the adopted polynomial, by solving the equation group and, thus, obtain the secret. Concretely, a $(t-1)$-degree polynomial is in the form of $s(x) = s_0 + s_1 x + s_2 x^2 + \cdots + s_{t-1} x^{t-1}$ where s_0 is the secret, $s_0 = s(0)$, which requires at least t pairs of $(x, s(x))$ to obtain that value. In the GMW protocol, XOR-based SS is used to hide private values. Specifically, a Boolean circuit is interactively evaluated on the XOR-secret-shared data gate by gate using OTs. Similar to Yao's GC protocol, XOR gates can be freely evaluated. The evaluation of AND gates requires: 1) one round of communication and 2) multiplication triples that can be precomputed using OTs [90]. Thus, the complexity of GMW protocol results from: 1) the total number of AND gates in the circuit and 2) the multiplicative depth of the circuit, i.e., the maximum number of AND gates on the critical path from any input to any output in the Boolean circuit.

Remark: In arithmetic sharing, additions and multiplications are very efficient. Arithmetic sharing and GMW strongly depend on low-depth circuits and a low network latency to perform well. However, they do not require symmetric cryptographic operations in the online phase, which makes them better suited for weaker devices than Yao's protocol [91], [92]. Shamir's sharing requires the secret distributor not to collude with any of the involved parties that hold the shares.

The GC, OT, and SS belong to the scope of secure multiparty computation (SMPC). A set of general-purpose compilers are available to facilitate real-world SMPC implementations. These tools provide high-level abstractions to describe arbitrary functions and execute SMPC protocols, i.e., GC, OT, and SS [93], [94]. Table II shows the widely used SMPC programming tools.

TABLE II
SMPC PROGRAMMING TOOLS

Libraries	Language	SMPC schemes
SoK [95]	C++	GC & OT & SS
EMP-toolkit [96]	C++	GC & OT
Obliv-C [97]	C/OCaml	GC
ObliVM [98]	JAVA	GC
TinyGarble [99]	C/C++	GC
SCALE-MAMBA [100]	Verilog	SS
Wysteria [101]	Wy	SS
Sharemind [102]	SECREC	SS
PICCO [103]	C/C++	SS
ABY [104]	C++	GC & OT & SS
ABY3 [105]	C++	GC & OT & SS
Frigate [106]	C	GC
CBMC-GC [107]	C	GC & OT & SS
HyCC [108]	C	GC & OT & SS
TASTY [109]	Python	GC
MP-SPDZ [110]	Python	GC & SS

5) Trusted Execution Environment: Secure computation between nonmutually trusting parties can be achieved with hardware-assisted security, by executing the code in a trusted execution environment (TEE). For example, the Intel Software Guard Extension (Intel SGX) [111], [112], a set of new instructions and memory access changes added to the Intel Architecture, is a widely adopted TEE implementation and is deployed in Intel's recent CPUs. SGX enables the implementation of secure two-party computation [113]–[115]. Specifically, SGX provides the capability to protect specified areas of an application from outside access. The area is called an enclave and hardware provides confidentiality and integrity for the specified area. SGX enables software developers to build trusted modules inside an application to protect secrets. For example, a software developer specifies the contents of an enclave and a relying party can confirm that the area is instantiated correctly on a remote machine. To enforce isolation, hardware performs extra memory checks such that only enclave code can access enclave data. When the external software attempts to access the enclave, hardware will abort the accesses. In this manner, SGX provides a trusted place to stand for application developers. Therefore, the security of TEE relies on the hardware architecture that isolates the code and data in TEE from external environments. A set of opensourced TEE libraries is available [116].

Remark: TEE enables efficient computation since plaintext data and code can be directly loaded and executed. On the other hand, data owners need to admit the plaintext computation over their plaintext data in the third-party environment, where security guarantee totally relies on that third-party vendor. This poses additional security concerns to data owners as there have been many attacks targeting at TEE-based systems, such as the famous side-channel attacks [117]–[119].

III. THREAT MODELS

There are mainly two adversarial behaviors considered in data-encryption-based privacy-preserving DL: 1) a participant is defined to be semihonest (SH) or passive if he executes the predefined protocols correctly, but tries to learn as much private information as possible by analyzing the messages exchanged during the protocol execution and 2) he is defined to be malicious (MA) or active if he arbitrarily deviates from the protocol specifications. Concretely, in data-encryption-based privacy-preserving DL with SH participants, the clients (data owners) and cloud servers are assumed to follow all protocols during the entire learning process, while individual client tries to learn the model parameters or private data of other clients (in training), and cloud servers try to infer clients' input data, or the model parameters (in training). As for data-encryption-based privacy-preserving DL with MA participants, the individual client arbitrarily deviates from the protocols by sending cloud servers wrong input or intermediate data, while cloud servers deviate from the protocols by applying wrong model parameters (in MLaaS) or by sending clients wrong intermediate data. The two adversarial behaviors result in either misleading prediction in MLaaS or meaningless model parameters in training. One should prove the security against either SH or MA adversaries in the designed privacy-preserving DL framework.

The security proofs against SH adversaries are given in Yao's protocol [77], [120] and the GMW protocol [86], [89], while the security proofs against MA adversaries are based on zero-knowledge protocols [89]. The SH threat model is weaker than the MA counterpart, but it allows to build highly efficient protocols and is therefore widely adopted in data-encryption-based privacy-preserving DL [121]. In this article, we mainly focus on the SH threat model.

Meanwhile, as the data-encryption-based privacy-preserving DL tackles the privacy issues during the learning process, i.e., inference or training, the black-box attacks that utilize the prediction results, e.g., model extraction [122], [123], model inversion [34], membership inference [124], detection evasion [125], [126], are out of the scope of this article.

TABLE III
LINEAR COMPUTATION FOR PRIVACY-PRESERVING DL

Schemes	Privacy preserving primitives					Threats	
	HE	GC	OT	SS	TEE	Se	Ma
Liu [48]	○	○	○	●	○	○	●
Falcon [121]	○	○	○	●	○	○	●
Privedge [127]	○	○	○	●	○	●	○
Secureml [128]	○	○	○	●	○	●	○
Swift [129]	○	○	○	●	○	●	○
Chameleon [130]	○	○	○	●	○	●	○
Minionn [131]	○	○	○	●	○	●	○
Securenn [132]	○	○	○	●	○	●	○
Trident [133]	○	○	○	●	○	●	○
DElphi [79]	○	○	○	●	○	●	○
Huang [134]	○	○	○	●	○	●	○
Leia [135]	○	○	○	●	○	●	○
Quotient [84]	○	○	●	○	○	●	○
Soteria [136]	○	●	○	○	○	●	○
Deepsecure [137]	○	●	○	○	○	●	○
Xonn [138]	○	●	○	○	○	●	○
Zhu [139]	●	○	○	○	○	●	○
Bayhenn [140]	●	○	○	○	○	●	○
Cheetah [141]	●	○	○	○	○	●	○
Cryptonets [142]	●	○	○	○	○	●	○
Faster [143]	●	○	○	○	○	●	○
Gelunet [144]	●	○	○	○	○	●	○
E2dm [50]	●	○	○	○	○	●	○
Falcon [145]	●	○	○	○	○	●	○
Gazelle [76]	●	○	○	○	○	●	○
Cheetah [146]	●	○	○	○	○	●	○
Homopai [147]	●	○	○	○	○	●	○
Autoprivacy [148]	●	○	○	○	○	●	○
Ensei [149]	●	○	○	○	○	●	○
Spindle [150]	●	○	○	○	○	●	○
Badawi [151]	●	○	○	○	○	●	○
Xu [152]	●	○	○	○	○	●	○
Helen [153]	●	○	○	○	○	○	●
Chen [59]	●	○	○	○	○	●	○
Cryptodl [154]	●	○	○	○	○	●	○
Hervé [155]	●	○	○	○	○	●	○
Sesame [156]	○	○	○	○	●	●	○
Darknight [157]	○	○	○	○	●	●	○
Slalom [158]	○	○	○	○	●	●	○
Chiron [159]	○	○	○	○	●	○	●

"●" and "○" denote the adopted and unadopted item, respectively.

IV. Linear Computation in Data-Encryption-Based Privacy-Preserving DL

Table III shows the adopted primitives for linear computation, i.e., dot product and convolution, in state-of-the-art data-encryption-based privacy-preserving DL frameworks. The most chosen primitive is HE which is followed by SS, while other primitives, i.e., GC, OT, and TEE, are less preferred. This trend is largely due to the natural linearity of HE and SS. On one hand, HE enables efficient linear operations, e.g., the batching additions and multiplications in the recently optimized CKKS scheme. On the other hand, SS is also efficient to compute dot product based on multiplication triplet. As the addition and multiplication are more suitable for arithmetic (i.e., numberwise) computation rather that Boolean (i.e., bitwise) computation, the OT and GC are not widely adopted. Furthermore, the TEE relies on the hardware trust such that all the computations, i.e., linear computation and nonlinear computation, can be completed in plaintext. While the tradeoff is that the data owners should subscribe to the TEE module and load their private data into that environment. The detailed analysis for each adopted primitive is listed as follows.

A. HE-Based Linear Computation

The most appealing property of HE is the ability to conduct linear operations, i.e., addition and multiplication, over ciphertext without knowing the secret. As the dot product mainly involves addition and multiplication, HE can be directly applied such that the data matrix (or a data vector) is first encrypted by clients and then multiplied with the weight vector (or a weight matrix) by cloud servers [140], [142], [144], [150], [152]–[155]. Meanwhile, by considering the computation complexity of each type of operation, e.g., HE addition is faster than HE multiplication and plaintext computation is cheaper than ciphertext counterpart, the total cost can be optimized by securely selecting the cheaper operations [50], [76], [141], [146]. Moreover, each HE operation can also be optimized by applying sparse polynomial multiplication in the process of individual HE addition and HE multiplication [143]. Furthermore, as different crypto parameters result in different complexities for HE operations, the parameter selection is optimized to boost the overall efficiency [148]. Besides converting convolution to dot product for convolution computation, some schemes rely on another fact that convolution can be converted to elementwise multiplication by the Fourier transform, e.g., the fast Fourier transform [145] and discrete Fourier transform [149]. Therefore, the cost for convolution is reduced since only elementwise multiplication is involved. On the other hand, the parallel techniques are also considered to accelerate the HE computation, such as message passing interface (MPI)-based acceleration [147] and graphics processing unit (GPU)-based acceleration [151]. Finally, given different data owners use different keys to encrypt their private data, MKHE [59], [139] enables linear computation over these ciphertexts.

Remark: HE schemes are often preferred for linear computation because it can allow for "batched" parallel computations over ciphertext, which are called SIMD operations [58]. This technique is exemplified by the CryptoNets framework [142]. Furthermore, the leveled HE schemes, such as the BFV used in GAZELLE framework [76], are unable to perform many nested multiplications, a requirement for state-of-the-art DL models [19]. Generally, with the continuing optimization for HE schemes, it is promising to use HE to achieve efficient linear computation.

B. GC-Based Linear Computation

GC is based on the Boolean circuit and there are mainly two branches for GC-based linear computation in state-of-the-art schemes. The first branch is to select suitable DL models, i.e., the binary neural network (BNN) [138] or ternary neural network (TNN) [136]. In these networks, the model parameters are among -1, 0, and $+1$, which simplifies the linear computation, i.e., the dot product and convolution, into operations among XOR gates, which are cheap to GC [80]. Thus, BNN and TNN are efficiently calculated by GC techniques. On the other hand, the network performance, e.g., model accuracy, of BNN (or TNN) is not as good as the state-of-the-art DL models with floating-point parameters [19], which limits the scalability of such networks. The second branch is to optimize the process of GC itself, e.g., optimization of circuit generation [137]. Given that the cost of GC mainly comes from the nonXOR (NXOR) gates in the circuits, an optimization function is formed to minimize the needed NXOR gates and thus the resultant circuits computation is accelerated.

Remark: GC is not preferred for linear computation and we list three reasons as follows. First, the addition and multiplication are more suitable to arithmetic (i.e., numberwise) circuit rather than Boolean (i.e., bitwise) counterpart. Second, given the large number of additions and multiplications in state-of-the-art DL models [19], the Boolean circuits for such models involve a significant amount of NXOR gates, which are not advantageous to use GC. Finally, communication is another critical factor to the GC cost [160], which poses another hurdle when applying GC to state-of-the-art DL models.

C. OT-Based Linear Computation

For OT-based linear computation, the main method is to use correlated OT (COT) for the dot product [84], [128]. Specifically, COT involves two parties, i.e., a sender (data owner) and a chooser (cloud server). Given that each value of dot product is the sum of several elementwise multiplications, the shares of the dot product are, respectively, obtained by adding all the shares of elementwise multiplications at two parties. Therefore, the sender forms two messages, $m_0 \in \mathbb{Z}_{2^l}$ and $f(x, m_0) \in \mathbb{Z}_{2^l}$, and the chooser relies on each bit of binarized $y \in \{0, 1\}^l$ to make each party get individual share of xy by bitwise multiplication. The complexity of such process depends on the bit length of the two multipliers, i.e., bit length l of y. In a BNN, the model parameters act as the chooser with binary values, i.e., 0 or $+1$, which makes $l = 1$ and thus results in efficient linear computation [84]. On the other hand, the above efficiency gain is for BNNs while most of the DL models have floating-point parameters, e.g., $l \gg 1$, which poses a challenge for OT-based linear computation. One mitigation for such limitation is to put the input-independent computation to the offline phase [128].

Remark: Similar to GC, the OT-based linear computation has limitations as follows. First, the chooser works in bit wise such that the complexity depends linearly on the bit length of multipliers. While the BNNs (or TNNs) are suitable for OT-based linear computation as the model parameters have short bit length, the network performance, e.g., model accuracy, is limited. Second, although optimizations for OT and OT extension [161] have largely improved the OT performance to enable over ten million OT executions in one second [162], the OT complexity for linear computation still needs further optimization, by considering the heavy-load linear computation in state-of-the-art DL models as shown in Table VI, e.g., tens of millions of gates need to be calculated for one layer.

D. SS-Based Linear Computation

The main method for SS-based linear computation is to combine input-independent offline computation and input-dependent online computation. Specifically, the multiplication triplet is generated between a data owner and a cloud server in the offline phase, e.g., each party holds individual share of random numbers a, b, and ab. In the online phase, the dot product is efficiently calculated based on the precomputed shares [48], [79], [121], [127]–[135]. For example, in the Minionn framework [131], the noise r is added into the private input x as $(x-r)$ by the data owner. Then, $(x-r)$ is sent to the cloud server, which conducts dot product as $(x - r)w$, where w is the weight matrix of a well-trained DL model at the cloud server.

The cloud server needs to cancel the term rw in $(x-r)w$ such that the output is the desired result, i.e., xw. Here, rw, which is independent of the input x, can be canceled by the pregenerated shares at both parties: in the offline phase, the data owner applies HE to encrypt a random number (or a random vector) r and then sends the encrypted r to the cloud server. Next, the cloud server homomorphically calculates rw, generates a random share r_s, and returns a share of rw, i.e., $rw-r_s$, to data owner. In this way, the data owner and cloud server each has a share of rw, i.e., $rw - r_s$ and r_s and, thus, xw can be shared as r_0 at cloud server and $((x - r)w + r_s - r_0 + (rw - r_s))$ at data owner, where r_0 is randomly generated by cloud server. Meanwhile, the sharing for rw can also be completed by an OT-based generation [128].

Remark: SS can be efficient for linear computation as the online cost, i.e., the total cost in the input-dependent phase, is cheap. This mitigation comes with the precomputation load, e.g., HE/OT-based share pregeneration, in the offline phase. As a result, SS-based linear computation is more advantageous in

TABLE IV
NONLINEAR COMPUTATION FOR PRIVACY-PRESERVING DL

Schemes	Privacy preserving primitives					Approx.	
	HE	GC	OT	SS	TEE	Pi	Po
Liu [48]	○	○	○	●	○	○	○
Falcon [121]	○	○	○	●	○	○	○
Swift [129]	○	○	○	●	○	●	○
Securenn [132]	○	○	○	●	○	●	○
Trident [133]	○	○	○	●	○	●	○
Huang [134]	○	○	○	●	○	●	○
Leia [135]	○	○	○	●	○	○	○
Bayhenn [140]	○	○	○	●	○	○	○
Gelunet [144]	○	○	○	●	○	○	○
Cheetah [146]	○	○	○	●	○	○	○
Xu [152]	○	○	○	●	○	●	○
Privedge [127]	○	●	○	○	○	○	○
Secureml [128]	○	●	○	○	○	●	○
Chameleon [130]	○	●	○	●	○	○	○
Minionn [131]	○	●	○	○	○	●	○
Quotient [84]	○	●	○	○	○	○	○
Soteria [136]	○	●	○	○	○	○	○
Deepsecure [137]	○	●	○	○	○	○	○
Xonn [138]	○	●	○	○	○	○	○
Cheetah [141]	○	●	○	○	○	○	○
Falcon [145]	○	●	○	○	○	○	○
Gazelle [76]	○	●	○	○	○	○	○
Autoprivacy [148]	○	●	○	○	○	○	○
Ensei [149]	○	●	○	○	○	○	○
Helen [153]	○	●	○	○	○	○	○
DElphi [79]	○	●	○	●	○	○	○
Zhu [139]	●	○	○	○	○	○	●
Cryptonets [142]	●	○	○	○	○	○	●
Faster [143]	●	○	○	○	○	○	●
E2dm [50]	●	○	○	○	○	○	●
Homopai [147]	●	○	○	○	○	○	●
Spindle [150]	●	○	○	○	○	○	●
Badawi [151]	●	○	○	○	○	○	●
Chen [59]	●	○	○	○	○	○	●
Cryptodl [154]	●	○	○	○	○	○	●
Hervé [155]	●	○	○	○	○	○	●
Sesame [156]	○	○	○	○	●	○	○
Darknight [157]	○	○	○	○	●	○	○
Slalom [158]	○	○	○	○	●	○	○
Chiron [159]	○	○	○	○	●	○	○

"●" and "○" denote adopted and unadopted item, respectively.

a low-delay setting as the offline phase can also be cheap. In delay-sensitive settings, the computation load in offline is nonignorable, which makes SS-based linear computation less preferred.

E. TEE-Based Linear Computation

TEE is a hardware-level primitive for either privacy-preserving linear computation or privacy-preserving nonliner computation (see Table IV) [156]. Concretely, both of the data owners and cloud servers negotiate with TEE at the very beginning, such that TEE is authorized to operate on data owners' private data and cloud servers' invaluable DL models (in MLaaS). Then, TEE receives encrypted data from data owners and encrypted model parameters from cloud servers. As TEE is authenticated, it has the decryption keys to get plaintext data and model parameters, which enables plaintext calculation for either inference or training. Since TEE is resource constrained, further acceleration is achieved by utilizing more powerful computation units such as GPUs. Given the distrust of GPU, the security can be guaranteed by secure data blinding [157] and integrity checking [158]. Furthermore, the Chiron framework [159] deals with training under the assumption that TEE runs standard DL training toolchain (including the popular Theano framework and C compiler) while the server provides model-creation code and each data owner individually submits the private training data. The untrusted model-creation code from the server is confined in a Ryoan sandbox to avoid the leakage of training data outside the TEE.

Remark: TEE requires a hardware budget to achieve privacy-preserving linear computation (or nonlinear computation as shown in Table IV), which adds extra burden to data owners during the system deployment. Meanwhile, the execution in TEE totally depends on the data from data owners and DL models from cloud servers (in MLaaS), without extra security guarantee. Thus, the commitment from both parties is required [158], [159]. Furthermore, TEE's interaction with external computation units, e.g., GPUs, should also be carefully designed to provide security guarantee.

V. NONLINEAR COMPUTATION IN DATA-ENCRYPTION-BASED PRIVACY-PRESERVING DL

Table IV shows the primitives for data-encryption-based privacy-preserving nonlinear computation in state-of-the-art schemes, where HE, GC, and SS are preferred. For HE-based nonlinear computation, various polynomial (Po) functions are proposed to approximate the original nonlinear functions, e.g., ReLU and Sigmoid, since HE does not support nonlinearity. On the other hand, non convergence may occur in training with polynomials, as unbounded derivatives can result in gradient explosion. While the system accuracy of DL models trained with polynomials could degrade, compared to the counterparts with original nonlinear functions [155]. For GC-based nonlinear computation, the nonlinear functions, e.g., ReLU, are first transformed into Boolean circuits, which can be exactly computed by GC [76]. Meanwhile, the pipelining design [138] is needed as state-of-the-art DL models [9] involve large-size Boolean circuits. For SS-based nonlinear computation, the bitwise sharing [35] is proposed to enable approximation-free calculation for nonlinear functions such as ReLU. While the communication cost poses a hurdle to its scalability [132], when applied to state-of-the-art DL models [121]. The detailed analysis for each primitive is listed as follows.

A. HE-Based Nonlinear Computation

Due to the linearity of HE, polynomial approximation (Po) is mainly adopted in HE-based nonlinear computation. The second-degree polynomial approximations [143] and third-degree polynomial approximations [147] are generally preferred. Among them, square approximation is widely used [50], [59], [139], [142], [151] which involves HE multiplication. On the other hand, replacing all the original nonlinear functions, e.g., ReLU, with square approximation could result in performance degradation. The natural architecture search (NAS) technique [79] makes a tradeoff between accuracy and complexity by partially replacing the original functions with squares. Additionally, specific approximations are designed for specific nonlinear functions, such as the least squares and Chebyshev polynomials for sigmoid [150], [154], Chebyshev polynomials for softmax [150], and Chebyshev polynomials and polynomial regression for ReLU [154], [155].

Remark: Polynomial approximations have limited convergence intervals, outside of which the derivatives vary a lot. Batch normalization (BN) [121], [163] is introduced to normalize the input of nonlinear functions, such that the input values are within convergence intervals [155], [164], [165]. Meanwhile, piecewise (Pi) functions, e.g., Sign function and ReLU, which involve the value comparison, can be computed by bitwise HE [166] that operates on each bit of plaintext. While Pi functions can infinitely approximate any functions, one should make a tradeoff between efficiency and system accuracy [167].

B. GC-Based Nonlinear Computation

The comparison function can be efficiently implemented by GC, which makes GC an especially preferred primitive for nonlinear computation, e.g., GC-based ReLU [79], [84], [136], [141], [145], [148], [149], [153], GC-based Leaky-ReLU [127], GC-based maxpooling [136], [141], and GC-based maximum function in last layer of DL models [130]. Note that the above GC-based functions introduce no approximation and thus achieve system performance comparable to the original DL models. Other nonlinear functions can be approximated by the GC-based comparison function [131]. For example, Sigmoid can be approximated by ReLu variants [128]. Other works also propose to construct more efficient Boolean circuits for further speedup [76], [137], [138].

Remark: Comparison-based nonlinear functions, e.g., ReLU, benefit efficient computation from GC without approximation. The comparison-based GC module can also approximate any target functions [131] but it incurs an accuracy-efficiency tradeoff. While GC has a constant communication round for a constructed circuit, it results in heavy communication load [160] and large circuit size [137] for state-of-the-art DL models [9], [19]. The pipelining technique is considered to mitigate the GC cost by parallel computation [138].

C. SS-Based Nonlinear Computation

A comparison with any number is in fact the comparison with 0, which is equivalent to evaluating the most significant bit (MSB) of a number [132]. While the MSB evaluation can be efficiently implemented by SS [121]. Therefore, the SS-based MSB evaluation enables the approximation-free computation for ReLU [48], [129], [130], [132]–[134], [152], derivative of ReLU [48], [132], maxpooling [48], [132], derivative of maxpooling [48], [132], BN [121], and activation functions in BNNs [135]. SS-based MSB evaluation is also applied for function approximation such as Relu-based approximation for Sigmoid [129], [133], [152]. Some schemes propose to split the input of nonlinear functions and use the split data to separately get partial nonlinear results, which are then merged to retrieve the exact output of nonlinear functions [140], [144], [146].

Remark: SS-based nonlinear computation has more cost of communication round compared with HE and GC-based counterparts. Besides data owners and cloud servers, some schemes introduce the trust third party (see more discussion in Section VI), which poses limitation for the scalability of SS-based nonlinear computation in real-world cases, where parties mutually distrust each other. One should also bear in mind about the accuracy–efficiency tradeoff in SS-based approximation for nonlinear functions [135].

TABLE V
RECAP FOR PRIVACY-PRESERVING DL

Schemes	Primitives		# parties		
	Single comp.	Hybrid comp.	2	3	≥4
Liu [48]	SS	○	○	●	○
Falcon [121]	SS	○	○	●	○
Swift [129]	SS	○	○	●	●
Securenn [132]	SS	○	○	●	○
Trident [133]	SS	○	○	○	●
Huang [134]	SS	○	○	●	○
Leia [135]	SS	○	●	○	○
Soteria [136]	GC	○	●	○	○
Deepsecure [137]	GC	○	●	○	○
Xonn [138]	GC	○	●	○	○
Zhu [139]	HE	○	○	○	●
Cryptonets [142]	HE	○	●	○	○
Faster [143]	HE	○	●	○	○
E2dm [50]	HE	○	●	○	○
Homopai [147]	HE	○	●	○	○
Spindle [150]	HE	○	○	○	●
Badawi [151]	HE	○	●	○	○
Chen [59]	HE	○	○	○	●
Cryptodl [154]	HE	○	●	○	○
Hervé [155]	HE	○	●	○	○
Sesame [156]	TEE	○	●	○	○
Darknight [157]	TEE	○	●	○	○
Slalom [158]	TEE	○	●	○	○
Chiron [159]	TEE	○	○	○	●
Privedge [127]	○	SS+GC	●	○	○
Secureml [128]	○	SS+GC	●	○	○
Chameleon [130]	○	SS+GC	○	●	○
Minionn [131]	○	SS+GC	●	○	○
DElphi [79]	○	SS+GC	●	○	○
Quotient [84]	○	OT+GC	●	○	○
Bayhenn [140]	○	HE+SS	●	○	○
Gelunet [144]	○	HE+SS	●	○	○
Cheetah [146]	○	HE+SS	●	○	○
Xu [152]	○	HE+SS	●	○	○
Cheetah [141]	○	HE+GC	●	○	○
Falcon [145]	○	HE+GC	●	○	○
Gazelle [76]	○	HE+GC	●	○	○
Autoprivacy [148]	○	HE+GC	●	○	○
Ensei [149]	○	HE+GC	●	○	○
Helen [153]	○	HE+GC	○	○	●

"●" and "○" denote adopted and unadopted item, respectively. Each item in "Hybrid comp." is listed with linear primitive + nonlinear primitive.

VI. JOINT RECAP OF LINEAR AND NONLINEAR COMPUTATIONS IN DATA-ENCRYPTION-BASED PRIVACY-PRESERVING DL

Table V shows the combined primitives for layerwise privacy-preserving computation (including both linear computation and nonlinear computation) in state-of-the-art schemes. There are two main categories, namely, single-primitive computation, denoted by "Single comp." in Table V, and hybrid-primitive computation, denoted by "Hybrid comp." in Table V. In single-primitive computation, HE, SS, and GC are three preferred primitives given less hardware budget, while the combination of HE and GC is preferred in hybrid-primitive computation. Generally, the hybrid-primitive computation achieves better overall performance compared with single-primitive counterpart.

Concretely, single-HE schemes [50], [59], [139], [142], [143], [147], [150], [151], [154], [155] need no extra trusted party while they need to approximate the nonlinear functions into polynomials. Single-SS schemes [48], [121], [129], [132]–[135] generally involve more parties, e.g., an extra trusted party or multiple noncolluding cloud servers, and larger amount of interactions, e.g., communication round among the parties. Single-GC schemes [136]–[138] aim to minimize the NXOR gates in the constructed Boolean circuits or to adopt more GC-friendly models such as BNNs.

As the most preferred combination in hybrid-primitive computation, HE-GC schemes [76], [141], [145], [148], [149], [153] utilize HE's capability for efficient linear computation and GC's advantage in computing comparison function. Meanwhile, no extra trusted party is needed, which is more suitable for scenarios where parties are mutually distrusted. Specifically, the HE-GC framework, GAZELLE [76], has demonstrated three orders of magnitude faster than the single-HE framework, CryptoNets [142], which is one of the classic privacy-preserving systems.

In order to have a more comprehensive overview about the system performance, e.g., computation cost and communication cost, for schemes in each category, i.e., single-HE, single-GC, single-SS and hybrid-primitive schemes, Fig. 3 shows the layerwise and accumulated running time/communication cost of four representative schemes, i.e., CryptoNets [142] for single-HE scheme, DeepSecure [137] for single-GC scheme, SecurNN [132] for single-SS scheme, and GAZELLE [76] for hybrid-primitive scheme.[2]

As for CryptoNets, the linear computation dominates the runtime cost. The communication cost lies in the encrypted input sent from the client to server at very beginning, and the encrypted inference result sent back from server to client at the last. While the communication load is light, the computation cost, i.e., over hundreds of second, limits its practicality and scalability. As for DeepSecure, the large circuit size results in high computation/communication cost. SecureNN's overhead mainly depends on its communication complexity while a trusted third party is needed. Gazelle balances computation complexity and communication complexity by separately considering different properties of linear and nonlinear functions, i.e., HE-based linear computation and GC-based ReLU, thus generally outperforms single-primitive schemes.

[2]The two tested DL models are AlexNet [8] and VGG [9], which are tailored for CIFAR-10 database [168]. The running time and communication cost of CryptoNets are based on the code from https://github.com/Huelse/SEAL-Python; The performance of GAZELLE are based on the code from https://github.com/chiraag/gazelle_mpc and the parameter setting is in line with CHEETAH [146]. We use two workstations as the client and server. Both machines run Ubuntu 16.04LST with Intel i7-8700 3.2-GHz CPU with 12 threads and 16-GB RAM. The network link between them is a Gigabit Ethernet. The performance of DeepSecure and SecureNN are based on the reported computation and communication complexity in their papers. Both of the time and communication cost are for privacy-preserving inference. The ReLU and maxpooling are substituted with square and meanpooling, respectively.

TABLE VI
SYSTEM LOAD OF GAZELLE ON DIFFERENT NETWORKS

Net. Configuration			# XOR gate	# NXOR gate	# HE Cipher
AlexNet	L1	Conv1	-	-	16
		ReLU1	~8712000	~4356000	-
		MaxPool1	~16796160	~8398080	-
	L2	Conv2	-	-	7
		ReLU2	~5598720	~2799360	-
		MaxPool2	~10383360	~5191680	-
	L3	Conv3	-	-	5
		ReLU3	~1946880	~973440	-
	L4	Conv4	-	-	7
		ReLU4	~1946880	~973440	-
	L5	Conv5	-	-	7
		ReLU5	~1297920	~648960	-
		MaxPool5	~2211840	~1105920	-
	L6	FC6	-	-	1
		ReLU6	~122880	~61440	-
	L7	FC7	-	-	1
		ReLU7	~122880	~61440	
	L8	FC8	-	-	1
VGG-16	L1	Conv1	-	-	16
		ReLU1	~96337920	~48168960	-
	L2	Conv2	-	-	322
		ReLU2	~96337920	~48168960	-
		MaxPool2	~72253440	~36126720	
	L3	Conv3	-	-	81
		ReLU3	~48168960	~24084480	-
	L4	Conv4	-	-	161
		ReLU4	~48168960	~24084480	-
		MaxPool4	~36126720	~18063360	-
	L5	Conv5	-	-	41
		ReLU5	~24084480	~12042240	-
	L6	Conv6	-	-	81
		ReLU6	~24084480	~12042240	-
	L7	Conv7	-	-	81
		ReLU7	~24084480	~12042240	-
		MaxPool7	~18063360	~9031680	-
	L6	Conv8	-	-	21
		ReLU8	~12042240	~6021120	-
	L9	Conv9	-	-	41
		ReLU9	~12042240	~6021120	-
	L7	Conv10	-	-	41
		ReLU10	~12042240	~6021120	-
		MaxPool10	~9031680	~4515840	-
	L11	Conv11	-	-	11
		ReLU11	~3010560	~1505280	-
	L12	Conv12	-	-	11
		ReLU12	~3010560	~1505280	-
	L13	Conv13	-	-	11
		ReLU13	~3010560	~1505280	-
		MaxPool13	~2257920	~1128960	-
	L14	FC14	-	-	3
		ReLU14	~122880	~61440	-
	L15	FC15	-	-	1
		ReLU15	~122880	~61440	-
	L16	FC16	-	-	1

To further identify how different network sizes, i.e., different dimensions for linear computation and nonlinear computation, affect the computation/communication performance in real-world DL models, we further investigate the hybrid-primitive, the GAZELLE framework, by applying the original input, i.e., the IMAGENET data set [169] with high-resolution images, and original nonlinear functions, i.e., ReLU and maxpooling, on AlexNet and VGG. Table VI shows the concrete computation/communication cost of GAZELLE.[3] Specifically, the

[3]The numbers of XOR and NXOR gates are estimated based on reported complexity in DeepSecure [137].

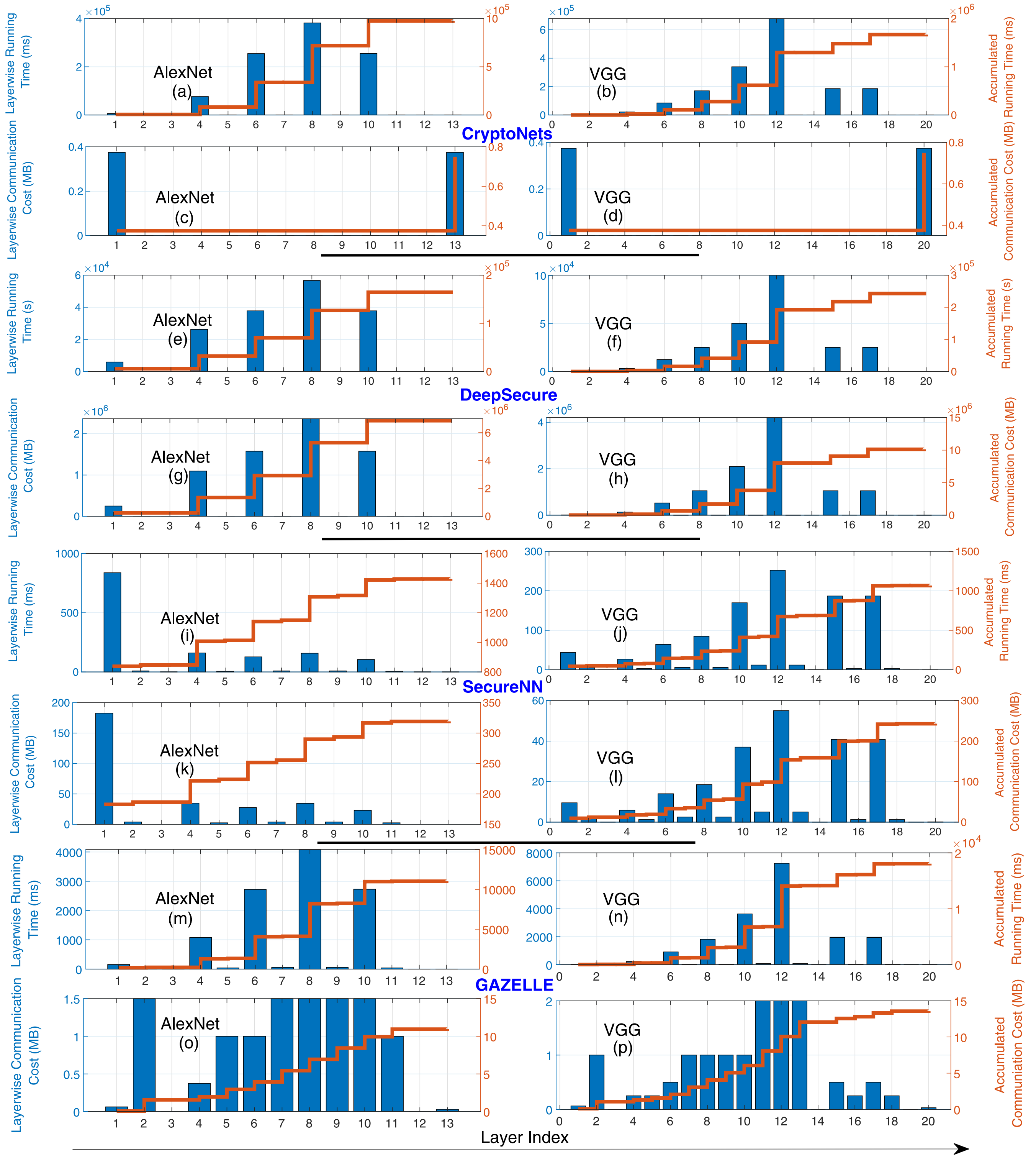


Fig. 3. Layerwise and accumulated running time and communication cost for different frameworks. (a), (e), (i), and (m) Layerwise and accumulated running time of CryptoNets, DeepSecure, SecureNN and Gazelle on AlexNet. (b), (f), (j), and (n) Layerwise and accumulated running time of CryptoNets, DeepSecure, SecureNN and Gazelle on VGG. (c), (g), (k), and (o) Layerwise and accumulated communication cost of CryptoNets, DeepSecure, SecureNN and Gazelle on AlexNet. (d), (h), (l), and (p) Layerwise and accumulated communication cost of CryptoNets, DeepSecure, SecureNN and Gazelle on VGG. For (a), (b), (e), (f), (i), (j), (m), and (n), the bar with values on the left *y*-axis indicates layerwise running time, and the curve with values on the right *y*-axis indicates the accumulated running time. For (c), (d), (g), (h), (k), (l), (o), and (p), the bar with values on the left *y*-axis indicates layerwise communication cost, and the curve with values on the right *y*-axis indicates the accumulated communication cost.

linear computation relies on the HE operations over ciphertexts while the nonlinear computation, i.e., ReLU and maxpooling, involves GC calculation over XOR and NXOR gates. Two observations are listed as follows: 1) large-size data involve a large number of ciphertexts and 2) large-size input of the nonlinear functions results in large-size circuits, e.g., over

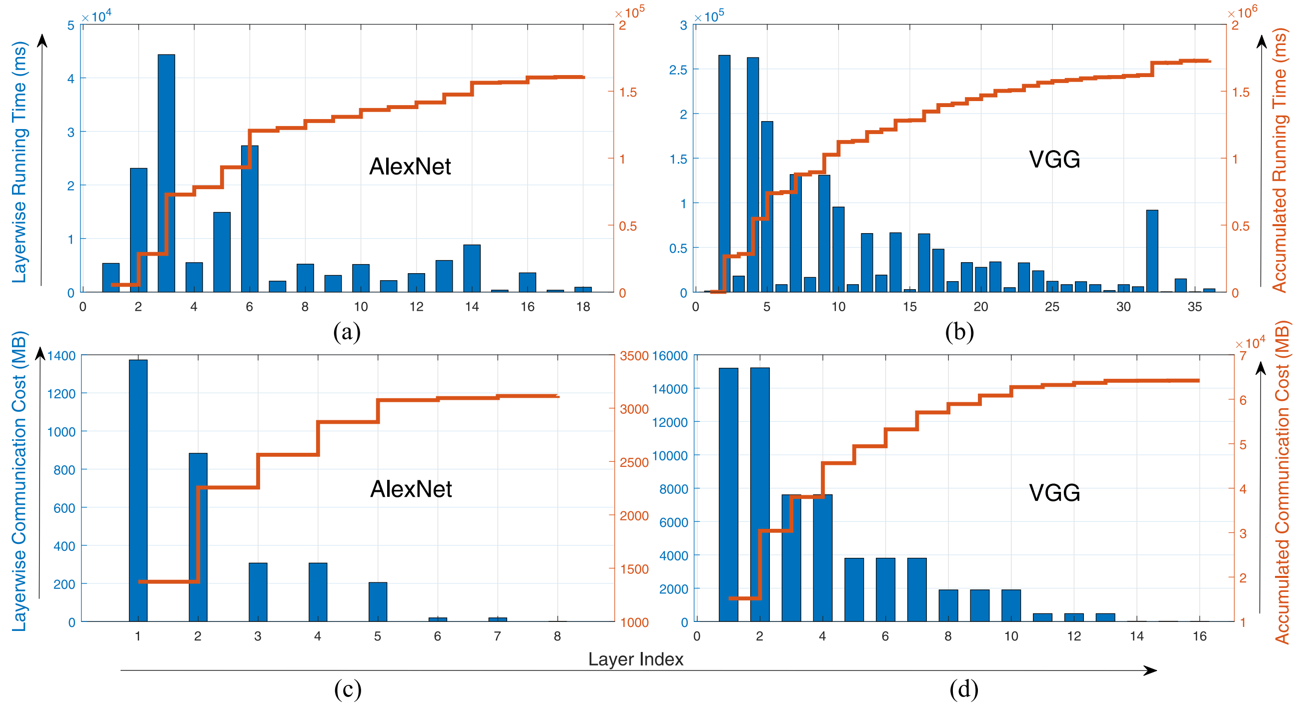


Fig. 4. Layerwise and accumulated running time and communication cost of Gazelle on state-of-the-art networks. Layerwise and accumulated running time on (a) AlexNet and (b) VGG-16. Layerwise and accumulated communication cost on (c) AlexNet and (d) VGG-16. For (a) and (b), the bar with values on the left y-axis indicates layerwise running time, and the curve with values on the right y-axis indicates the accumulated running time. For (c) and (d), the bar with values on the left y-axis indicates layerwise communication cost, and the curve with values on the right y-axis indicates the accumulated communication cost. The layer index corresponds to the layer sequence in Table VI.

millions of XOR and NXOR gates are needed for the ReLU and maxpooling. This large ciphertexts/circuits load poses challenges for the scalability and practicality for GAZELLE, which is one of the flagships in state-of-the-art data-encryption-based privacy-preserving frameworks.

Fig. 4 further shows GAZELLE's corresponding running time and communication cost. The running time for AlexNet and VGG is over 100 and over 1000 s, respectively. Large-size feature maps and massive kernels lead to high computation cost, e.g., the running time for the first a few layers in both AlexNet and VGG is high. This cost is prohibitive in many applications. For example, the time constraints in many real-time speech recognition systems (such as Alexa and Google Assistant) are within 10 s [170], [171] while autonomous cars even demand an immediate response less than one second [172]. Meanwhile, several gigabytes and tens of gigabytes for AlexNet and VGG also pose a challenge to the network traffic. Overall dedicated effort is still needed to make the data-encryption-based privacy-preserving schemes more practical.

On the other hand, there is a tradeoff between efficiency and model accuracy in an effort to reduce the communication/computation cost. For example, the model accuracy could drop by adopting the approximation mechanism for nonlinear computation, while the communication/computation cost is reduced. To have a more quantitative estimation for such a tradeoff in data-encryption-based privacy-preserving DL, Table VII shows the model accuracy and efficiency w.r.t. different network sizes,[4] i.e., the number of model layers, in single-primitive and hybrid-primitive schemes, e.g., single-HE schemes [50], [59], [139], [142], [143], [147], [150], [151], [154], [155], single-SS, schemes [48], [121], [129], [132]–[135], single-GC schemes [136]–[138], and HE-GC schemes [76], [141], [145], [148], [149], [153].

For single-HE training schemes [147], [150], [154], backpropagation is explored in small-size networks, e.g., the 2-layer linear model or a 3-layer neural network. The massive iterations in backpropagation result in significant computation cost even in plaintext DL models [19], which cannot be easily addressed with deeper networks. While the efficiency decreases from minutes (Mins) to days with more layers, the model accuracy of trained models also drops [154] since the nonlinear functions are approximated. For single-HE inference schemes, the classic CryptoNets [142] tackles a small CNN model with an efficiency level of Mins. The efficiency is further improved to seconds (Ss) by optimizations to reduce the complexity of HE operations [50], [59], [139], [155]. Meanwhile, the inference accuracy drops [143], [151] with more layers due to the approximated nonlinear functions.

As for the single-SS training schemes, the model accuracy is kept with negligible loss as the network goes deeper, while the tradeoff lies in the efficiency from hours (Hrs) to days or even to weeks. For single-SS inference schemes, the inference accuracy is maintained by introducing more parties, i.e., the extra trusted party [134] or multiple servers [135], to generate the desired shares. Furthermore, special computation-efficient models, e.g., BNNs in [135], could result in accuracy drop with deeper networks.

[4]The statistics are reported in the respective paper.

TABLE VII
QUANTITATIVE EVALUATION FOR PRIVACY-PRESERVING DL SCHEMES

Schemes	# of layers	Accuracy drop (%)	Efficiency level			
			Days	Hrs	Mins	Ss
All-HE schemes						
Homopai [147]	∼2	∼1	○	○	●	○
Spindle [150]	∼2	∼1	○	○	●	○
Cryptodl [154]	∼3	∼4	○	●	○	○
Cryptonets [142]	∼3	∼1	○	○	●	○
E2dm [50]	∼3	∼1	○	○	○	●
Chen [59]	∼3	∼1	○	○	○	●
Zhu [139]	∼8	∼1	○	○	○	●
Hervé [155]	∼8	∼1	○	○	●	○
Badawi [151]	∼11	∼4	○	○	●	○
Faster [143]	∼50	∼4	○	○	●	○
All-SS schemes						
Securenn [132]	∼4	∼1	○	●	○	○
Liu [48]	∼6	∼2	○	●	○	○
Falcon [121]	∼16	∼1	●	○	○	○
Swift [129]	∼3	∼1	○	○	○	●
Trident [133]	∼3	∼1	○	○	○	●
Huang [134]	∼10	∼1	○	○	○	●
Leia [135]	∼13	∼7	○	○	○	●
All-GC schemes						
Deepsecure [137]	∼3	∼1	○	○	○	●
Soteria [136]	∼13	∼10	○	○	○	●
Xonn [138]	∼16	∼13	○	○	○	●
HE-GC schemes						
Helen [153]	∼2	∼1	○	●	○	○
Gazelle [76]	∼5	∼1	○	○	○	●
Falcon [145]	∼10	∼1	○	○	○	●
Ensei [149]	∼18	∼1	○	○	○	●
Autoprivacy [148]	∼32	∼1	○	○	○	●
Cheetah [141]	∼50	∼1	○	○	○	●

Systems marked with orange indicate they are training-enabled.

As for single-GC schemes, the tradeoff tends to maintain the efficiency (at the level of seconds) while accepting a certain loss of model accuracy. One of the reasons is that the networks are appropriately modified to keep the efficiency [138] as complexity increases with deeper models.

As for HE-GC training schemes, small-size networks, e.g., a simple 2-layer linear model, are explored to have a good balance between model accuracy and efficiency. For HE-GC inference schemes, the model accuracy is maintained as the nonlinear functions, e.g., ReLU, can be exactly computed by GC. Meanwhile, the efficiency is improved by deep optimizations of respective HE and GC calculation [76], [141], [145], [148], [149].

VII. Discussions and Future Directions

In a nutshell, the practicality-oriented efficiency improvement toward modern networks, and the corresponding balance between model accuracy and efficiency form two main optimization targets in data-encryption-based privacy-preserving DL. As a promising mitigation to balance these two targets, the hybrid-primitive schemes are preferred with a better system performance, i.e., model accuracy and efficiency. However, there is still a significant gap between the achieved performance and practical demand [146] (see the detailed analysis in Section VI). Therefore, we suggest several promising directions to possibly shorten the gap.

A. Conversion Among Shares in Hybrid-Primitive Schemes

Generally, different properties of linear and nonlinear computation make hybrid-primitive schemes outperform single-primitive counterparts. In hybrid-primitive schemes, it is inevitable to conduct a particular conversion among different data types (e.g., arithmetic shares, the Boolean shares, and Yao's GC shares), which are resulted from different primitives. The conversion is an important factor that affects the overall system cost, especially for large-size input and models. While the hybrid-primitive computation is involving and being optimized [104], [105], adopting single-primitive methodology can circumvent the share conversion [146]. However, it may also incur challenges for efficient and accurate computation, i.e., linear and nonlinear computation, as linear functions are suitable for the arithmetic values while the nonlinear functions are suitable for the Boolean values. Thus, designing more efficient and accurate modules with single primitive for both linear and nonlinear functions may speedup the overall system performance.

B. Reconsidering the Linear-And-Nonlinear Logic for Privacy-Preserving DL

The fundamental workflow in DL is the repetition of linear and nonlinear computation, which is also a golden rule in privacy-preserving DL. Specifically, all privacy-preserving schemes come up with designs to first tackle the linear computation, and then solve the computation of nonlinear functions by taking the linear output as nonlinear input. As shown in [146], this seemly logical rule may hinder the improvement for the overall system. For example, given the input to one layer in the DL model, the intermediate data (i.e., the output of linear function) are calculated for the final output (i.e., the nonlinear result). Obviously, each layer does not necessarily need that specific intermediate data, i.e., the linear output, if the nonlinear output can be obtained by efficiently calculating another intermediate data. While it is interesting, the construction of that intermediate data remains challenging and needs more insights.

C. Parallel Computing-Based Hardware-Software Codesign for Larger and Deeper Networks

The state-of-the-art DL models have massive layers, e.g., over one hundred layers [19], and large-size input, e.g., three-channel images with 2-D size of 227×227 [169]. Besides optimization for computation algorithms, how to pipelining such large networks with large-volume input remains another hurdle for privacy-preserving DL, as privacy-preserving primitives, e.g., HE and GC, may process data by big modulus, which are not efficiently fit for current parallel computing techniques. The batching technique that computes privacy-preserving data, e.g., encrypted input, in parallel is mostly used, e.g., SIMD for HE [54], [76] and circuit pipeline for GC [138]. Meanwhile, several protocols for the linear and nonlinear functions are recently proposed [121], [173], which involve large amount of plaintext computation, e.g., computation for arithmetic numbers, as such they can benefit more from the current parallel computing techniques.

On the other hand, there are some emerging schemes that involve specifically designed hardware to accelerate the primitive, e.g., speedup the number theoretic transform for HE, and thus improve the system efficiency [174]–[177]. Overall, the designs for algorithmic parallelism and hardware acceleration are still relatively disjoint, and the performance may be further improved by hardware-software codesign toward better pipelining for encrypted data.

D. Network Architecture Selection on Primitive-Integrated Platform

As pointed out in many works [45], [178]–[181], the DL models always contain redundancy. Therefore, lots of networks are compressed to boost the computation efficiency. Meanwhile, the advanced computation algebra [182]–[185] further speedups the plaintext-level computation. However, how to apply these optimizations into privacy-preserving computation remains challenging as plaintext-level computation should be transformed into crypto arithmetic with big modulus, which makes the plaintext-level acceleration infeasible.

In another word, there is a need to search crypto-friendly computation architecture that is both efficient and accuracy-guaranteed. A few works have considered the network structure search in privacy-preserving DL, i.e., nonlinear function selection [79] and BNN selection [136]. Searching for the network architecture includes finding: 1) the fit kernel sizes for convolution; 2) the pooling methods; 3) the nonlinear activation functions; and 4) the connections for the above three elements, all should consider the properties of the primitives, e.g., HE-based square is more efficient that GC-based ReLU. Meanwhile, a recent work shows an efficient Integer-Arithmetic-Only CNN structure [186], which may be used as a searching option for the integer-in-nature primitives such as HE.

Furthermore, given the optimized and modularized DL platforms, e.g., tensorflow [165], integrating the privacy-preserving primitives in those platforms can utilize the well developed computation advantage and thus give a chance to improve the efficiency of model search. A few works have embedded some primitives, e.g., HE, into tensorflow [187]–[192] while the comprehensive integration into DL platforms needs more efforts to facilitate the network search for optimal privacy-preserving DL architectures.

VIII. CONCLUSION

In this article, we have reviewed the state-of-the-art researches in privacy-preserving DL based on multiparty secure computation that relies on data encryption techniques and have summarized these techniques in both training phase and inference phase. Specifically, we have categorized the data-encryption-based privacy-preserving techniques w.r.t. the linear and nonlinear computations, which are the two basic and repeated building blocks in DL. By first giving an overview of each primitive, we have then concretely introduced the way each primitive is used in linear and nonlinear functions. Next, we have had a combinational analysis of both linear and nonlinear computations w.r.t. efficiency and model accuracy. After the quantitative comparison for several representative schemes, we have presented some technical hurdles and discussed several promising directions for future research.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for the constructive and insightful guidance and comments.

REFERENCES

[1] W. Li, H. Song, and F. Zeng, “Policy-based secure and trustworthy sensing for Internet of Things in smart cities,” *IEEE Internet Things J.*, vol. 5, no. 2, pp. 716–723, Apr. 2018.

[2] G. Li *et al.*, “Energy efficient data collection in large-scale Internet of Things via computation offloading,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4176–4187, Jun. 2019.

[3] A. Adshead, *Data Set to Grow 10-Fold by 2020 as Internet of Things Takes Off*, vol. 9, ComputerWeekly.com, London, U.K., 2014.

[4] M. S. Hossain, M. Al-Hammadi, and G. Muhammad, “Automatic fruit classification using deep learning for industrial applications,” *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 1027–1034, Feb. 2019.

[5] C. Esposito, X. Su, S. A. Aljawarneh, and C. Choi, “Securing collaborative deep learning in industrial applications within adversarial scenarios,” *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4972–4981, Nov. 2018.

[6] A. Bashar, “Survey on evolving deep learning neural network architectures,” *J. Artif. Intell.*, vol. 1, no. 2, pp. 73–82, 2019.

[7] W. G. Hatcher and W. Yu, “A survey of deep learning: Platforms, applications and emerging research trends,” *IEEE Access*, vol. 6, pp. 24411–24432, 2018.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2012, pp. 1097–1105.

[9] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014. Accessed: Aug. 5, 2020. [Online]. Available: arXiv:1409.1556.

[10] G. Hinton *et al.*, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[11] G. E. Dahl, D. Yu, L. Deng, and A. Acero, “Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition,” *IEEE Trans. Audio, Speech, Language Process.*, vol. 20, no. 1, pp. 30–42, Jan. 2012.

[12] L. Deng, G. Hinton, and B. Kingsbury, “New types of deep neural network learning for speech recognition and related applications: An overview,” in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, Vancouver, BC, Canada, 2013, pp. 8599–8603.

[13] M. Sabokrou, M. Fayyaz, M. Fathy, and R. Klette, “Deep-cascade: Cascading 3D deep neural networks for fast anomaly detection and localization in crowded scenes,” *IEEE Trans. Image Process.*, vol. 26, pp. 1992–2004, 2017.

[14] W. Luo *et al.*, “Video anomaly detection with sparse coding inspired deep neural networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 3, pp. 1070–1084, Mar. 2021.

[15] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim, “An empirical study on network anomaly detection using convolutional neural networks,” in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, Vienna, Austria, 2018, pp. 1595–1598.

[16] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, “Deep learning for IoT big data and streaming analytics: A survey,” *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2923–2960, 4th Quart., 2018.

[17] F. Tang, Z. M. Fadlullah, B. Mao, and N. Kato, “An intelligent traffic load prediction-based adaptive channel assignment algorithm in SDN-IoT: A deep learning approach,” *IEEE Internet Things J.*, vol. 5, no. 6, pp. 5141–5154, Dec. 2018.

[18] C. Szegedy *et al.*, “Going deeper with convolutions,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 1–9.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.

[20] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, "MLaaS: Machine learning as a service," in *Proc. IEEE 14th Int. Conf. Mach. Learn. Appl.*, Miami, FL, USA, 2015, pp. 896–902.

[21] H. Assem, L. Xu, T. S. Buda, and D. O'Sullivan, "Machine learning as a service for enabling Internet of Things and people," *Pers. Ubiquitous Comput.*, vol. 20, no. 6, pp. 899–914, 2016.

[22] T. C. Rindfleisch, "Privacy, information technology, and health care," *Commun. ACM*, vol. 40, no. 8, pp. 92–100, 1997.

[23] M. Meingast, T. Roosta, and S. Sastry, "Security and privacy issues with health care information technology," in *Proc. Int. Conf. IEEE Eng. Med. Biol. Soc.*, New York, NY, USA, 2006, pp. 5453–5458.

[24] M. A. Rothstein, "Is deidentification sufficient to protect health privacy in research?" *Amer. J. Bioethics*, vol. 10, no. 9, pp. 3–11, 2010.

[25] W. Gao, W. Yu, F. Liang, W. G. Hatcher, and C. Lu, "Privacy-preserving auction for big data trading using homomorphic encryption," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, pp. 776–791, Apr.–Jun. 2020.

[26] P. Voigt and A. Von dem Bussche, "The EU general data protection regulation (GDPR)," in *A Practical Guide*, 1st ed. Cham, Switzerland: Springer, 2017.

[27] W. B. Chik, "The Singapore personal data protection act and an assessment of future trends in data privacy reform," *Comput. Law Security Rev.*, vol. 29, no. 5, pp. 554–575, 2013.

[28] E. Goldman, "An introduction to the California consumer privacy act (CCPA)," in *Santa Clara University Legal Studies Research Paper*, Santa Clara Univ., Santa Clara, CA, USA, 2020.

[29] "Health insurance portability and accountability act of 1996," in *Public Law 104-191*, ASPE, Washington, DC, USA, 1996.

[30] (2018). *Uber to Pay 148 Million Penalty to Settle 2016 Data Breach*. Accessed: Aug. 5, 2020. [Online]. Available: https://www.wsj.com/articles/uber-to-pay-148-million-penalty-to-settle-2016-data-breach-1537983127

[31] (2019). *Data Breach News: PDPC Fines IHiS, SingHealth*. Accessed: Aug. 5, 2020. [Online]. Available: https://www.channelnewsasia.com/news/singapore/ihis-singhealth-fined-1-million-data-breach-cyberattack-11124156

[32] (2019). *France Fines Google 57 Million for Breaking Europe's Strict New Privacy Rules*. Accessed: Aug. 5, 2020. [Online]. Available: https://www.businessinsider.com/france-fines-google-57-million-for-gdpr-breach-2019-1

[33] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 587–601.

[34] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, 2015, pp. 1322–1333.

[35] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.

[36] H. C. Tanuwidjaja, R. Choi, and K. Kim, "A survey on deep learning techniques for privacy-preserving," in *Proc. Int. Conf. Mach. Learn. Cyber Security*, 2019, pp. 29–46.

[37] M. Al-Rubaie and J. M. Chang, "Privacy-preserving machine learning: Threats and solutions," *IEEE Security Privacy*, vol. 17, no. 2, pp. 49–58, Mar./Apr. 2019.

[38] Z. Shan, K. Ren, M. Blanton, and C. Wang, "Practical secure computation outsourcing: A survey," *ACM Comput. Surveys*, vol. 51, no. 2, pp. 1–40, 2018.

[39] F. Mirshghallah, M. Taram, P. Vepakomma, A. Singh, R. Raskar, and H. Esmaeilzadeh, "Privacy in deep learning: A survey," 2020. Accessed: Aug. 5, 2020. [Online]. Available: arXiv:2004.12254.

[40] A. Boulemtafes, A. Derhab, and Y. Challal, "A review of privacy-preserving techniques for deep learning," *Neurocomputing*, vol. 384, pp. 21–45, Apr. 2020.

[41] C. Dwork, "Differential privacy: A survey of results," in *Proc. Int. Conf. Theory Appl. Models Comput.*, 2008, pp. 1–19.

[42] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, nos. 3–4, pp. 211–407, 2014.

[43] C. Dwork, G. N. Rothblum, and S. Vadhan, "Boosting and differential privacy," in *Proc. IEEE 51st Annu. Symp. Found. Comput. Sci.*, Las Vegas, NV, USA, 2010, pp. 51–60.

[44] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proc. Theory Cryptogr. Conf.*, 2006, pp. 265–284.

[45] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.

[46] Y. LeCun *et al.*, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*. San Mateo, CA, USA: M. Kaufmann, 1990, pp. 396–404.

[47] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. 22nd ACM Int. Conf. Multimedia*, 2014, pp. 675–678.

[48] Z. Liu, I. Tjuawinata, C. Xing, and K.-Y. Lam, "MPC-enabled privacy-preserving neural network training against malicious attack," 2020. Accessed: Aug. 5, 2020. [Online]. Available: arXiv:2007.12557.

[49] A. Aloufi, P. Hu, Y. Song, and K. E. Lauter, "Computing blindfolded on data homomorphically encrypted under multiple keys: An extended survey," IACR Cryptol. ePrint Archive, Lyon, France, Rep. 2020/447, 2020.

[50] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2018, pp. 1209–1222.

[51] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, 1999, pp. 223–238.

[52] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

[53] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. Theory Cryptogr. Conf.*, 2005, pp. 325–341.

[54] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, 2009, pp. 169–178.

[55] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *Proc. Annu. Cryptol. Conf.*, 2012, pp. 868–886.

[56] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," IACR Cryptol. ePrint Archive, Lyon, France, Rep. 2012/144, 2012.

[57] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Security*, 2017, pp. 409–437.

[58] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," *Designs Codes Cryptogr.*, vol. 71, no. 1, pp. 57–81, 2014.

[59] H. Chen, W. Dai, M. Kim, and Y. Song, "Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2019, pp. 395–412.

[60] A. Wood, K. Najarian, and D. Kahrobaei, "Homomorphic encryption for machine learning in medicine and bioinformatics," *ACM Comput. Surveys*, vol. 53, p. 70, Aug. 2020.

[61] *HE Libs*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/jonaschn/awesome-he

[62] *SEAL*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/microsoft/SEAL

[63] *HElib*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/homenc/HElib

[64] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, 2014.

[65] *TFHE*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/tfhe/tfhe

[66] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Proc. Annu. Cryptol. Conf.*, 2013, pp. 75–92.

[67] *HEANN*.Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/snucrypto/HEAAN

[68] *Palisade*. Accessed: Aug. 5, 2020. [Online]. Available: https://git.njit.edu/palisade/PALISADE

[69] D. Stehlé and R. Steinfeld, "Making NTRU as secure as worst-case problems over ideal lattices," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2011, pp. 27–47.

[70] $\wedge \circ \lambda$. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/cpeikert/Lol

[71] *Cingu*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/CEA-LIST/Cingulata

[72] *FV-NFLlib*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/CryptoExperts/FV-NFLlib

[73] *Lattigo*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/ldsec/lattigo

[74] *NuFHE*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/nucypher/nufhe

[75] *Marble*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/MarbleHE/Marble

[76] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Security Symp.*, 2018, pp. 1651–1669.

[77] A. C. Yao, "Protocols for secure computations," in *Proc. IEEE 23rd Annu. Symp. Found. Comput. Sci.*, Chicago, IL, USA, 1982, pp. 160–164.

[78] Y. Lindell and B. Pinkas, "Secure two-party computation via cut-and-choose oblivious transfer," *J. Cryptol.*, vol. 25, no. 4, pp. 680–722, 2012.

[79] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Proc. 29th USENIX Security Symp.*, 2020, pp. 2505–2522.

[80] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *Proc. Int. Colloquium Automata Lang. Program.*, 2008, pp. 486–498.

[81] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proc. 22nd Annu. ACM Symp. Theory Comput.*, 1990, pp. 503–513.

[82] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *Proc. IEEE Symp. Security Privacy*, 2013, pp. 478–492.

[83] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, 2015, pp. 220–250.

[84] N. Agrawal, A. S. Shamsabadi, M. J. Kusner, and A. Gascón, "QUOTIENT: Two-party secure neural network training and prediction," in *Proc. ACM SIGSAC Conf. Compu. Commun. Security*, 2019, pp. 1231–1247.

[85] P. Snyder, "Yao's garbled circuits: Recent directions and implementations," in *Literature Review*, Dept. Comput. Sci., Univ. Illinois Chicago, Chicago, IL, USA, 2014.

[86] O. Goldreich, S. Micali, and A. Wigderson, "How to play ANY mental game," in *Proc. 19th Annu. ACM Symp. Theory Comput.*, 1987, pp. 218–229.

[87] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Proc. Annu. Int. Cryptol. Conf.*, 2003, pp. 145–161.

[88] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[89] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[90] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proc. Annu. Int. Cryptol. Conf.*, 1991, pp. 420–432.

[91] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara, "High-throughput semi-honest secure three-party computation with an honest majority," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2016, pp. 805–817.

[92] T. Araki *et al.*, "Optimized honest-majority MPC for malicious adversaries–breaking the 1 billion-gate per second barrier," in *Proc. IEEE Symp. Security Privacy*, San Jose, CA, USA, 2017, pp. 843–862.

[93] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "SoK: General purpose compilers for secure multi-party computation," in *Proc. IEEE Symp. Security Privacy*, San Francisco, CA, USA, 2019, pp. 1220–1237.

[94] *SMPC List*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/rdragos/awesome-mpc

[95] *SoK*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/MPC-SoK/frameworks

[96] *EMP*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/emp-toolkit

[97] *Obliv-C*. Accessed: Aug. 5, 2020. [Online]. Available: https://oblivc.org/

[98] *ObliVM*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/oblivm

[99] *TinyG*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/esonghori/TinyGarble

[100] *SCALE-MAMBA: General MPC With Secret Sharing*. Accessed: Aug. 5, 2020. [Online]. Available: https://homes.esat.kuleuven.be/∼nsmart/SCALE/

[101] *Wysteria: Multparty Computation with GMW*. Accessed: Aug. 5, 2020. [Online]. Available: https://bitbucket.org/aseemr/wysteria/wiki/Home

[102] *Sharemind*. Accessed: Aug. 5, 2020. [Online]. Available: https://sharemind.cyber.ee/

[103] *PICCO*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/PICCO-Team/picco

[104] *ABY*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/encryptogroup/ABY

[105] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2018, pp. 35–52.

[106] *Frigate: C-Style Compiler for Optimizing Boolean Circuits*. Accessed: Aug. 5, 2020. [Online]. Available: https://bitbucket.org/bmood/frigaterelease/src/master/

[107] *CBMC-GC: Creates Boolean Circuits for Secure Computation*. Accessed: Aug. 5, 2020. [Online]. Available: https://gitlab.com/securityengineering/CBMC-GC-2

[108] *HyCC: Optimizes Circuits for Hybrid MPC From ANSI-C*. Accessed: Aug. 5, 2020. [Online]. Available: https://gitlab.com/securityengineering/HyCC

[109] *TASTY*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/encryptogroup/tasty

[110] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," IACR Cryptol. ePrint Archive, Lyon, France, Rep. 2020/521, 2020.

[111] F. McKeen *et al.*, "Intel® software guard extensions (Intel® SGX) support for dynamic memory management inside an enclave," in *Proc. Hardw. Archit. Support Security Privacy*, 2016, pp. 1–9.

[112] F. McKeen *et al.*, "Innovative instructions and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardw. Archit. Support Security Privacy (HASP ISCA)*, vol. 10, 2013, p. 10.

[113] R. Bahmani *et al.*, "Secure multiparty computation from SGX," in *Proc. Int. Conf. Financ. Cryptogr. Data Security*, 2017, pp. 477–497.

[114] P. Koeberl, V. Phegade, A. Rajan, T. Schneider, S. Schulz, and M. Zhdanova, "Time to rethink: Trust brokerage using trusted execution environments," in *Proc. Int. Conf. Trust Trustworthy Comput.*, 2015, pp. 181–190.

[115] M. Barbosa, B. Portela, G. Scerri, and B. Warinschi, "Foundations of hardware-based attested computation and application to SGX," in *Proc. IEEE Eur. Symp. Security Privacy*, Saarbrucken, Germany, 2016, pp. 245–260.

[116] *Collection of TEE Based Open-Source Libraries*. Accessed: Aug. 5, 2020. [Online]. Available: https://github.com/Maxul/Awesome-SGX-Open-Source

[117] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *Proc. 11th USENIX Workshop Offensive Technol.*, 2017, p. 11.

[118] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on Intel SGX," in *Proc. 10th Eur. Workshop Syst. Security*, 2017, pp. 1–6.

[119] A. Biondo, M. Conti, L. Davi, T. Frassetto, and A.-R. Sadeghi, "The guard's dilemma: Efficient code-reuse attacks against Intel SGX," in *Proc. 27th USENIX Security Symp.*, 2018, pp. 1213–1227.

[120] A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. 27th Annu. Symp. Found. Comput. Sci.*, Toronto, ON, Canada, 1986, pp. 162–167.

[121] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "FALCON: Honest-majority maliciously secure framework for private deep learning," 2020. [Online]. Available: arXiv:2004.02229.

[122] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu, "Hermes attack: Steal DNN models with lossless inference accuracy," 2020. [Online]. Available: arXiv:2006.12784.

[123] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Security Symp.*, 2016, pp. 601–618.

[124] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Security Privacy*, San Jose, CA, USA, 2017, pp. 3–18.

[125] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2805–2824, Sep. 2019.

[126] B. Biggio *et al.*, "Evasion attacks against machine learning at test time," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, 2013, pp. 387–402.

[127] A. S. Shamsabadi, A. Gascón, H. Haddadi, and A. Cavallaro, "PrivEdge: From local to distributed private training and prediction," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3819–3831, 2020.

[128] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Security Privacy*, San Jose, CA, USA, 2017, pp. 19–38.

[129] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "SWIFT: Super-fast and robust privacy-preserving machine learning," 2020. [Online]. Available: arXiv:2005.10296.

[130] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. Asia Conf. Comput. Commun. Security*, 2018, pp. 707–721.

[131] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 619–631.

[132] S. Wagh, D. Gupta, and N. Chandran, "Securenn: Efficient and private neural network training," IACR Cryptol. ePrint Archives, Lyon, France, Rep. 2018/442, 2018.

[133] H. Chaudhari, R. Rachuri, and A. Suresh, "Trident: Efficient 4PC framework for privacy preserving machine learning," in *Proc. 27th Annu. Netw. Distrib. Syst. Security Symp.*, 2020, pp. 23–26.

[134] K. Huang, X. Liu, S. Fu, D. Guo, and M. Xu, "A lightweight privacy-preserving cnn feature extraction framework for mobile sensing," *IEEE Trans. Depend. Secure Comput.*, early access, Apr. 26, 2019, doi: 10.1109/TDSC.2019.2913362.

[135] X. Liu, B. Wu, X. Yuan, and X. Yi, "Leia: A lightweight cryptographic neural network inference system at the edge," IACR Cryptol. ePrint Archive, Lyon, France, Rep. 2020/463, 2020.

[136] A. Aggarwal, T. E. Carlson, R. Shokri, and S. Tople, "SOTERIA: In search of efficient neural networks for private inference," 2020. [Online]. Available: arXiv:2007.12934.

[137] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proc. 55th Annu. Design Autom. Conf.*, 2018, pp. 1–6.

[138] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN XNOR-based oblivious deep neural network inference," in *Proc. 28th USENIX Security Symp.*, 2019, pp. 1501–1518.

[139] R. Zhu, C. Ding, and Y. Huang, "Practical MPC+FHE with applications in secure multi-partyneural network evaluation," IACR Cryptol. ePrint Archive, Lyon, France, Rep. 2020/550, 2020.

[140] P. Xie, B. Wu, and G. Sun, "BAYHENN: Combining Bayesian deep learning and homomorphic encryption for secure dnn inference," 2019. [Online]. Available: arXiv:1906.00639.

[141] B. Reagen *et al.*, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," 2020. [Online]. Available: arXiv:2006.00505.

[142] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 201–210.

[143] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster CryptoNets: Leveraging sparsity for real-world encrypted inference," 2018. [Online]. Available: arXiv:1811.09953.

[144] Q. Zhang, C. Wang, H. Wu, C. Xin, and T. V. Phuong, "GELU-net: A globally encrypted, locally unencrypted deep neural network for privacy-preserved learning," in *Proc. 27th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2018, pp. 3933–3939.

[145] S. Li *et al.*, "FALCON: A fourier transform based approach for fast and secure convolutional neural network predictions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 8705–8714.

[146] Q. Zhang, C. Wang, C. Xin, and H. Wu, "CHEETAH: An ultra-fast, approximation-free, and privacy-preserved neural network framework based on joint obscure linear and nonlinear computations," 2019. [Online]. Available: arXiv:1911.05184.

[147] Q. Li *et al.*, "HomoPAI: A secure collaborative machine learning platform based on homomorphic encryption," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 1713–1717.

[148] Q. Lou, B. Song, and L. Jiang, "AutoPrivacy: Automated layer-wise parameter selection for secure neural network inference," 2020. [Online]. Available: arXiv:2006.04219.

[149] S. Bian, T. Wang, M. Hiromoto, Y. Shi, and T. Sato, "ENSEI: Efficient secure inference via frequency-domain homomorphic convolution for privacy-preserving visual recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Seattle, WA, USA, 2020, pp. 9403–9412.

[150] D. Froelicher *et al.*, "Scalable privacy-preserving distributed learning," 2020. [Online]. Available: arXiv:2005.09532.

[151] A. A. Badawi *et al.*, "Towards the AlexNet moment for homomorphic encryption: HCNN, thefirst homomorphic CNN on encrypted data with GPUs," 2018. [Online]. Available: arXiv:1811.00778.

[152] X. Ma, X. Chen, and X. Zhang, "Non-interactive privacy-preserving neural network prediction," *Inf. Sci.*, vol. 481, pp. 507–519, May 2019.

[153] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *Proc. IEEE Symp. Security Privacy*, San Francisco, CA, USA, 2019, pp. 724–738.

[154] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 3, pp. 123–142, 2018.

[155] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," IACR Cryptol. ePrint Archive, Lyon, France, Rep. 2017/35, 2017.

[156] S. Banerjee, P. Ramrakhyani, S. Wei, and M. Tiwari, "SESAME: Software defined enclaves to secure inference accelerators with multi-tenant execution," 2020. [Online]. Available: arXiv:2007.06751.

[157] H. Hashemi, Y. Wang, and M. Annavaram, "DarKnight: A data privacy scheme for training and inference of deep neural networks," 2020. [Online]. Available: arXiv:2006.01300.

[158] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," 2018. [Online]. Available: arXiv:1806.03287.

[159] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," 2018. [Online]. Available: arXiv:1803.05961.

[160] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Found. Trends Privacy Security*, vol. 2, nos. 2–3, pp. 70–246, 2017.

[161] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2013, pp. 535–548.

[162] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient multiparty computation toolkit," 2016. Accessed: Aug. 3, 2020. [Online]. Available: https://github.com/emp-toolkit

[163] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015. [Online]. Available: arXiv:1502.03167.

[164] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 4700–4708.

[165] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement.*, 2016, pp. 265–283.

[166] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *Proc. 22nd Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, 2015, pp. 1–14.

[167] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," IACR Cryptol. ePrint Archives, Lyon, France, Rep. 2017/1114, 2017.

[168] *The CIFAR-10 Dataset*. Accessed: Aug. 5, 2020. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[169] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Miami, FL, USA, 2009, pp. 248–255.

[170] *Developers' Alexa Interface for Device APIs*. Accessed: Aug. 5, 2020. [Online]. Available: https://developer.amazon.com/zh/docs/device-apis/alexa-interface.html

[171] *The Google Developer Interface for Web Conversation*. Accessed: Aug. 5, 2020. [Online]. Available: https://developers.google.com/actions/reference/rest/conversation-webhook

[172] V. V. Dixit, S. Chand, and D. J. Nair, "Autonomous vehicles: Disengagements, accidents and reaction times," *PLoS ONE*, vol. 11, no. 12, 2016, Art. no. e0168054.

[173] D. Rathee *et al.*, "Cryptflow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2020, pp. 325–342.

[174] R. S. Agrawal, L. Bu, A. Ehret, and M. A. Kinsy, "Fast arithmetic hardware library for rlwe-based homomorphic encryption," 2020. [Online]. Available: arXiv:2007.01648.

[175] M. S. Riazi, K. Laine, B. Pelton, and W. Dai, "HEAX: An architecture for computing on encrypted data," in *Proc. 25th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2020, pp. 1295–1309.

[176] T. Morshed, M. M. A. Aziz, and N. Mohammed, "CPU and GPU accelerated fully homomorphic encryption," 2020. [Online]. Available: arXiv:2005.01945.

[177] D. Reis, J. Takeshita, T. Jung, M. T. Niemier, and X. S. Hu, "Computing-in-memory for performance and energy efficient homomorphic encryption," 2020. [Online]. Available: arXiv:2005.03002.
[178] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015. [Online]. Available: arXiv:1510.00149.
[179] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2015, pp. 1135–1143.
[180] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 243–254, 2016.
[181] A. Kozlov, I. Lazarevich, V. Shamporov, N. Lyalyushkin, and Y. Gorbachev, "Neural network compression framework for fast model inference," 2020. [Online]. Available: arXiv:2002.08679.
[182] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "SpArch: Efficient architecture for sparse matrix multiplication," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, San Diego, CA, USA, 2020, pp. 261–274.
[183] F. Le Gall, "Powers of tensors and fast matrix multiplication," in *Proc. 39th Int. Symp. Symb. Algebraic Comput.*, 2014, pp. 296–303.
[184] R. Yuster and U. Zwick, "Fast sparse matrix multiplication," *ACM Trans. Algorithms*, vol. 1, no. 1, pp. 2–13, 2005.
[185] X. Huang and V. Y. Pan, "Fast rectangular matrix multiplication and applications," *J. Complexity*, vol. 14, no. 2, pp. 257–299, 1998.
[186] H. Zhao, D. Liu, and H. Li, "Efficient integer-arithmetic-only convolutional neural networks," 2020. [Online]. Available: arXiv:2006.11735.
[187] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure tensorflow inference," in *Proc. IEEE Symp. Security Privacy*, San Francisco, CA, USA, 2020, pp. 336–353.
[188] T. Ryffel *et al.*, "A generic framework for privacy preserving deep learning," 2018. [Online]. Available: arXiv:1811.04017.
[189] M. Dahl *et al.*, "Private machine learning in tensorflow using secure computation," 2018. [Online]. Available: arXiv:1810.08130.
[190] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "nGraph-HE2: A high-throughput framework for neural network inference on encrypted data," in *Proc. 7th ACM Workshop Encrypted Comput. Appl. Homomorphic Cryptogr.*, 2019, pp. 45–56.
[191] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," in *Proc. 16th ACM Int. Conf. Comput. Front.*, 2019, pp. 3–13.
[192] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "MP2ML: A mixed-protocol machine learning framework for private inference," in *Proc. 15th Int. Conf. Availability Rel. Security*, 2020, pp. 1–10.

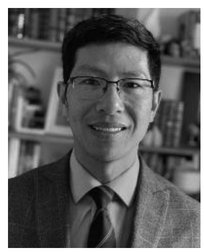
Qiao Zhang received the B.S. and M.S. degrees from the School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing, China, in 2014 and 2017, respectively. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, VA, USA.

Her current research focuses on privacy-preserving machine learning.

Chunsheng Xin (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the State University of New York at Buffalo, Buffalo, NY, USA, in 2002.

He is a Professor with the School of Cybersecurity, and the Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, VA, USA. His research has been supported by almost 20 NSF and other federal grants, and results in more than 100 papers in leading journals and conferences, including three best paper awards, as well as books, book chapters, and patent. His interests include cybersecurity, machine learning, wireless communications and networking, cyber–physical systems, and Internet of Things.

Prof. Xin has served as a Co-Editor-in-Chief/Associate Editors of multiple international journals, and the Symposium/Track Chair of multiple international conferences, including IEEE Globecom and ICCCN.



Hongyi Wu (Fellow, IEEE) received the B.S. degree in scientific instruments from Zhejiang University, Hangzhou, China, in 1996, the M.S. degree in electrical engineering, and the Ph.D. degree in computer science from the State University of New York at Buffalo, Buffalo, NY, USA, in 2000 and 2002, respectively.

He is the Batten Chair of Cybersecurity and the Director of the School of Cybersecurity, Old Dominion University (ODU), Norfolk, VA, USA, where he is also a Professor with the Department of Electrical and Computer Engineering and holds joint appointment with the Department of Computer Science. Before joining ODU, he was an Alfred and Helen Lamson Endowed Professor with the Center for Advanced Computer Studies, University of Louisiana at Lafayette, Lafayette, LA, USA. His research focuses on networked cyber–physical systems for security, safety, and emergency management applications, where the devices are often lightweight, with extremely limited computing power, storage space, communication bandwidth, and battery supply.

Prof. Wu received the NSF CAREER Award in 2004 and the UL Lafayette Distinguished Professor Award in 2011.