Received 11 October 2022, accepted 28 October 2022, date of publication 3 November 2022, date of current version 10 November 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3219049

SURVEY

A Survey of Deep Learning Architectures for Privacy-Preserving Machine Learning With Fully Homomorphic Encryption

ROBERT PODSCHWADT[1], (Student Member, IEEE), DANIEL TAKABI[1], (Member, IEEE), PEIZHAO HU[2], (Member, IEEE), MOHAMMAD H. RAFIEI[1], AND ZHIPENG CAI[1], (Senior Member, IEEE)
[1]Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA
[2]Department of Computer Science, Rochester Institute of Technology, Rochester, NY 14623, USA

Corresponding author: Robert Podschwadt (rpodschwadt1@gsu.edu)

This work was supported in part by the National Science Foundation under Grant 2054968 and Grant 2118083, and in part by the Microsoft Faculty Fellowship Program.

ABSTRACT Outsourced computation for neural networks allows users access to state-of-the-art models without investing in specialized hardware and know-how. The problem is that the users lose control over potentially privacy-sensitive data. With homomorphic encryption (HE), a third party can perform computation on encrypted data without revealing its content. In this paper, we reviewed scientific articles and publications in the particular area of Deep Learning Architectures for Privacy-Preserving Machine Learning (PPML) with Fully HE. We analyzed the changes to neural network models and architectures to make them compatible with HE and how these changes impact performance. Next, we find numerous challenges to HE-based privacy-preserving deep learning, such as computational overhead, usability, and limitations posed by the encryption schemes. Furthermore, we discuss potential solutions to the HE PPML challenges. Finally, we propose evaluation metrics that allow for a better and more meaningful comparison of PPML solutions.

INDEX TERMS Deep learning, homomorphic encryption, neural networks, privacy, privacy preservation, machine learning.

I. INTRODUCTION

In Machine Learning as a Service (MLaaS), a service provider (also called server or the cloud), offers computational resources and sometimes trained models to a client who owns the data. However, the need to share data raises privacy concerns for one or both parties; clients need to share their data with the server, or the server needs to share its model with the client. To address this, different private outsourced computation techniques such as Differential Privacy [1], Secure Multiparty Computation [2], Homomorphic Encryption (HE) [3], Functional Encryption [4], or Trusted Execution Environments [5] have been applied to machine learning (ML). These solutions are called Privacy-Preserving Machine Learning (PPML). In this paper, our focus is only on HE for privacy preservation in neural networks (NN).

HE has been called the holy grail of encryption [6], [7]. It offers privacy for both the client's and the server's data. Since the server can do all the processing offline, no information about the network leaks to the client. Besides, the majority of the NN operations are HE-friendly, e.g., dot products are very simple to evaluate on encrypted data. However, using HE with NNs poses some challenges, including Multiplicative Depth, limited operations, and computational complexity. **Multiplicative Depth** is the number of consecutive multiplications that can be applied to a ciphertext before the ciphertext can no longer be decrypted correctly. In contrast to plain data, the supported HE **operations** are limited to only addition and multiplication. The ciphertext operations **computational complexity** is much higher than plaintext operations both in terms of memory consumption and processing time.

In this work, we review state-of-the-art papers that (1) include a wide range of solutions to the challenges of

The associate editor coordinating the review of this manuscript and approving it for publication was Yiming Tang.

NNs with HE, (2) present innovative solutions and establish important techniques in the field, and (3) investigate challenging problems. We discuss their pros and cons and propose some future avenues to tackle. Furthermore, we propose a set of evaluation guidelines to compare solutions in terms of resource requirements. Similar surveys mostly focus on either a broader approach to cryptographic tools [8], security and privacy [9], PPML libraries [10], bench-marking and comparison of HE schemes [11], or fully HE (FHE) compilers [12]. In contrast, our approach focuses specifically on PPML with HE for NNs. In fact, we conduct a survey of the existing PPML research, providing an in-depth comparison of their techniques. The main motivation behind this work is to help the PPML research community observe (1) current efforts/available solutions in HE PPML and (2) the existing gaps/challenges and their potential solutions. We analyzed the changes to neural network models and architectures to make them compatible with HE and how these changes impact performance. Next, we find numerous challenges to HE-based privacy-preserving deep learning, such as computational overhead, usability, and limitations posed by the encryption schemes. Furthermore, we discuss potential solutions to the HE PPML challenges. Finally, we propose evaluation metrics that allow for a better and more meaningful comparison of PPML solutions.

Our work is organized as follows: In Section II, we provide an introduction to MLaaS, HE, and NNs techniques. In Section III, we present an overview of the selected studies, their approaches, and strengths and weaknesses. In Section IV, we discuss literature-based strategies to make NN layers computation possible over HE data, dubbed making NN HE-friendly. In Section V, we take an in-depth look at frequently-used activation functions for HE. We discuss NNs adaptions to HE constraints in Section VI. In Section VII, we investigate potential security weaknesses of the selected studies' solutions. In Section VIII, we compare the experimental results reported by the papers. In Section IX, we propose resource-independent performance metrics and evaluation and reporting standards for a fair comparison of PPML solutions. In Section X, we discuss challenges and future research directions. We conclude in Section XI.

II. BACKGROUND

A. MLaaS WITH PRIVACY PROTECTION

MLaaS is a form of outsourced computation, shown schematically in Fig. 1. A client uses resources provided by a service provider (server). In MLaaS without privacy preservation (Fig. 1a), the client sends their plain data to the server, where an ML model processes it. Afterward, the server returns the result to the client. The problem is that the client has no control over what the server does with the data; it needs to trust the server to process the data correctly and only use it in the agreed-upon manner, e.g., the server does not sell it to another party. The correct computation issue can be addressed using zero-knowledge proofs [13], which is beyond the scope of this work. Zero-knowledge proofs allow one

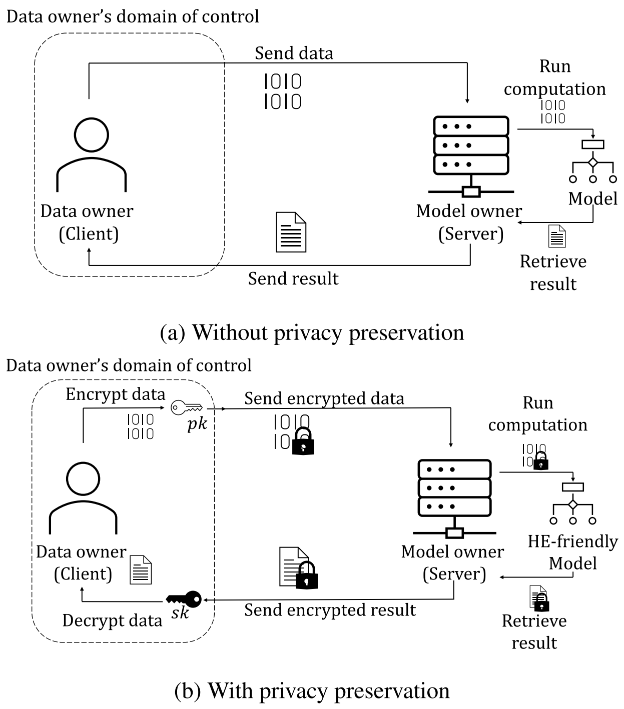


FIGURE 1. Machine learning as a service architecture, with and without privacy preservation. The area inside the dotted line in 1b represents the area in which the client can keep secrets.

party to prove to another party that a certain statement is true without revealing any other information. The privacy of the client's data can be protected using HE, as shown in Fig. 1b. Here the data never leaves the control of the client in plaintext. The server only receives encrypted data, and it can perform HE computation on it, e.g., evaluate an NN. The result of the computation is also encrypted. This way, the server learns neither the input nor the output, guaranteeing complete privacy for the data.

The client can also use the server to train a model on encrypted data. Here they transfer an encrypted dataset and let the server train an NN on it. This is less common due to the high computational complexity of training an NN, especially on encrypted data. Other techniques besides HE can provide preservation in MLaaS, which we will review in the following section.

B. TECHNIQUES FOR PRIVACY PRESERVATION

Different privacy preservation techniques can be used for MLaaS. However, some techniques are better suited for certain scenarios than others. Also, the cryptographic guarantees the techniques provide vary. Here we will give an overview of popular techniques, starting with the less secure and proceeding to the more secure ones.

1) FEDERATED LEARNING

[14] is a technique to train ML models on decentralized data. The data is spread across multiple parties; the goal is to

train a model without needing to share the data. Each party often trains a model locally on their own data and shares gradient updates to train a combined model. This way, they do not need to share their data with the other parties involved. On its own, federate learning does not provide cryptographic privacy guarantees [15]. However, the gradient aggregation can further be secured using Secure Multiparty Computation. A central authority can aggregate the gradients [16], or the involved parties can each collect the gradient updates [17]. Hence, every party has a copy of the final model.

2) DIFFERENTIAL PRIVACY

[1] adds random noise to the data obscuring private information in the data. Based on the amount of noise added, a probability bound on the information leakage can be computed, indicating how likely an adversary can extract private information from the data. The main application of Differential Privacy is to protect personal information in the training data. The goal is to prevent an attacker from extracting private information from the trained model by controlling how much each sample can influence the parameters during training [18]. Multiple parties can also use it to train a shared model on distributed data without disclosing private data to other participants [19]. The primary strength of Differential Privacy is that it operates on standard numerical data types, meaning most ML libraries and hardware accelerators are supported, making the implementation and run-time overhead negligible. On the other hand, using Differential Privacy reduces the quality of the model predictions. Differential Privacy does not offer any cryptographic guarantees and only offers a probability bound on possible information leakage.

3) SECURE MULTIPARTY COMPUTATION

[20] is a term for numerous algorithms and protocols, like garbled circuits [2], secret sharing [21], and oblivious transfer [22], that allow two or more parties to jointly evaluate a function without revealing their inputs to each other. With these primitives, researchers have developed protocols to evaluate NNs while preserving the privacy of the inputs [23], [24], [25]. However, these protocols often require a lot of communication during the computation, which can make network latency a problem. It is possible to avoid some communication by performing expensive pre-computation in an offline phase [26], [27].

4) FUNCTIONAL ENCRYPTION

[4], [28] is a form of encryption that allows the evaluation of certain functions over encrypted data. The results of these functions are "leaked" from the ciphertexts, i.e., the result of the function is in plain data. This is beneficial for NN computation since only the first layer needs to be run on encrypted data [29], [30], and the rest can be run on plain data. However, this does leak some client data information to the server; the server learns the model output and the intermediate results of the computation.

5) TRUSTED EXECUTION ENVIRONMENTS

Like ARM TrustZone [31] and Intel SGX [32], are hardware enclaves inside the central processing unit (CPU). The data is inaccessible while inside the CPU. Before writing the data to memory, the CPU encrypts it. However, in practice, multiple attacks compromise the security of these hardware enclaves [33], [34], making them less attractive for PPML.

C. HOMOMORPHIC ENCRYPTION (HE)

Public-key (asymmetric) encryption schemes [35] use separate keys for encryption and decryption. The key used for encryption is called the public key pk, and the key used for decryption is called the private or secret key sk. The public key can be freely shared with anyone and be used for encryption, but only the holder of the secret key can decrypt the message. Public-key schemes are designed in such a way that having access to the public key does not allow an attacker to extract the private key [36]. In addition to being public-key schemes, HE schemes allow computation on encrypted data. The result of the computation is also encrypted; the data does not need to be decrypted during the computation. Having all the stages of the computation encrypted, from the inputs over the intermediate values to the results, makes HE schemes ideal for outsourced computation. After decryption, the result of the encrypted computation is the same as if the computation was performed on plain data [37].

For encryption, we start with a message $m \in \mathcal{M}$. $\mathcal{M}$ is called the message space in this paper. Common message spaces are integers $\mathcal{M} = \mathbb{Z}$, real numbers $\mathcal{M} = \mathbb{R}$, or single bits $\mathcal{M} = \{0, 1\}$. To encrypt m, it needs to be encoded into a plaintext p, which comes from the plaintext space $\mathcal{P}$. The encoding of m into p is done by the encoding function encode and the reverse operation decode. p can then be encrypted into a ciphertext c. The transformation from a plaintext into a ciphertext is done by the encryption function Enc. It uses the public key to encrypt the plaintext p into the ciphertext c:

$$c = \mathrm{Enc}(\mathrm{pk}, p) \tag{1}$$

The decryption operation is performed by the decryption function Dec, which uses the secret key sk to turn a ciphertext c back into a plaintext p:

$$p = \mathrm{Dec}(\mathrm{sk}, c) \tag{2}$$

where HE schemes differ from other public-key schemes is that they also have an evaluation function Eval that can evaluate a circuit C, a sequence of operations. E.g., a circuit can be an ML model. Evaluating C on plain data and on encrypted data gives us the same result after decryption:

$$\mathrm{Dec}(\mathrm{sk}, \mathrm{Eval}(pk, C, c_0, \cdots, c_n)) = C(p_0, \cdots, p_n) \tag{3}$$

where $c_0, \cdots, c_n = \mathrm{Enc}(\mathrm{pk}, p_0, \cdots, p_n)$ are the encryptions of the plaintexts $p_0, \cdots, p_n$.

We can categorize HE schemes by the operations that can be used in the circuit C and the depth of C. The depth of a circuit is the number of consecutive operations that need to be performed to evaluate the circuit. Here, we provide brief explanations of the HE schemes categorization established by Armknecht et al. [38]:

1) PARTIALLY HOMOMORPHIC ENCRYPTION

Schemes are the "simplest" HE schemes; they only support a limited set of circuits since they can only evaluate either addition or multiplication on encrypted data [39], [40].

2) SOMEWHAT HOMOMORPHIC ENCRYPTION

Schemes can support both multiplication and addition on encrypted data. However, the size of the ciphertexts grows with every computation performed. The depth of the supported circuits can be controlled by the encryption parameters [41].

3) LEVELED HOMOMORPHIC ENCRYPTION

Schemes are very similar to somewhat homomorphic schemes in definition. However, their schemes require that the size of the ciphertexts does not grow as operations are performed. The depth of the circuits that can be evaluated using Leveled Homomorphic Encryption schemes can be controlled with a parameter d. The size of the ciphertexts needs to be independent of d and only rely on the security level. Leveled Homomorphic Encryption schemes that support both addition and multiplication are sometimes called leveled fully homomorphic [42].

4) FULLY HOMOMORPHIC ENCRYPTION

Schemes have no restrictions on the circuits they can evaluate. That means they need to be able to evaluate circuits of arbitrary depth and have no restrictions regarding the operations required.

The limited circuit depth stems from the way encryption works. In simple terms: encryption adds noise to the plaintext, thereby obscuring it. The decryption process removes that noise. Every operation that is performed on encrypted data increases the noise inside the ciphertexts. If the noise passes a certain threshold, correct decryption becomes impossible [43]. One solution is to decrypt a ciphertext c and then encrypting it again, leading to a fresh ciphertext c' with a reset noise level. However, this requires access to the secret key sk. To address this issue, Gentry [3] describes how to build an FHE scheme out of a scheme that can evaluate its decryption function on encrypted data by proposing a bootstrapping method:

$$c' = \text{Eval}(\text{pk}, \text{Dec}, \text{Enc}(\text{sk}), c) \tag{4}$$

In Eq. 4, the decryption circuit, $C = \text{Dec}$, is evaluated with an encryption of sk as input. This does not completely reduce the noise inside the ciphertext but reduces the noise level; hence, further computation can be performed. The bootstrapping operation can be performed as often as necessary, allowing FHE schemes to evaluate circuits of arbitrary depth.

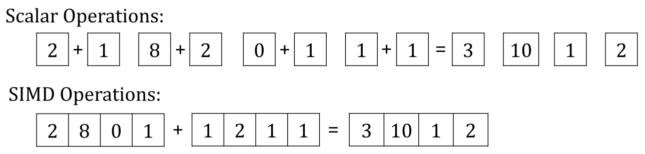


FIGURE 2. Scalar vs Single Instruction Multiple Data (SIMD) operations with 4 slots.

However, bootstrapping relies on circular security, i.e., the secret key needs to be encrypted with its corresponding public key [44]. Circular security can be eliminated by using keyswitching [45], [46]. Keyswitching uses a chain of different secret keys where each key is encrypted with the following key in the chain. Evaluating the bootstrapping function using one of the keys from the chain results in the ciphertext c' being encrypted with the next key in the chain. However, once all keys in the chain have been used, no further computation can be performed. Additionally, bootstrapping is computationally expensive, which is why it is often not used in practice. We refer the reader to Brakerski [45] for a complete discussion of FHE and to Armknecht et al. [38] for more in-depth information about the different HE scheme types.

D. SINGLE INSTRUCTION MULTIPLE DATA

Single Instruction Multiple Data (SIMD) [47] is a form of parallel processing. It can be applied to encrypted data in some HE schemes [48]. SIMD operations can reduce the computational overhead introduced by encrypted computation. SIMD allows the user to encode multiple messages into a single plaintext. Operations on the plaintext (or a ciphertext that encrypts it) are performed on all messages encoded within it. The number of messages a ciphertext can hold is called slots. The number of filled slots does affect the complexity of the operations; they have the same complexity if one or all slots are filled. The encryption parameters govern a ciphertext's number of slots. Fig. 2 shows the difference between scalar and SIMD additions in an example with four slots. However, in practice, the number of slots is typically in the order of thousands. In the scalar case, the operation requires four additions, but when using SIMD, four values can be packed together, and only one addition is necessary. All schemes, except for TFHE [49] (a fast Fully Homomorphic Encryption scheme over the Torus) support SIMD operations.

E. HOMOMORPHIC ENCRYPTION (HE) SCHEMES

Some authors, especially in earlier work, use the **YASHE** (Yet another somewhat homomorphic encryption scheme) [50] scheme. However, Albrecht et al. [51] show that this scheme can be attacked using subfield attacks. The other four schemes rely on the Ring Learning With Errors [52] hardness assumption, a problem that is thought to be quantum hard. Like YASHE, **BGV** (Brakerski/Fan-Vercauteren scheme) [42] and **BFV** (Brakerski-Gentry-Vaikuntanathan) [53] only support integers. The **CKKS**

TABLE 1. A comparison of fully homomorphic encryption (FHE) schemes with respect to their supported message space, supported operations on encrypted data (division is only with respect to plaintext divisors), Single Instruction Multiple Data (SIMD) support, and strengths and weaknesses.

Scheme	Message Space			Supported Operations				SIMD	Strengths	Weaknesses
	Integer	Real	Binary	Addition	Multiplication	Division	Bitwise			
YASHE	•			•	•			•	- exact arithmetic	- not secure - plaintext overflow
BFV	•			•	•			•	- exact arithmetic	- plaintext overflow
BGV	•			•	•			•	- exact arithmetic	- plaintext overflow
CKKS		•		•	•	•		•	- real numbers - division - no plaintext overflow	- approximate arithmetic
TFHE			•				•		- fast bootstrapping - fast binary gates	- slow polynomials - no SIMD

(Cheon-Kim-Kim-Song scheme), also called HEAAN (Homomorphic Encryption for Arithmetic of Approximate Numbers), [54] scheme enables computation on real numbers. However, we need to relax the definition of HE to include CKKS. The usual definition for HE requires that the result of computation on encrypted data and plain data must be the same. CKKS only performs approximate computation. The result on encrypted data and plain data will be almost the same except for a small approximation error. The authors designed the scheme so that the error will first appear in the least significant bits of the results. The **TFHE** [49] scheme only supports individual bits as messages. It supports various binary gates from which complex circuits can be built. One of its main features is comparatively fast bootstrapping.

In Table 1, we compare the strength and weaknesses of the schemes. YASHE has the obvious downside that it is insecure. YASHE, BGV, and BFV all perform exact computation on integers, meaning the result on plain and encrypted data will be the same. These schemes' downside is that they need to deal with growing values during computation. The absolute value of the plaintext in the encrypted ciphertext can not grow larger than one of the scheme's parameters, the plaintext modulus. If, during the computation, the values grow larger than the plaintext modulus, the decryption will no longer be correct. The parameters must be chosen large enough to hold any possible value that could arise during the computation. The problem worsens because the schemes only support integer messages and no division. A typical approach is to encode floating point numbers as integers by scaling them up by a constant factor. However, repeated multiplication, as in dot products or polynomials, quickly drives up the magnitude of the encrypted values. And due to the lack of division operation, there is no possibility to scale the values and reduce their size during computation.

There are some encryption libraries that include one or more of the aforementioned encryption schemes. PALISADE [55] is an encryption library that provides support for BFV, BGV, CKKS, and TFHE. HElib [56], the first FHE library, supports BGV, BFV, and CKKS. However, it only supports bootstrapping for BGV and BFV. SEAL [57], a popular encryption library, provides implementations of CKKS and BFV without bootstrapping. HEAAN [58], the official implementation of the CKKS scheme, is the only library that supports CKKS bootstrapping. The TFHE library [59] implements the scheme by the same name.

F. NEURAL NETWORKS (NNs)

In this paper, we focus on privacy preservation for NNs. NNs are weighted, directed graphs in which input nodes are connected to output nodes through several hidden nodes. Nodes, also called neurons or units, are organized in layers. Typically, the computation of a layer is the weighted sum of all inputs to which a non-linear activation function is applied. An iterative optimization algorithm, like stochastic gradient descent, updates the weights to reduce a loss function. This process is also called training.

1) NNs ON ENCRYPTED DATA

The majority of the computation, the weighted sums, in an NN can be performed on encrypted data. The non-linear activation functions pose the main problem. It is impossible to efficiently evaluate the most common activation functions like the Rectified Linear Unit (ReLU), Hyperbolic Tangent (Tanh), Softmax, and Sigmoid on encrypted data. Without these non-linearities, the NN would only be able to learn linear functions.

2) TRAINING VS. INFERENCE

Often, privacy-preserving solutions for NNs only cover inference. On the other hand, training is more computationally expensive since it requires a forward and backward pass through the model. The increase in operations leads to significant noise buildup, making training a much higher multiplicative depth process than inference. Furthermore, the weights will be encrypted with the same key as the data, meaning the server will have a model it cannot access after.

III. STRENGTHS AND WEAKNESSES

This section provides an overview of state-of-the-art literature in the emerging field of PPML with NNs and HE, along with a critical review of their strength and weaknesses (summarized in Table 2).

A. LOW OVERHEAD

Bourse et al. [60] propose a method for privacy-preserving NN inference to evaluate NNs of arbitrary depth on the MNIST repository. To achieve this, the authors perform bootstrapping after every layer. Due to a significantly reduced message space, the used ciphertexts are small; however, it is unclear if this approach is scalable to more complex data.

B. HIGH THROUGHPUT

Dowlin et al. [61] propose CryptoNets, one of the first solutions using FHE for NN inference. Chabanne et al. [62] propose an extension of CryptoNets [61]. They show that using a batch normalization layer before each activation layer stabilizes training with polynomial activation functions. Hesamifard et al. [63] build CryptoDL, a system similar to CryptoNets [61]. However, the authors aim to find a better low-degree polynomial approximation for Sigmoid, Tanh, and ReLU, thereby improving the quality of the network's predictions.

All these solutions, e.g., CryptoNets and CryptoDL, use SIMD batching to offset some of the computational overhead, allowing them to process large batches of inputs efficiently. The downside is that they can only process small batches with a large overhead. Another weakness is the limited plaintext space and the accommodations authors need to make to account for it. Zhang et al. [64] develop a solution for encrypted speech recognition that also employs SIMD batching. Although their solution has unconstrained message space in contrast to, say, CryptoNets and CryptoDL, the final decoding of the NN's output needs to be performed by the client.

C. IMPROVED COMPUTATION

Lou et al. [65] propose a solution called SHE for Shift-accumulation-based leveled-HE-enabled deep NN, based on TFHE, which allows the implementation of the ReLU function and max-pooling layers. This solution increases the performance since there is no need for function approximation. However, TFHE performs matrix operations slower than the other HE schemes.

Chou et al.'s [66] solution, called Faster CryptoNets, improves upon the work by Dowlin et al. [61] by pruning and quantizing the models to reduce the number of required operations and increase the sparsity in the weights' polynomial encoding. This encoding increases the efficiency of the multiplication algorithms [67].

Jiang et al. [68] introduce an algorithm for HE matrix multiplication based on CKKS. It uses one ciphertext to multiply two $d \times d$ matrices, a complexity of $O(d)$, which is an improvement over Halevi [56], with a complexity of $O(d^2)$ and d ciphertexts. The authors build a new framework, called E2DM (which stands for encrypted data and encrypted model), for privacy-preserving NN inference based on this algorithm and its extension for non-squared matrices.

Brutzkus et al. [69] present different encrypted data representations using SIMD and ways to switch between these representations to reduce the latency of NN. These data representations reduce the memory requirement and allow to perform operations like convolutions and matrix multiplication efficiently. Lee et al. [70] propose an algorithm for faster encrypted convolutions with a stride of two based on SIMD data packing. For convolutions with stride one, the authors use the algorithm presented by Juvekar et al. [71]. Furthermore, to speed up the computation of polynomials, they use a baby step giant algorithm [72]. For evaluating deep models, like Resnet-20 [73], the authors implement advanced bootstrapping techniques [72], [74] in SEAL. Mihara et al. [75] propose a solution for training NN privately. To speed up the computation, the authors propose a new SIMD matrix batching technique in which a matrix is arranged diagonally into a ciphertext. This new arrangement reduces the number of operations required to transpose the matrix. A considerable downside of this solution is that it requires an interactive phase for noise removal, despite being evaluated on a tiny dataset. A shared weakness of the above solutions is that they all rely on a particular data layout that needs to be designed case-by-case.

Jang et al. [76] propose a variation of the CKKS scheme, MatHEAAN (Matrix HEEAN), that specializes in matrix operations. Based on this scheme, the authors implement Gated Recurrent Units (GRU) [77] to handle sequential data. Since recurrent NNs (RNNs) typically have high multiplicative depth, the authors use their custom bootstrapping algorithm to refresh the noise during computation.

D. HARDWARE ACCELERATION

PrivFT by Al Badawi et al. [78] is a privacy-preserving adaption of fastText [79] for text classification. fastText is a model for text classification. PrivFT proposes the following privacy-preserving solutions: (1) using a plaintext model and encrypted inputs for inference and (2) using an encrypted dataset for training an encrypted model. To speed up inference, the authors implement a Residue Number System (RNS) variant of CKKS [80] over GPUs. In an RNS, integers can be represented as the residuals (remainders) with respect to two coprime integers. In HE, this can be used to keep the coefficients of polynomials small, which speeds up computation.

Al Badawi et al. [81] implement and evaluate convolutional NNs (CNNs) over HE ciphertexts using the BFV scheme on graphic processing units (GPU) to speed up the computation. The authors use plaintext space Chinese remainder theorem (CRT) decomposition for deeper networks to avoid overflows. CRT allows the authors to split the plaintext into multiple smaller values with respect to some primes. It should be noted that such use of CRT requires frequent swapping out of GPU memory.

E. PROBLEMS WITH HIGH MULTIPLICATIVE DEPTH

Podschwadt and Takabi [82] propose a privacy-preserving text classification solution using word embeddings and

TABLE 2. Strengths and weaknesses. + indicates a strength and − a weakness. A blank field means a solution is neither strong nor weak in this area. And ? means the information is missing in the study. For a link to the implementation, see Table 3.

Study	Overhead	Throughput	Model Performance	Improved Computation	Hardware Acceleration	Memory	Bandwidth	Client Privacy	Server Privacy	Crypto Parameters	Usability	Code Available	Comments
Al Badawi et al. [81]	-	+	-		+	+				+		-	
Nandakumar et al. [85]	-	-	+				?	+	+	+		-	Model training on encrypted data
Bourse et al. [60]	+	-	+	+		?	+	+		+		+	Unclear generalizability
Chabanne et al. [62]	-	-	+			?	?	?		?		-	Unknown crypto parameters
Faster Cryptonets [66]	-	+	-			-		+		+		-	Performance drops on CIFAR-10
	-	+	?			-	-	+	-	+		-	Transfer learning approach
CHET [87]	+	+	+				?	+		+	+	-	Automatic optimizations
Cryptonets [61]	-	+	+			+		+		+		+	
CryptoDL [63]	-	+	+			+	+	+		?		+	
E2DM [68]	+		+	+		?	+	+		+		+	
SHE [65]		-	+	+		-	+	+		+		+	
Brutzkus et al. [69]		-	+	+		+	?	+		+		+	
ngraph-he [86]	-	+	-			-	?	+		+	+	+	
Mihara et al. [75]	-		+	+		+	?	+		+	+	-	Small evaluation task, model training on encrypted data, uses client interaction
Podschwadt & Takabi [82]	-		+			+	-	+	-	-		+	Requires client interaction during computation
PrivFT [78]		+	+		+	+	-	+		-		-	
	-	+	?		+	+	-	+		-		-	Model training on encrypted data with a small number of epochs
CryptoRNN [84]		?	+				-	+	-	-		-	Requires client interaction; small evaluation task
RNN Blocks [83]		+				-	-	+		?		+	
Zhang et al. [64]	-	+	?			?	?	+	-	?		-	
Lee et al. [70]	-		+	+			?	+	+	+		-	Uses bootstrapping; ReLU function
Jang et al. [76]	-		+	+			?	+	+	+		-	Uses bootstrapping; GRU

RNNs, which are known to have high Multiplicative Depth. However, the authors outsource the embedding operation to the client, which requires sharing the embedding layer with the client. Furthermore, interactions between the client and server are required to reset the noise. In a later study, the authors enhanced their solution [83] to eliminate such interactions. They introduce a new architecture, dubbed parallel RNN blocks, which reduces the multiplicative depth of the problem by splitting the input sequence into smaller chunks and processing them in parallel. Bakshi & Last [84] propose CryptoRNN for RNN privacy-preserving classification. However, this approach requires client-server interactions for noise removal, although it only uses small datasets. Nandakumar et al. [85] propose a privacy-preserving solution for training fully-connected NNs on encrypted data, which is also known to possess high Multiplicative Depth. The trained model is also encrypted with the data owner's key; only the encrypted network is accessible by the server. Training data and network parameters are all in binary representation; the authors only encode a single bit, although the ciphertext can hold more information. The authors use bootstrapping to refresh the ciphertexts to keep the noise under control. Bootstrapping is necessary after each layer in the forward and backward pass. The downside of the approach is the large overhead brought on by bootstrapping and inefficient data encoding.

F. USEABILITY

Boehmer et al. [86] develop nGraph-HE, a backend for Intel's nGraph graph compiler. nGraph-HE takes existing HE-friendly NNs from popular ML frameworks like TensorFlow or PyTorch and translates them into HE operations. The goal is to require as little knowledge about HE as possible. nGraph-HE can perform several optimizations that speed up the NN inference. CHET is an optimizing compiler for tensor circuits over HE by Datathri et al. [87]. CHET's optimizations are encryption parameter selection, data layout selection, rotation key selection, and fixed-point scaling factor selection. The compiler uses a cost model to find the most efficient operations and crypto parameters.

IV. NEURAL NETWORK (NN) LAYERS WITH HOMOMORPHIC ENCRYPTION (HE)

Designing an NN architecture depends on the task and the data. It is not a simple task and requires domain knowledge and experience, even on plain data. Encrypted data adds another layer of complexity to architecture design. The architecture needs to fit the task and data; it needs to be within the constraints of the focus HE scheme. In this section, we discuss common NN layers and their applicability to encrypted data. Table 4 presents a list of studies that provide solutions to make common NN layers encryption-friendly.

TABLE 3. Links to available implementations.

Study	Link
Bourse et al. [60]	https://github.com/mminelli/dinn
Cryptonets [61]	https://github.com/microsoft/CryptoNets
CryptoDL [63]	https://github.com/inspire-lab/CryptoDL
E2DM [68]	https://github.com/K-miran/HEMat
SHE [65]	https://github.com/qianlou/SHE
Brutzkus et al. [69]	https://github.com/microsoft/CryptoNets
ngraph-he [86]	https://github.com/NervanaSystems/he-transformer
Podschwadt & Takabi [82]	https://github.com/inspire-lab/CryptoDL
RNN Blocks [83]	https://github.com/inspire-lab/CryptoDL

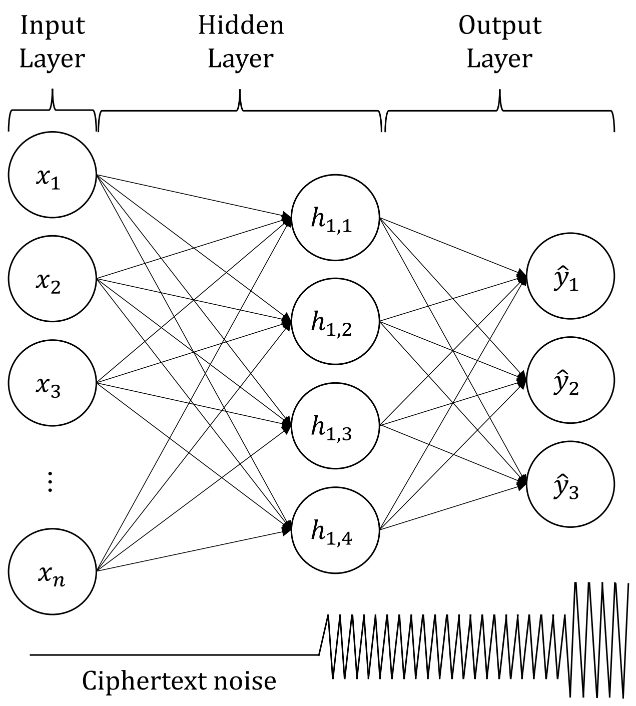


FIGURE 3. A multi-layer perceptron with one hidden layer. On encrypted data, the ciphertext noise grows as computation progresses into the lower layers.

A. FULLY CONNECTED AND CONVOLUTIONAL LAYERS

Fully Connected layers are the simplest forms of NN layers (Fig. 3). The fundamental building block is a neuron or unit (Fig. 4). Given inputs x_i, weights w_i, a bias b and an activation function f, an output, $\hat{y} = f(b + \sum_{i=1}^{n} w_i x_i)$, is computed. Instead of computing the output of every neuron individually, for the neurons of a layer, the output values can be computed using simple matrix multiplication. Previous studies summarized in Table 4 all support fully connected layers.

Unlike fully connected networks, CNNs often include various NN layers, such as convolution, pooling, and batch normalization. Fundamentally, fully connected and convolutional layers are similar. In a convolutional layer, a window is moved across the input data (Fig. 5). Each cell in the window has a weight associated with a weight. The output of the window is the weighted sum of all elements in

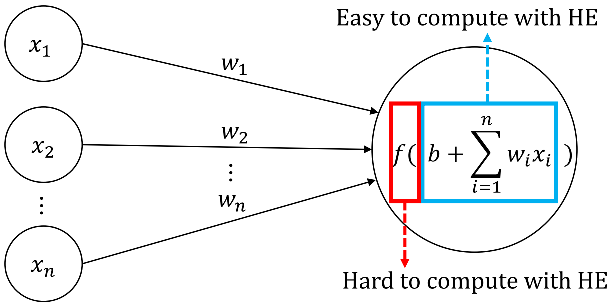


FIGURE 4. Calculating the output value of a single neuron using the inputs x_i, weights w_i, bias b, and activation function f. In most homomorphic encryption schemes, evaluating f is hard.

TABLE 4. A list of common layers and which studies support them on encrypted data.

Study	Fully Connected	Convolutions	RNN	GRU	LSTM	Batch Normalization	Max Pooling	Average Pooling	Scaled Average Pooling
Al Badawi et al. [81]	●	●	-	-	-	-	-	●	-
Nandakumar et al. [85]	●	-	-	-	-	-	-	-	-
Bourse et al. [60]	●	-	-	-	-	-	-	-	-
Chabanne et al. [62]	●	●	-	-	-	●	-	●	-
Faster Cryptonets [66]	●	●	-	-	-	●	-	-	●
CHET [87]	●	●	-	-	-	-	-	●	-
Cryptonets [61]	●	●	-	-	-	-	-	●	●
CryptoDL [63]	●	●	-	-	-	●	-	-	●
E2DM [68]	●	●	-	-	-	-	-	●	-
SHE [65]	●	●	-	-	●	●	●	-	-
Brutzkus et al. [69]	●	●	-	-	-	-	-	●	-
ngraph-he [86]	●	●	-	-	-	●	-	-	●
Mihara et al. [75]	●	-	-	-	-	-	-	-	-
Podschwadt & Takabi [82]	●	-	●	-	-	-	-	-	-
PrivFT [78]	●	-	-	-	-	-	-	-	-
CryptoRNN [84]	●	-	●	-	-	-	-	-	-
RNN Blocks [83]	●	-	●	-	-	-	-	-	-
Zhang et al. [64]	●	-	-	-	-	-	-	-	-
Lee et al. [64]	●	●	-	-	-	●	-	●	-
Jang et al. [76]	●	-	-	●	-	●	-	●	-

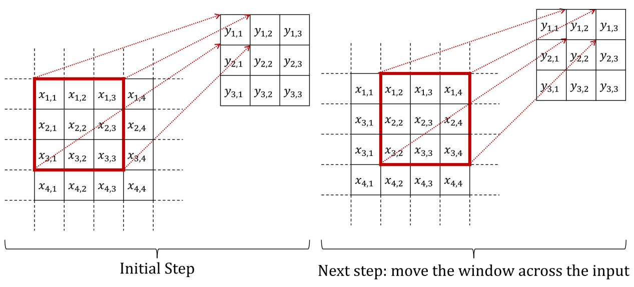

$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$
$y_{1,1}$ $y_{1,2}$ $y_{1,3}$
$y_{2,1}$ $y_{2,2}$ $y_{2,3}$
$y_{3,1}$ $y_{3,2}$ $y_{3,3}$
Initial Step
Next step: move the window across the input

FIGURE 5. Connection between the input and the output for convolutions and pooling layers. The two differ in the function applied to values in the window.

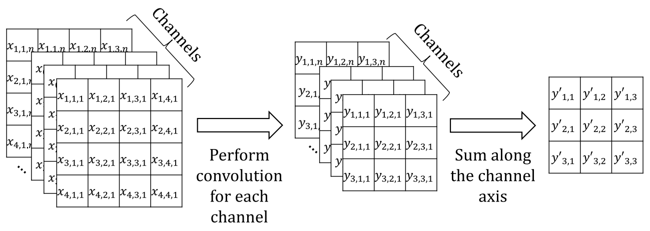

Channels
Perform convolution for each channel
Channels
Sum along the channel axis

FIGURE 6. **Convolutions on inputs with multiple channels.**

the window. For multiple channels, the window is moved across all channels, and the output of each cell is summed up along the channel dimension (Fig. 6). Like fully connected layers, convolution layers can be expressed as matrix multiplications and dot products. Matrix multiplication and dot products are HE-compatible since they only consist of additions and multiplications. No work needs to be done to adapt them to HE. However, a naive implementation can be costly; every element of a matrix/vector is a single ciphertext. The naive implementation of d-dimensional matrix multiplication requires d^3 multiplications and the dot product of two d-dimensional vectors requires d multiplications.

One strategy for speeding up computation is to reduce the number of operations needed. SHE [65] replace matrix multiplication and dot product with shift operation to reduce the number of multiplications. When a number is represented in binary format, as in SHE, multiplying it by two can be achieved by shifting the decimal point right by one. In fact, multiplying by any power of two is simply shifting the decimal point. Shift operation requires quantizing the NN's weights to be a power of two and representing the NN's inputs as fixed-point numbers in their binary representation. Faster CryptoNets [66] uses a very similar approach. The authors also quantize the NN's weights to be a power of two, giving them a sparse polynomial (mononomial) representation. Multiplication between ciphertexts and these sparse plaintexts is much faster. Research on plaintext [88] suggests that weight quantization does not sacrifice accuracy.

Another option is using the SIMD operations offered by many schemes. The goals are to use fewer ciphertexts, decrease latency, and organize the data to speed up computation. This organization is called ciphertext packing. It is essential to have efficiently packed ciphertexts since it can drastically reduce the number of operations needed to evaluate specific algorithms. For example, E2DM [68] includes an encoding map that allows packing one or more matrices in a single ciphertext for efficient matrix multiplications. PrivFT [78] uses a packing structure that encodes the input in a relatively small number of ciphertexts. For efficient dot products, the embedding matrix is packed vertically. Brutzkus et al. [69] propose packing schemes for efficient convolutions and matrix multiplications. The packing schemes are memory efficient since they use fewer ciphertexts. The authors propose algorithms for switching between packing schemes depending on the operation, i.e., convolutions or matrix multiplications. Lee et al. [70] propose a packing scheme for strided convolutions. In contrast to E2DM and PrivFT, the CHET compiler [87] can automatically select packing schemes depending on the input model, HE scheme, and data domain by introducing a cost model. Mihara et al. [75] develop a diagonal ciphertext packing scheme for efficient matrix transpose operation; it boosts the training backpropagation speed.

The primary challenge of ciphertext packing schemes is that they need to work efficiently with data for operations such as convolutions or matrix multiplications; there is no unique packing scheme for all data/operations. Besides, except for CHET [87], other techniques such as E2DM and PrivFT require trial and error to identify efficient ciphertext packing schemes. Furthermore, packing often requires rotations of the slots. Switching packing schemes, which often requires rotations, impacts the overall NN runtime. Additionally, the transformation from one representation to another needs to be considered when calculating the noise budget.

Most HE schemes support SIMD operations except for TFHE; it cannot be used in these packing schemes.

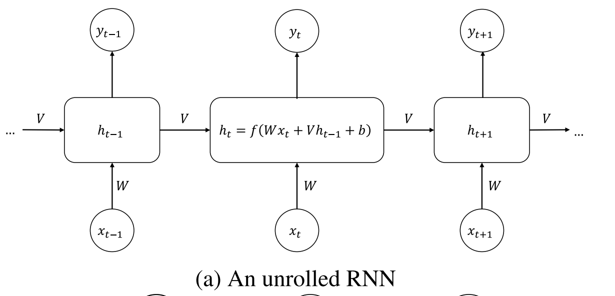
B. POOLING LAYERS

Pooling layers are often crucial to the success of CNNs. Pooling layers, like convolutional layers, work by moving a window across the data (Fig. 5). The output of the window is the pooling function, which is applied to all values in the window. The most common pooling layer is max-pooling, where the pooling function is the maximum of all values in the window. However, it is inefficient with HE schemes except SHE [65]; it benefits from the TFHE scheme. Some studies replace the max-pooling operation with either average [62], [81], [87] or scaled (by a constant factor) average pooling [61], [63], [64], [66], [86]. The most common scaling factor is the number of input elements; it turns average pooling into input summations without any multiplication (i.e., plaintext division) operation. The advantage of using average pooling is the tighter control over the magnitude of the values. The parameters of the HE scheme need to be chosen so that all values that occur during computation fit into the plaintext space. With average pooling, the output of the pooling operations is in the same range as the inputs. With scaled average pooling, the output values can grow larger. The cost of using average pooling is an additional multiplication.

Al-Badawi et al. [81] show that pooling is not necessary on simple data sets such as MNIST, but on more complex data sets such as CIFAR-10, it does improve performance. Usually, studies that utilize MNIST do not implement pooling layers in their NNs.

C. BATCH NORMALIZATION LAYERS

Batch normalization stabilizes the training process on plain data by reducing the internal covariate shift. It forces the inputs to a layer to follow a zero-mean normal distribution. This normalizing operation makes inputs with large absolute values less likely. Chabanne et al. [62] used batch



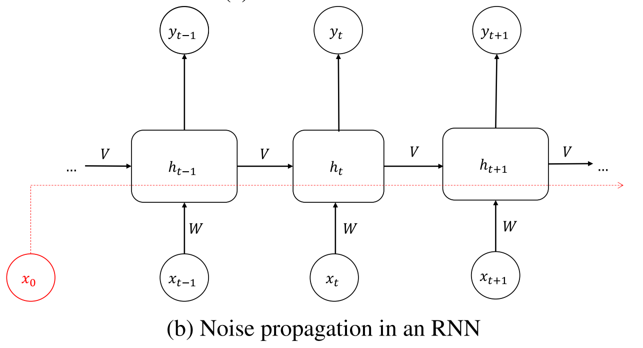


FIGURE 7. **Overview of a simple recurrent neural network (RNN) layer 7a "unrolled" along the time dimension *t*. The red dotted line in 7b shows the initial input value passing x_0 through every unrolled layer.**

normalization to stabilize NN training with polynomial activation functions. Polynomial activations usually work well in a small interval around zero. Outside of that interval, they are unbounded and have rapidly growing derivatives during training (exploding gradient problem). By placing a batch normalization before a polynomial activation layer, the probability of encountering values outside the optimal polynomial activation range decreases. In addition to stabilizing training, it helps prevent values from overflowing the limits of the plaintext space. This approach is used in many studies [63], [64], [65], [66], [86].

D. RECURRENT LAYERS

There is little work on PPML using recurrent layers. Simple recurrent layers (Fig. 7a) at their core are similar to fully connected layers; they only consist of matrix multiplication. The depth of recurrent layers depends on the number of elements in the input sequence (Fig. 7b). Recurrent layers often have higher multiplicative depth (i.e., more noise), requiring numerically larger crypto parameters than convolutional or fully connected layers. Noise buildup prevents the network from completely processing the inputs. Large crypto parameters require more computation and memory resources. To alleviate these challenges, Podschwadt and Takabi [82] and Bakshi & Last [84] (CryptoRNN) propose to use the client to remove the built-up noise in the ciphertext. CryptoRNN uses simple RNNs with client communication too. They refresh noise at three predefined points in the networks: 1) after every multiplication, 2) before every non-linear activation function, and 3) after processing each sequence element. It should be noted that at point (2), the client also computes the activation functions. Similarly, Podschwadt and Takabi [82] use naive matrix multiplication and a degree three polynomial Tanh approximation in simple RNNs. The authors dynamically decide when to use the client for interactive noise removal depending on the remaining noise budget (which prevents unnecessary communication) instead of at predefined points.

In a later study, Podschwadt and Takabi [83] propose RNN Blocks, an RNN architecture that does not need client interaction. It reduces the multiplicative depth by splitting each instance into multiple shorter chunks across the time dimension. Any recurrent operation on these chunks is parallelized, decreasing the multiplication depth significantly due to shorter chunks. The RNN produces an output for each chunk; they are then concatenated and fed to a fully connected layer.

Later recurrent layers such as long short-term memory [89] (LSTM) (Fig. 8), or gated recurrent units [77] (GRU), impose even higher multiplicative depth due to their complex structure. SHE [65] modified the LSTM structure to use the ReLU function, which is comparatively easier to compute using TFHE than Tanh. Typically, RNNs use the Tanh function, which requires at least a polynomial of degree three for an adequate approximation. The authors also replace all the multiplications with shift operations, which they can perform at no cost to the noise budget.

Jang et al. [76] implement GRUs on encrypted data using their proposed encryption scheme MatHEAAN. The internal structure of a GRU is similar to that of an LSTM and introduces high multiplicative depth. To address this issue, the authors rely on bootstrapping during the computation of the network.

RNN architectures are not well suited for execution over homomorphically encrypted data [83]; they require either fundamental changes as seen in SHE [65], and RNN Blocks [83], bootstrapping [76], or establishing client communication as seen in [84] and [82]. However, interactive communication takes away some HE key advantages over Secure Multiparty Computation, such as the lower communication overhead and the ability to perform the computation independently from the client.

The primary strength of RNNs is that they have some "memory" of previous states. However, it is their downfall on encrypted data since this "memory" produces networks with a large multiplicative depth.

E. EMBEDDING LAYERS

Word embeddings are real-valued vector representations of words' meaning in natural language processing; they are useful tools since they turn words into real values (instead of integers) to be later fed to NNs. There are different word embedding methods. PrivFT [78] relies on the fasttext [79] architecture, where the words need to be encoded into "one-hot" vector representations. In a one-hot vector representation, all elements are zero except one. In PrivFT, the client does the one-hot encoding. On plaintext, the translation from

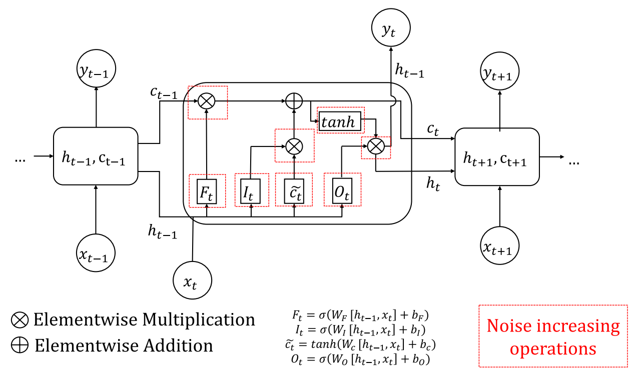

y_t
y_{t-1}
h_{t-1}
y_{t+1}
c_{t-1}
$tanh$
c_t
… h_{t-1}, c_{t-1}
h_{t+1}, c_{t+1} …
h_t
F_t
I_t
$\tilde{c}_t$
O_t
h_{t-1}
x_{t-1}
x_t
x_{t+1}
⊗ Elementwise Multiplication
⊕ Elementwise Addition
$F_t = \sigma(W_F\,[h_{t-1}, x_t] + b_F)$
$I_t = \sigma(W_I\,[h_{t-1}, x_t] + b_I)$
$\tilde{c}_t = tanh(W_c\,[h_{t-1}, x_t] + b_c)$
$O_t = \sigma(W_O\,[h_{t-1}, x_t] + b_O)$
Noise increasing operations

FIGURE 8. Internals of a long-short term memory (LSTM) cell with noise-increasing operations highlighted.

a word to one-hot encoding would happen on the server. However, this operation is prohibitively costly over encrypted data. The client sums up all the one-hot encodings, encrypts the results, and sends the ciphertext to the server alongside the number of encoded words. The client transmits this number of words in plain. The server multiplies the vector with the embedding matrix and scales the results with the number of encoded words. Podschwadt and Takabi [82], [83] also work with textual data. In their work, they outsource the embedding to the client. The client performs the embedding operation (a simple table lookup) and sends the encrypted result to the server for computation.

In all approaches above, the server needs to share some information about the model and the data with the client. They all need to provide the client with an enumerated vocabulary of all the words known to the model. This only leaks very little information. In Podschwadt and Takabi [82], [83] studies, the client also needs to access the learned embeddings in plaintext. This plaintext access is not a problem in their later work RNN Blocks [83], where they use publicly available, pretrained embeddings. However, using publicly available embeddings prevents the server from using fine-tuned or self-learned embeddings and keeping them secret from the client. Besides, embeddings further increase the size of the data. Embeddings often turn a single word into a vector of 100+ real values, increasing size during encryption.

V. ACTIVATION FUNCTIONS

PPML approaches are often different from standard ML in their choice of activation functions. The choice of activation often depends on the scheme. With some HE schemes, one can use popular NN activation functions. For example, SHE [65] uses the TFHE scheme, enabling the user to implement ReLU on encrypted data using binary gates. Bourse et al. [60] use a step function that outputs either 1 or -1 based on the sign of the input value. This sign function can be evaluated using a custom bootstrapping operation in TFHE. Every activation function evaluation is also a bootstrapping operation, allowing unlimited deep networks. However, training a network with the step function is difficult since it does not provide gradient information. Another option for using standard activation functions is client-side computation. CryptoRNN [84] outsources the computation of the activation function to the client, allowing the authors to use arbitrary activation functions.

In other schemes, low-degree polynomials are popular choices for activation functions since most schemes can easily evaluate polynomials. The simplest non-linear polynomial, the square function $f(x) = x^2$, is often used as a replacement for ReLU [61], [63], [64], [68], [69], [75], [81], [84], [86]. The square function is relatively fast and adds little overhead to the computation. However, it does not perform well in all problem domains. For example, using it in RNNs is infeasible due to its rapidly growing derivative [82]. The square function can not be used to replace ReLU in an already trained network since their outputs are too different.

More complex polynomials can reduce the approximation error. The Stone-Weierstrass theorem [90] states that any continuous function can be approximated with arbitrary precision over a closed interval using polynomials. However, accurate approximation requires high-degree polynomials, which introduce large computational overhead and noise. One needs to find a trade-off between approximation accuracy and polynomial degree for HE solutions.

Chabanne et al. [62] study polynomials with even degrees two, four, and six as approximations of the Relu function. They find the approximations by applying the "polyfit" method of the numpy[1] package to the output of ReLU on the standard normal distribution. The authors use this distribution since they put a batch normalization before every ReLU activation function. Combining polynomial activation functions with batch normalization can also help during training; polynomial activation functions' unbounded derivatives can lead to exploding gradients, which are counteracted by batch normalization. Hesamifard et al. [63] propose a method for finding approximations of common activation functions. The authors find a polynomial approximation for the function's derivative using Chebyshev polynomials [91]. By integrating that function, they can find an approximation for the activation function. The intuition behind their approach is that the derivative of the activation function plays a large role in training the NN, and approximating it more closely leads to better results. Their approximation approach is used by Podschwadt and Takabi [82], and RNN Blocks [83] to approximate the Tanh function, with degree three polynomials, for the use in RNNs.

Lee et al. [70] approximate the ReLU functions as $\mathrm{ReLU}(x) = \frac{1}{2}x(1 + \mathrm{sign}(x))$. They use a composition of polynomials [92] to approximate the sign function. In their experiments, they use three polynomials of degrees 7, 15, and 27. This allows for a very accurate approximation of ReLU that can replace the activation in a trained NN without the need for further fine-tuning. However, the high degree of the polynomials requires bootstrapping to control the noise.

[1]https://numpy.org

The authors further propose a method for approximating the softmax function:

$$\text{softmax}(x) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}; \quad \text{for each } i = 0, 1, \cdots, n$$

Using the least-squares method, the authors use a degree 12 polynomial to approximate the exponential function. To compute the inverse function, the authors use Goldschmidt division [93], allowing them to approximate the division as a series of multiplications. Al Badawi et al. [78], use minimax approximation [94] to find a polynomial approximation for the softmax function. Minimax approximation is to minimize the maximum error. Using this approach, the authors find $1/8x^2 + 1/2x + 1/4$ as an approximation of the softmax function. Since the softmax function is usually the last operation in an NN, most studies leave its evaluation to the client. However, this poses a risk since directly exposing the logits of the NN to the client makes it more vulnerable to adversarial and model inversion attacks.

CHET [87], nGraph-HE [86] and Jang et al. [76] use a different approach to finding polynomial approximations. Rather than numerically finding an approximation, in polynomials of the form $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2x^2 + a_1x$, the coefficients a_i are learned during the NN's training. CHET and nGraph-HE learn polynomials of degree two while Jang et al. [76] train polynomials with degree seven, which requires bootstrapping during inference.

Nandakumar et al. [85] do not use polynomial approximations for the sigmoid activation function; they use a lookup table instead [95]. The client needs to precompute the table before running the network.

VI. ADAPTIONS TO HE CONSTRAINTS

A. DATA ENCODING

Most HE schemes support only integer messages. CKKS supports real numbers, while TFHE only supports individual bits. Most plaintext ML applications work on either 32 or 64-bit floating-point numbers. This means that solutions using CKKS (see Table 5) can use real numbers. However, since computation in CKKS is approximate, these solutions must correctly choose the precision parameter to ensure accurate classification on encrypted data.

Solutions based on schemes other than CKKS need to adapt to the reduced message space. CryptoDL [63] chooses an integer-only solution. They scale the weights of the network and round them to be integers. Al Badawi et al. [81] use a similar approach. They also use integers as inputs but use CRT to decompose the values into multiple parts. This decomposition keeps individual values smaller and thereby leaves room for intermediate values. Large intermediate values can become a problem in CryptoDL [63]. The authors need to make sure that intermediate values never overflow the message space by choosing the crypto parameters large enough.

Many solutions use fixed-point encoding to address the limited message space [61], [62], [64], [65], [66], [85], [87]. Fixed-point encoding suffers from the same problems that other integer-only solutions suffer, mainly the overflow of the message space.

SHE [65], and Nandakumar et al. [85] encode the fixed-point number in binary representation despite using different encryption schemes. In the case of SHE [65], this is due to the limitation of the encryption scheme. However, Nandakumar et al., [85] use the binary representation to use a lookup table as an activation function. However, this binary representation leaves a lot of the ciphertext's capacity unused, as it can hold much larger numbers than a single bit.

Bourse et al. [60] have an even more radical encoding approach. The authors map all input values to either -1 or 1 to fit their computation scheme. They do this by dividing the original message space in half and mapping everything equal to or larger than the mean to 1 and everything smaller to -1. However, this mapping loses a significant amount of information. While all other schemes have to deal with small losses due to their encoding or approximate computation, the authors reduce the information by over 99%. In our opinion, this is due to using MNIST in the experiments. The MNIST images are already high contrast, greyscale and have very few color gradients. Removing 7 of the 8 bits of information should not harm the recognizability of the digits. The authors note: "...a quick visual inspection of the result shows that the digits depicted in the images are still clearly recognizable." It is questionable if this would hold for other datasets than MNIST.

B. WEIGHT CONSTRAINTS AND CONVERSION

Some studies place further constraints on the weights or their encoding to often help speed up computation. As discussed in the previous section, solutions that work on integers or fixed-point encoding need to transform their weights into integers. This transformation is done by applying a scaling factor to the weights of each layer. However, for each layer, the scaling factor needs to be carefully computed since, depending on the degree of the polynomial activation function, the neuron values might have already been significantly magnified due to the previous layer's scaling factor. The scaling can cause the weights to grow large in the last network layers, causing overflow problems.

Other studies use varying forms of quantization. Chou et al. [66] use quantization to enforce sparsity in the polynomial representation of the weights, speeding up the computation. Zhang et al. [64] use Lloyd-max quantization [96] of the network's weights during training. Lloyd-max quantization quantizes the network's weights based on their local density instead of uniform intervals. Although Lloyd-max quantization increases the network's predictions quality on fixed-point numbers, it does not speed up computation like in other solutions.

SHE [65] uses fixed-point numbers in their binary representation. To speed up NN computation, the authors use log-quantization to transform all weights into powers of 2. The quantized weights allow them to replace the multiplications with shifts and accumulators. Shifting operations are fast since they do not require an operation on the encrypted data itself. Bourse et al. [60] not only discretize the inputs as either -1 or 1 but also the weights. The discretization of inputs and weights allows the authors to run an NN with all binary values, speeding up computation.

C. CONSTANT FOLDING

Reducing the number of operations increases performance. Several studies include constant folding to reduce the number of operations. Constant folding is to combine constant factors of successive operations into one. Boehmer et al. [86] describe how to fold some factors of the activation functions into the previous layer's weights. When a polynomial activation layer follows a convolutional or fully connected layer, one of the polynomial coefficients can be moved to the previous layer's weights, removing one ciphertext-plaintext multiplication. Similarly, the authors show how to fold a batch normalization layer into the previous layer. The average pooling and linear layers (e.g., convolutional layer) can be collapsed or folded [61]. Boemer et al. [86] replace the average pooling with a scaled average pooling layer when an average pooling layer follows a linear layer. The authors move the scaling factors into the weights of the linear layer.

D. CLIENT-SIDE COMPUTATION

Some solutions offload work to the client to work around some of the limitations of HE. Chou et al. [66] and Brutzkus et al. [69] propose using a transfer learning-like approach. Chou et al. [66] use the topmost layers of an NN already trained on a general task and fine-tune the lower layers on a specific task. The upper layers are sent to and evaluated by the client on plaintext. The client encrypts the upper layers' output and sends it to the server, which evaluates the lower layers. Brutzkus et al. [69] use a similar technique where the client runs data through a pre-trained NN. The NN's output is encrypted and sent to the server for further processing. This output is smaller than the original inputs and can be classified using shallow NNs. The client-side NN works as a form of dimensionality reducer. While both approaches reduce the computation the server needs to perform on encrypted data, they place a higher computational burden on the client. Podschwadt and Takabi [82], and RNN Block [83] use a somewhat similar approach in which they send the embedding part of the NN to the client for the embedding operation. Compared to Chou et al. [66] and Brutzkus et al. [69], this is less computationally expensive for the client but requires a more extensive data transfer since the embedding process increases the dimensionality of the data. The client can perform these computations in a setup phase independent of the server.

Several authors involve the client more deeply in computation by establishing interactive protocols. Interactive protocols require multiple rounds of client-server communication to complete the computation. For example, CryptoRNN [84] uses the client to compute the activation functions. The server sends the preactivation values to the client. The client computes the activation function on plaintext and sends the encrypted result back to the server. Since the client decrypts and encrypts the data, this operation also refreshes the noise. The authors also study a different protocol in which the client only performs noise removal. The noise removal happens at fixed points in the NN: after every multiplication, after every activation function, or after every sequence element. While this does not allow the use of standard activation functions (e.g., Tanh), it can help preserve the privacy of the NN. Before sending the data to the client, the server can add a random value which it can subtract later. This random value is to preserve the privacy of the NN; without it, the client would learn the internal state of the NN computation, e.g., reconstructing the weights. Adding such a random value is impossible when the client computes the activation function; the activation function cannot be reversed to remove the random value. Podschwadt and Takabi [82] use a similar approach. However, they do not use fixed points to refresh the noise. The authors monitor the noise and dynamically decide when to refresh, avoiding unnecessary communication. Client communication is usually a replacement for bootstrapping, as noted by Mihara et al. [75]. In their study, the authors use the client to refresh the noise during training. They opt to send the data to the client after every minibatch during training instead of using bootstrapping. The authors show that replacing bootstrapping with interactive protocols usually saves computation time.

E. BOOTSTRAPPING

Bootstrapping removes the depth restrictions that HE solutions face. Although interactive protocols can alleviate these problems, too, they introduce network latency and can lead to information leakage. Nandakumar et al. [85] use bootstrapping during the training to refresh the noise. However, they need to perform bootstrapping after every layer in the forward and backward pass, adding significant overhead. Unlike CryptoRNN [84] and Podschwadt and Takabi [82], Jang et al. [76] propose a communication-free RNN using bootstrapping during inference for noise refresh. The authors measure the noise budget after processing every sequence element and perform bootstrapping if necessary. Lee et al. [70] use bootstrapping to run the comparatively deep ResNet-20 architecture. Using leveled FHE, the crypto parameters would be prohibitively large. The authors utilize bootstrapping operations during ReLU polynomial approximation and after convolutional operations. Bootstrapping after convolution operations reduces the number of rotations required by the packing scheme and speeds up the computation. Furthermore, the authors do not use all the slots in a ciphertext since they find that sparsely packed ciphertexts are faster to bootstrap.

F. BATCHING

In Section IV-A, we already discussed ciphertext packing using SIMD processing. Another way of using SIMD processing is to process multiple instances simultaneously, so-called batching, at no additional cost. A common form of batching is to encrypt the same feature of multiple instances into a single ciphertext, leading to one ciphertext per feature. Since ciphertexts are usually orders of magnitude larger than plaintexts, this form of batching requires a lot of memory. It is also less efficient for a single instance or small batch processing. Supported batch sizes are usually large (> 1000), and the memory and runtime requirements are independent of the number of instances in a batch. It makes no difference if 1 or 1000 instances are being processed. Batching typically has a decent amortized time per instance. Still, the latency is relatively large, making it more attractive for bulk processing and less attractive for close to real-time applications. CryptoNets [61] uses this approach with a batch size of 4096. CryptoDL [63], and Al Badawi et al. [81] support a batch size of 8192. Podschwadt and Takabi [82] use a comparatively small batch size of 128 for their works on RNNs. In contrast, their RNN Blocks work [83] uses batch sizes of up to 32,768. Nandakumar et al. [85] also work with smaller batch sizes of 60 or 1800 instances, depending on the parameters.

G. TRAINING ON ENCRYPTED DATA

Even with powerful computational resources available, PPML training needs to be severely constrained to be somewhat practical. It needs interaction with the client, as used by Mihara et al. [75], or the task needs to be simplified. Typically, on plain data, models train for many epochs. Al Badawi et al. [78] need to constrain the training to two epochs with large minibatch sizes. Nandakumar et al. [85] circumvent this problem by using bootstrapping. However, bootstrapping is so expensive that the authors need to simplify the problem drastically; the NN model is relatively small, and they subsample the input data from 28×28 to 8×8. Even with these simplifications, training one minibatch takes 40 minutes. With the parameter settings used in the paper, a minibatch contains 60 instances. The MNIST dataset contains 60,000 training instances. Using this configuration, one epoch of training would take 666.7 hours or almost 28 days.

VII. SECURITY

A. ENCRYPTION SCHEMES

Except for the YASHE scheme, most schemes rely on the *Learning with Errors* or its ring variant *Ring Learning with Errors*. *Learning with Errors* is the problem of finding a linear function with multiple inputs given a number of function outputs. However, some of the outputs may be noisy. *Ring Learning with Errors* is the extension of Learning with Error to polynomial rings. YASHE scheme is based on NTRU (Number Theorists are US); its security can be compromised by subfield lattice attacks [51]. In addition to enabling FHE, these problems are quantum hard. (Ring) Learning with Errors can be reduced to worst-case problems over ideal lattices. By definition, FHE schemes can not be secure against chosen-ciphertext attacks (CCA). Further, Li et al. [97] show that the CKKS scheme is not secure against chosen-plaintext attacks (CPA). An adversary can draw conclusions about the secret key from the errors in a decrypted plaintext introduced by the scheme's approximate nature. HElib [56], for example, circumvents this problem by adding key-independent noise to the plaintext during decryption. However, this approach further reduces the accuracy of decrypted values.

The underlying hardness assumption of prevalent schemes such as BGV, BFV, CKKS, and TFHE (Table 5) is known to be secure. However, CKKS has a vulnerability that does not stem from the hardness assumption but from the noise management in the scheme.

TABLE 5. HE schemes and libraries used by the different studies. * added CKKS bootstrapping. ** own implementation based on NTL (https://libntl.org/).

Study	Scheme	Library
Cryptonets [61]	YASHE	SEAL
Chabanne et al. [62]	BGV	HElib
Nandakumar et al. [85]	BGV	HElib
CryptoDL [63]	BGV	HElib
Al Badawi et al. [81]	BFV	SEAL, A*HE
Faster Cryptonets [66]	BFV	SEAL
Brutzkus et al. [69]	BFV	SEAL
Encrypted Speech [64]	BFV	SEAL
CHET [87]	CKKS	SEAL, HEEAN
E2DM [68]	CKKS	HEEAN
CryptoRNN [84]	CKKS	SEAL
Podschwadt & Takabi [82]	CKKS	HElib
RNN Blocks[83]	CKKS	HElib
PrivtFT [78]	CKKS	SEAL
Mihara et al. [75]	CKKS	SEAL
ngraph-HE [86]	CKKS	SEAL
Lee et al. [70]	CKKS	SEAL*
Jang et al. [76]	CKKS	**
Bourse et al. [60]	TFHE	TFHE
SHE [65]	TFHE	TFHE

TABLE 6. HE libraries and which schemes they support. The Bootstrapping column shows if the library supports bootstrapping. The GPU column shows support for GPU hardware acceleration.

Library	Supported Scheme				Bootstrapping	GPU
	BGV	BFV	CKKS	TFHE		
SEAL	•	•	•			
HElib	•		•		• (BGV only)	
HEAAN			•		•	
TFHE				•	•	
A*HE		•				•

B. INFORMATION LEAKAGE

Information leakage can occur in various directions. For example, either information about the client's data can leak to the server, or the server's model can leak to the client. All approaches in this survey focus on protecting the privacy of the client's data; however, the model's privacy can also be a concern. While the client does not have direct access to the model's parameters, Tramer et al. [99] show that an attacker can reverse engineer a model with access only to the model's predictions. Interestingly, there is more information leakage about the model to the client on encrypted data. One reason is

TABLE 7. Resource requirements, Tasks performed, Dataset Performance, and Runtime reported by different studies.Memory and Communication (Com.) are values reported in the studies. Performance shows the impact of the changes to facilitate encrypted execution. Plain gives the performance of the model on plain data, Enc. the performance of a similar model that is HE-friendly; and Loss is the relative difference.

Study	Task		Dataset	Memory	Com.	Performance			Runtime	BS
	T	I				Plain	Enc.	Loss		
Badawi [81]		•	MNIST	< 16 GB**	0.98 GB	-	96%	-	5.16 s	8,192
			CIFAR-10	< 16 GB**	1.77 GB	-	77.5%	-	304.43 s	8,192
Nandakumar [85]	•	•	MNIST	< 250 GB*	-	96.4%	96%	0.4%	0.7-6.5 h♠	60
Bourse et al. [60]		•	MNIST	-	7.82 KB	96.75%	96.43%	0.3 %	1.65 s	1
Chabanne et al. [62]		•	MNIST	-	-	99.59%	99.30%	0.3 %	-	-
Faster Cryptonets [66]		•	MNIST	< 48 GB*	0.41GB	99.13%	99.17%	-0.04%	45.7 s	1
		•	CIFAR-10	< 1433 GB*	1.57 GB	86.76%	75.99%	12.41%	0.9 h	1
		•	[98]	< 1433 GB*	789.2 GB	-	69.89%	-	-	-
CHET [87]		•	MNIST	< 224 GB*	-	99.3%	99.3%	0 %	35.2 s	1
		•	CIFAR-10	< 224 GB*	-	84%	81.5%	2.9%	164.7 s	1
Cryptonets [61]		•	MNIST	< 16 GB*	0.36 GB	-	99%	-	250 s	4096
CryptoDL [63]		•	MNIST	< 16 GB*	-	99.56%	99.52%	0.04%	320 s	8,192
		•	CIFAR-10	< 16 GB*	-	94.2%	91.4%	2.9%	3.2 h	8,192
E2DM [68]		•	MNIST	-	0.02 GB	-	98.1%	-	28.59 s	64
SHE [65]		•	MNIST	< 1024 GB*	0.12 GB	-	99.77%	-	9.3-124.9 s	1
		•	CIFAR-10	< 1024 GB*	0.16 GB	-	74.1%	-	0.6-3.3 h	1
		•	ImageNet	< 1024 GB*	7.7 GB	-	69.4%	-	5-63.9 h	1
		•	Penn Treebank	< 1024 GB*	0.01 GB	-	89.8 ‡	2.1%♣	576 s	1
Brutzkus et al. [69]		•	MNIST	-	-	-	98.95%	-	0.29-7.29 s	1
		•	CIFAR-10	12 GB	-	-	74.1%	-	730 s	1
ngraph-he [86]		•	MNIST	< 376 GB*	-	-	99%	-	16.7 s	4,096
		•	CIFAR-10	< 376 GB*	-	-	62.10%	-	0.4 h	8,192
Lee et al. [70]		•	CIFAR-10	172 GB	-	92.95%	92.43%	0.5%	2.9 h	1
Mihara et al. [75]	•	•	Iris	-	-	98.05%	98.47%	-0.4%	29.8 h♡	20
Podschwadt & Takabi [82]		•	IMDB	< 32 GB*	15.20 GB	-	86.47%	-	0.5 h	256
RNN Blocks [83]		•	Amazon Reviews	120 − 216 GB	19-100 GB	81.86%	79.46%	3%	2.9 h	32,768
		•	IMDB	120 − 216 GB	19-100 GB	78.53%	74.95%	3%	2.9 h	32,768
PrivFT [78]		•	IMDB	< 16 GB**	0.38 GB	-	91.5%	-	7.9 s	1
		•	YELP	< 16 GB**	0.38 GB	-	96.1%	-	7.8 s	1
		•	AGNews	< 16 GB**	0.38 GB	-	92.5%	-	7.8 s	1
		•	DBPedia	< 16 GB**	0.38 GB	-	98.8%	-	7.74 s	1
	•	•	YoutTube Spam	120 GB	549 GB		86.3%	-	121 h♡	
CryptoRNN [84]		•	Robot Navigation	-	1.6 MB	96.2%	96.2%	0%	0.2 s	-
		•	Wearable sensor	-	2.7 MB	87.7%	87.7%	0%	0.2 s	-
		•	EEG Eye State	-	1.2 MB	72.5%	72.5%	0%	1.1 s	-
		•	Occupancy	-	9.5 MB	99.7%	99.7%	0%	0.8 s	-
Jang et al. [76]		•	MNIST	1 GB	-	94.6%	94.2%	0.4%	0.7 h	-
		•	deepTarget	1 GB	-	89.7%	89.7%	0.0%	-	-
Encrypted Speech [64]		•	Switchboard	-	-	12.2%†	13.5%†	10.7%	-	-
		•	Cortana	-	-	12.9%†	14.8%†	14.7%	373 ms◇	-

Training (T) and **Inference (I)** indicate which task is performed and on which **Dataset**.
Memory shows the tasks required memory, where * indicates the total memory available (an upper bound of the required memory), and ** indicates that authors only report GPU memory and not the total memory.
Communication is the amount of data that needs to be transferred between the client and the server to complete the task. (♣ means reported by the original authors).
Performance values are accuracy; except for † which is word error rate (WER) and ‡ which is Perplexity per Word (PPW). Wall-clock runtime reported by the different studies. ♡ indicates training of the entire network, ♠ training time of a mini batch. ◇ is the average latency per utterance.

that it is common practice to leave the decoding of the results to the client (usually softmax or beam search [64]). These raw values make it easier for the client to learn the network's weights. Another reason is that the client can infer some of the network architecture through the crypto parameters. Especially when using leveled HE, the client gains information about the depth of the computation because the multiplicative depth is directly correlated to the number of layers in NNs. The server can obfuscate this information by choosing larger parameters than the network requires, increasing running time and memory consumption. Alternatively, bootstrapping can deny this information to the client. However, this also increases resource requirements. Additionally, the encoding methods used with ciphertext packing can reveal the model's information.

The information leakage becomes even more severe when the client needs to participate in the computation. CryptoRNN [84] and Podschwadt and Takabi [82] face the problem of revealing internal states (e.g., layer's outputs) to the client during noise removal. Knowing the internal state could allow the client to learn the model's parameters since the client learns the exact number of units used in the hidden layers. Furthermore, with access to the intermediate states, the client gains more information than assumed by most model stealing attacks [100]. It can potentially use this additional information to improve the attack. The server can prevent the client from learning the actual internal state by adding randomness to the data before sending it to the client for noise removal. The server can remove the randomness once the client returns the encrypted ciphertext to the server. This obfuscation becomes more complicated when the client computes nonlinear functions, as in CryptoRNN [84].

Multiple approaches need to share additional information with the client for preprocessing. Podschwadt and Takabi [82], RNN Blocks [83], and PrivFT [78] need to share all the words known to the model. Unless the model operates in a very specific domain with a very specific vocabulary, the client does not gain valuable information. Podschwadt and Takabi [82] and RNN Blocks [83] additionally need to share the embedding matrix. However, this does not leak any information to the client since there are plenty of pretrained embeddings [101] that are publicly available. So, the client does not learn anything new about the model.

When it comes to the security level in bits, many studies opt to use 80-bit [60], [76], [81], [85]. Only Hesamifard et al. [63], RNN Blocks [83], and Faster Cryptonets [66] choose parameters that provide 128-bit security or more. Authors often decide to use a lower security level since it does speed up computation.

VIII. REPORTED EXPERIMENTS

In this section we compare the experimental results presented in the selected studies.

A. TASKS AND DATASETS

The task and data used for evaluating PPML solutions in various studies are essential for comparison. The data significantly influences the network architecture and affects the encoding scheme. For example, it is easier to encode a fixed set of discrete values, such as images, than to encode a range of continuous values. Many high-performing model architectures have been developed for image processing. Images are easily resizable, often with little information loss. Smaller images mean less data to encrypt and fewer input features, resulting in fewer computations within the network and less memory. Changing the dimensions of other data types is not easy. Text, for example, cannot be resized despite it consisting of discreet values. Additionally, models designed for text processing are usually more complex, which results in a higher multiplicative depth. Below we discuss some of the datasets used in various PPML studies (Table 7).

Image classification is the most common task used to evaluate PPML solutions. The MNIST hand-written digit dataset[2] is, perhaps, the most widely used image dataset [60], [61], [62], [63], [65], [66], [68], [81], [85], [87]. It consists of 60,000 training and 10,000 test samples distributed across ten classes. Each sample is a 28 by 28 pixels grey-scale image of a hand-written digit between 0 and 9. However, simply demonstrating a PPML solution works on MNIST is insufficient; it might be good for comparison with similar solutions, but it has the significant downside of being too simple and too clean as an evaluation dataset. Strategies that work on MNIST do not necessarily work on other datasets.

The CIFAR-10[3] dataset is a better choice since it is closer to a real-world dataset than MNIST. It contains 50,000 training and 10,000 test 32 by 32 pixels color images across ten classes. Some solutions, [63], [65], [66], [81], [87], additionally evaluate on the CIFAR-10 data. SHE [65] is the only solution that tackles the much harder Imagenet[4] dataset. Their solution evaluates over a million 224 by 224 pixels color images in 1001 classes from Imagenet. Color images, such as CIFAR-10 and Iamgenet, are more complex since they usually have three channels compared to the single channel in grey-scale images. MNIST images consist of $28 \times 28 = 784$ pixels. In comparison, the only slightly larger CIFAR-10 images contain $32 \times 32 \times 3 = 3072$ pixels, let alone Imagenet; it contains approximately 50 times as many pixels as a CIFAR-10 image, specifically $224 \times 224 \times 3 = 150,528$ pixels.

Text classification is not as common as image tasks to evaluate PPML solutions. IMDB movie review dataset[5] is, perhaps, the most common PPML text evaluation dataset, as seen in Podschwadt and Takabi [82], RNN Blocks [83], and PrivFT, [78]. The dataset contains 50,000 movie reviews grouped into negative and positive reviews.

Zhang et al. [64] and Lou et al. [65] are the only works focusing on tasks other than classification. Classification tasks seem to be easier to handle in the encrypted domain. One explanation is that the NN's output is easily interpretable and of comparatively low dimension. As shown in Zhang et al. [64], the interpretation of the NN output requires much more computation on the client side.

B. RESOURCE REQUIREMENTS

All solutions share the problem of having high resource requirements. Comparing their runtime is difficult since some experiments were performed on laptops and others on big servers with up to a hundred CPU cores. That is why we only highlight a few noteworthy runtimes. Solutions are more comparable in terms of memory consumption and communication overhead. However, papers do not always report memory requirements. Authors often only report the configuration of their test system. Knowing the system configuration gives us at least an upper bound on the memory requirement. Unfortunately, not all solutions report their communication overhead either. A complete overview of the memory and communication requirements can be found in Table 7.

[2]http://yann.lecun.com/exdb/mnist/

[3]https://www.cs.toronto.edu/ kriz/cifar.html

[4]https://www.image-net.org/

[5]https://ai.stanford.edu/ amaas/data/sentiment/

1) COMMUNICATION

We can see that batched solutions for MNIST [61], [63], [66] need to transfer about 330-410MB. Due to their packing scheme, E2DM [68] only needs to transfer 17.4MB. Other solutions using ciphertext packing should require data transfers of similar size. As the datasets grow in complexity, so does the size of the data. On CIFAR-10, the ciphertexts are much larger except for SHE [65], which only needs 160MB. Hesamifard et al. [63], and Faster Crypotnets [66] need 1,803MB and 1,160MB, respectively. To encrypt a single ImageNet image, SHE [65] requires 7.7GB. On medical data of similar size, in Faster Cryptonets [66], the client needs to transfer 789.2GB. Podschwadt and Takabi [82] do not use ciphertext packing, and the client does need to transfer encrypted vector representations. Initially, the client needs to send 14 GB. On top of that, there are several interactive phases during computation to remove the noise during the evaluation of the network, which amounts to a total of 15.2 GB.

In their later work [83], the client needs to transfer between 8.75GB and 100GB depending on the split of the input data. However, this solution does not require interactive phases during computation. Both approaches transfer a similar number of ciphertexts in the beginning. The significant difference in size can be explained by RNN blocks [83] using more secure crypto parameters. When training a model on encrypted data, the client needs to send the entire training data in encrypted form. On the Youtube Spam Collection, with less than 2000 instances, the encrypted training data in PrivFT [78] comes in at roughly 550 GB. We can see that the communication overhead for solutions with ciphertext packing is significantly smaller than for solutions without. Solutions with batching can reach pretty low amortized overhead. However, it is questionable if that would be beneficial in a real-world setting. It seems more likely that a client has a few instances for processing instead of the thousands required to best use the naive batching schemes.

2) MEMORY

The size of the ciphertexts is not only an issue in transmission but also impacts the memory requirements of the server. Like with communication, the overhead of solutions with ciphertext packing requires less memory than those without. Since cloud computing and pay-as-you-use models make machines with hundreds of gigabytes of memory relatively easily accessible, memory consumption is only a secondary issue. However, if we consider hardware acceleration, such as GPUs, the memory becomes an issue again. GPU memory is typically much smaller and can become a bottleneck. Therefore, looking at the memory requirements of these systems can give us a good idea of how much they can benefit from hardware acceleration. Many solutions [61], [63], [66], [69], [82] can be run on commodity hardware, requiring less than 48 GB of RAM. As the datasets get more complex, the models also become more complex. The Faster Cryptonets [66] CIFAR-10 model needs 1433.6 GB of memory. In comparison, Al Badawi et al. [81] only need 187 GB. To use GPUs, which have only 16 GB of memory, the authors perform subtasks and swap data in and out of GPU memory. In comparison, training takes less memory. The main difference in the communication overhead is that while all the training data needs to be transmitted, it does not need to be in memory all at once. This difference is why Al Badwai et al. [78] and Nandakumar et al. [85] can train on 250 GB or less memory.

3) RUNTIME

Two approaches emerge (Table 7) for reducing runtime. (1) Reducing the per instance runtime by using batching to run a large batch at once, and (2) using ciphertext packing to reduce the runtime of running the network on a single (or a small number) instance.

For example, CryptoNets [61] was the first HE-based solution on MNIST and takes 250 seconds. Brutzkus et al. [69], using the same model as CryptoNets [61], managed to achieve a 2.2s runtime mainly through ciphertext packing. However, the runtime per instance of CryptoNets is only about 0.06s. With more powerful hardware and optimizations of the HE code, nGraph-HE [86] can achieve a total runtime of 16.7s while using batching, achieving a per instance runtime of 0.004s. We can see in Table 7 that this trend holds for MNIST models. Packing solutions are faster, while batching solutions offer better per-instance runtime. However, the question is how applicable batched solutions are. Considering the MLaaS paradigm (Fig. 1), it is questionable that a user would have thousands of instances that need classification at once. It seems more likely that the client would only have a few instances they would like to have classified with little delay. However, there is no clear answer to which approach is better, as it depends on the usage scenario.

As expected, larger datasets and models take longer to run. The CHET compiler optimizes the fastest CIFAR-10 model [87]. It takes only 164.7s, which is faster than the Al Badawi et al. [81] model executed on GPUs. The model takes 553 seconds on a single GPU, while it takes 304 seconds on four GPUs in parallel. CHET [87] wins out here due to ciphertext packing, which Al Badawi et al. [81] does not use. However, the authors also benefit from the compiler optimizations, which we can see compared to Brutzkus et al. [69]. Their CIFAR-10 model takes 711 seconds, despite using specialized ciphertext packing strategies. Solutions that do not use ciphertext packing [63], [65], [66] take much longer with 3300-12000 seconds to evaluate the CIFAR-10 model. It is worth noting that the SHE model [65] using bootstrapping would take 42.5 million seconds (492 days). The weakness of batched approaches becomes apparent on these deeper models and more complex datasets. Deeper models require either large crypto parameters or bootstrapping. Either one increases the computation cost. Batched approaches require many ciphertext objects as input and intermediate results, while packing approaches can keep the number of ciphertexts low. While this causes memory problems, it also increases the runtime since operations need to be performed on more objects.

Studies using bootstrapping for deeper models, Lee et al. [70], and Jang et al. [76], do not use batching. Lee et al. [70] use bootstrapping to run ResNet-20 [73] on encrypted data. While the classification of one CIFAR-10 instance takes almost 3 hours, the model achieves the best accuracy with 92.43%. Jang et al. [76] run a GRU model on the MNIST data achieving 94.2%. They treat each image row as an entry in a sequence, leading to a sequence of length 28. As discussed earlier (Section IV-D), the depth of RNNs depends on the length of the input sequence, creating a model with high depth. In their evaluation, processing one sequence element takes 90s. In both studies [70], [76], the authors note that a significant portion of the runtime is used by bootstrapping. In Lee et al. [70], 31.6% of time is spent performing bootstrapping in between layers. However, there are additional bootstrapping operations during batch normalization and activation functions, which are not counted in 31.6%. The bootstrapping cost would be even higher if batching were used [70].

Training on encrypted data is even slower than inference. Using bootstrapping and SIMD batching, Nandakumar et al. [85] need 40 minutes per minibatch on MNIST. Mihara et al. [75] propose an encoding that takes only 29 hours to train for 400 epochs. However, the authors only evaluate the model on a tiny dataset. PrivFT [78] uses a more realistic YouTube spam dataset; training takes 120 hours using 8 GPUs in parallel. Training on encrypted data does not seem very practical at this point. The resource requirements are almost prohibitively large, especially when coupled with bootstrapping. Bootstrapping is practically necessary for training due to the noise growth in the encrypted data. Performing training without bootstrapping either limits the number of epochs or other options for noise removal need to be used, such as interactive phases.

C. PERFORMANCE IMPACT

All approaches in this work change the NN architecture to facilitate execution over encrypted data. As discussed earlier, these changes often include different activation functions than those used on plain data. However, these changes often come at a price in NN performance. For example, polynomial activation functions are not ideal from an ML perspective and are a concession to the limitations of HE schemes. In Table 7, we summarize the performance of the models evaluated in the studies. The *Enc.* column shows the performance of models that can be run on encrypted data; all the operations have been changed to HE-friendly alternatives, and all changes to reduce complexity have been made. The performance of these models is usually similar between actual encrypted execution or execution on plaintexts.

Most studies use accuracy as the metric for their evaluation. Accuracy a is defined as:

$$a = \frac{y_c}{y_c + y_f} \tag{5}$$

With y_c as the number of correct predictions and y_f as the number of false predictions. Zhang et al. [64] perform encrypted speech recognition. Here the model transcribes a spoken piece of text. Accuracy is not the best metric for this task. In their work, the authors use word error rate wer. Given the number of deletions d (words that were missed in transcription), the number of insertions i (words that were added during transcription that were not in the original text), the number of substations s (words that were transcribed differently than they appear in the original text), and the number of correct words n, wer is defined as:

$$\text{wer} = \frac{d + i + s}{s + d + c} \tag{6}$$

SHE [65] also works with textual data. However, the authors perform next-word prediction and user perplexity per word (PPW) as their metric. Jurafsky and Martin [102] define it as follows: "The perplexity [...] of a language model on a test set is the inverse probability of the test set, normalized by the number of words." Given an input sequence of length n consisting of the word $w_1, w_2, \cdots, w_n$ we can compute the PPW as:

$$\text{PPW} = \frac{1}{n} \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_1, \ldots, w_{i-1})}} \tag{7}$$

where $P(w_i|w_1, \ldots, w_{i-1})$ is the conditional probability of w_i given $w_1, \ldots, w_{i-1}$. The choice of metric is not dependent on privacy preservation, only on the task and data.

The *Plain* column presents the model's performance without any changes. This performance is not supposed to be state-of-the-art, but it shows what the model architecture could achieve with standard ML best practices. According to the Table, this information is often missing. The *Loss* column states the relative drop in performance between the standard ML model and the model capable of encrypted execution. A negative value indicates that the HE-friendly model performs better than the standard model.

Relative loss is an important metric to judge the quality of a PPML solution. Only reporting the result of the HE-friendly or encrypted model on the task is insufficient, as it does not capture any performance sacrifices made to enable encrypted execution. On the other hand, comparing the performance of the HE-friendly model only to state-of-the-art (if the state-of-the-art performance is achieved using a different architecture) is interesting. Still, it does not capture the entire picture. State-of-the-art models are often vastly more complex than models that can be run on encrypted data.

The ideal is a three-way comparison involving state-of-the-art performance on the task, plaintext model, and HE-friendly model. Reporting performance like this allows for a better comparison of approaches independent of the task and dataset.

IX. PROPOSED EVALUATION CRITERIA

Tables 2 and 7 compare the approaches from different angles. We can see that not all studies report the same information. Even if they report the same information, it is often not directly comparable. For example, is an approach that achieves 99% accuracy on MNIST better than one that reaches 80% on CIFAR-10? Or is the solution running in 90 seconds on a big server faster than the solution taking 120 seconds on a laptop? Comparing the speedup and performance of different solutions is often not straightforward. As previously mentioned, comparing runtimes is an issue due to hardware differences. Both memory and bandwidth requirements are hardware-independent performance indicators and only depend on the algorithm and the crypto parameters. We find, however, that only some solutions report their memory or bandwidth requirements. From the observations made during this work, we propose metrics and evaluation guidelines that focus on comparability and help make robust claims about the proposed solution.

A. RESOURCE REQUIREMENTS

Efficiency is important when executing ML models, especially on encrypted data. However, wall-clock time offers poor comparability since it is very hardware dependent. When evaluating the runtime of a PPML approach, two questions are of particular interest: (1) How does the solution perform compared to the plaintext model? and (2) how does it perform compared to other solutions?

We can achieve the first objective by running the model, first on the plaintext and next on encrypted data, and comparing the wall-clock time. However, it is crucial to choose the correct execution environment. If the encrypted solution runs without hardware acceleration, running the plaintext version on the CPU provides a fair comparison. If the PPML solution uses hardware acceleration, so should the plaintext solution for comparison. Either way, the same hardware should be used for plain and encrypted execution.

Comparison with other PPML solutions is more complicated. Comparing the wall clock numbers provided in other studies can be a decent indicator, but it provides an incomplete picture. Often the hardware that experiments are performed on is too different for a direct comparison. One possibility is to run the other PPML solutions oneself. However, this runs into the problem that the source code is often not released. Only half of the papers in this study made their source code available. Not having access to the source code would only leave the implementation from scratch, which can be time-consuming. We propose hardware-independent guidelines. Ideally, authors should report their time requirements in terms of HE operations. They should break down the basic operations of the ML algorithm and report the number of HE additions, multiplications, and rotations required. For example, in Cryptonets [61], a fully connected layer (without bias and activation function) with m inputs and n neurons requires $n * m$ ciphertext-plaintext multiplications and $n * m - 1$ ciphertext-ciphertext additions. In Jiang et al. [68] the same layer requires $3n + 2m$ ciphertext-plaintext multiplications, $3n+2m+\log(n/m)$ ciphertext-ciphertext additions and $3m + 5\sqrt{n} + \log(n/m)$ rotations because of their encoding scheme. By comparing the two, we can see that, for larger n and m, Cryptonets require many more operations. This way of reporting complexity gives us a simple way of comparing the efficiency of the two solutions.

Other resource requirements also offer us insight into the complexity of different solutions and can be used as a comparison metric. Since HE ciphertext and keys can be large, the maximum memory required during execution is an informative statistic. However, only a few studies report the exact memory requirements [70], [76], [83]. Often only the memory capacity of the test system is reported. While this gives us an upper bound, it only allows a rough comparison. Especially on systems with a lot of memory, the difference between the maximum and required memory can be significant. Therefore we recommend reporting two metrics for memory requirements: (1) the maximum amount of memory required to run the solution, and (2) the size of a single ciphertext and the size of the required keys, public key, evaluation keys, rotation keys, etc. This information allows us to understand the memory requirements better; it is also beneficial when reporting the bandwidth requirements. While HE-based PPML solutions are mostly non-interactive [61], [63], [65], they still often require large amounts of bandwidth due to the size of the ciphertext and keys. We encourage reporting the communication requirements, which many solutions do. Ideally, the communication would be broken into multiple parts, the number and size of ciphertexts and the different keys and their respective size. The increase over plaintext should also be reported for both memory and bandwidth. Using these metrics provides comparability with the plaintext versions and other encrypted solutions.

We propose reporting the following values:

- Times it takes to run the model on plain and encrypted data using the same hardware
- Complexity in terms of HE operations
- Maximum memory required
- The size of basic objects (ciphertexts, keys, etc.)
- The size of the data that needs to be transferred; broken down by basic objects

B. MODEL PERFORMANCE

In terms of prediction quality, the model's performance is of equal importance to the efficiency of the execution. To facilitate comparison, it is recommended that the authors report the state-of-the-art. PPML solutions are unlikely to reach state-of-the-art due to adaptions to facilitate HE computation. It is still important since it indicates how well the solution performs in the grand scheme. We can also use it to compare different PPML solutions by comparing the relative loss in performance over the state-of-the-art. This comparison even works, to some extent, for solutions evaluating PPML models on other tasks.

The loss in performance over state-of-the-art is important for comparison as well. However, another important metric is the loss incurred by adaptions to make computation using

HE feasible. In previous sections, we discussed several such adaptions, such as polynomial activation functions, weight quantization, average pooling, etc. These changes usually come at a performance cost. To measure the performance impact, we suggest training two models. The first model, we call A, has all the necessary changes to be run on encrypted data. The second model, dubbed B, has the same structure as A but uses the best plaintext ML practices. If, for example, model A uses a fully connected layer with 200 neurons and square activation, model B should use a fully connected layer with 200 neurons and ReLU activation. The comparison of the performance of A and B shows the cost of the privacy preservation changes. The lower the difference, the better solution. The final important value is model A performance on actual ciphertexts. This step is essential to show that, for example, the approximate computation of CKKS or a plaintext space overflow does not alter results on encrypted data.

Another critical part of performance evaluation is selecting an evaluation task and dataset. A large part of the work has been done on image classification, and most authors use the MNIST dataset as one or even their only evaluation task. However, the relative loss on the MNIST dataset is usually lower than on more complex datasets, such as CIFAR-10. This difference is not surprising since MNIST is a straightforward and clean dataset. Techniques that work on MINST do not necessarily transfer to other data sets. Despite this weakness, six of the works considered here only use MNIST. Only using MNIST for evaluation does not allow us to make strong claims. Authors should show that their proposed method can also perform well on more complex data. MNIST has a place in the evaluation due to its widespread use, making it suitable for comparison. However, we recommend additionally evaluating more complex data sets such as [103], [104], and [105].

We propose reporting the following values:

- Difference to the state-of-the-art
- Performance of the model with HE adaptions
- Performance of model without HE adaptions
- Performance on encrypted data
- Evaluate datasets other than MNIST (or similar simple data sets in other domains)

X. CHALLENGES AND DIRECTIONS

As discussed in the previous sections, there are still many hurdles to making HE-based PPML practical. This section summarizes where the challenges lie and discuss potential directions and approaches.

A handful of libraries are available that support one or more encryption schemes. However, even if libraries support the same scheme, they are not interoperable. For instance, ciphertexts or keys do not have a standardized format. The same is true for the encryption parameters. The interpretation of what a parameter means can differ between libraries. In the past few years, there has been an effort to standardize HE by the community [106]. This standardization effort [107] includes an overview of the security and secure parameter recommendations, potential applications, and design considerations. It is a good start, but at the moment, developing solutions locks one into using a particular library.

Choosing a library is usually difficult; developing a well-performing secure PPML solution without in-depth knowledge about HE is hard, if not impossible. PPML developers need to have expertise in both ML and security. PPML, using HE, has not been widely adopted by the ML community, likely due to the high barrier of entry that is HE and the lack of user-friendly tools. On the other hand, security researchers often lack ML knowledge. The ML community has developed user-friendly tools allowing fast plaintext ML development [108], [109], [110]. These tools hide much of the complexity from the user; such complexity is necessary to understand when implementing solutions based on HE.

There should be an effort to develop HE-based PPML libraries by a shared community of ML specialists and crypto experts. Some work has been done on easing the entry from both sides by using compilers [86], [87]. Compilers not only make it easier to get started, but they can also significantly improve performance by applying optimizations. Compilation can help speed up the execution of encrypted data. However, the models are designed with plaintext execution assumptions.

Fast and efficient techniques on plaintext are not always the best solution for encrypted data [61], [63]. Researchers should take a more holistic approach that considers the entire ML pipeline. Much of the work focuses on running existing model architectures on encrypted data [61], [70], [87]. This work is often done by making small changes to the model architecture to make it HE-compliant and training the model from scratch [65], [66]. Ideally, the compiler suite would be able to perform all these transformations automatically while optimizing the process end-to-end. Most ML techniques in the field are minor adaptions of strategies that work on the plaintext [61], [65], [78]. But developing solutions specifically for encrypted execution might require new types of networks, activation functions, or optimizers.

Activation functions are a bottleneck in NNs using HE. These non-linear functions are necessary for the network to perform well. However, on encrypted data, they significantly impact the noise budget [66], [70]. Multiple alternatives have been proposed, but most of them make training the model harder, often due to their unbounded and non-monotonic derivatives [62], [83]. Other solutions that do not impact training, such as replacement after training or lookup tables, suffer from a loss in model performance [70]. Polynomials are an adequate replacement for traditional activation functions [61], [63], [70]. The question if there are activation functions that perform well, both on encrypted and plain data, is still open and worth investigating.

The need for HE-specific algorithms and adaptions is especially apparent in training. Work investigating training on encrypted data is minimal [75], [78], [85]. One of the big reasons is the noise build-up and the resulting need for bootstrapping or interactive noise removal. Solutions that use neither are limited to using training hyperparameters, such as batch size and the number of epochs, that are sub-optimal [78]. Approaches that do use bootstrapping are very slow and computationally expensive [76], [85], [111]. On the one hand, faster bootstrapping would remove some of the computational overhead of training on encrypted data, and we could use standard training parameters. On the other hand, having a training algorithm that works with the parameters required by HE would eliminate the need for bootstrapping or interactive noise removal and make training much more feasible.

Performance is not only an issue during training but also during testing. Researchers can evaluate models on the CIFAR-10 dataset within a reasonable time [81], [87]. However, larger models and larger inputs still take hours to compute [65], [70]. Hence, scaling up to larger, real-world datasets is costly. Improvements in latency and memory consumption have been due to ciphertext packing [25], [68], [69]. Without it scaling up to real-world data is impractical. Packing can reduce the memory requirement by an order of magnitude. Models on the CIFAR-10 dataset that do not use packing require hundreds of GB of memory [65], [66], [87], making scaling up to datasets like ImageNet near impossible.

Another potential source of increased performance is hardware accelerators like GPUs or FPGAs (Field Programmable Gate Arrays). Al Badawi et al. [81] ran models using HE on GPUs, but memory constraints were severe. High-end GPUs do not have more than 50GB of memory. The memory limitation of GPUs calls for new encoding and packing schemes for efficient execution. Research on GPU-aided HE is not limited to ML applications. For example, Geraldo et al. [112] propose an implementation of BFV accelerated by GPUs. Their results show an up to 5 times speedup in homomorphic operations. Increasing the efficiency of NNs can also help reduce the memory and runtime requirements of PPML. Much work has been done on running plaintext ML applications on resource-constrained devices like phones [113], [114]. Techniques used in that area, such as pruning and quantization, can be implemented in PPML [115]. Chou et al. [66] investigated reducing the number of computations in the network through pruning. Helbitz and Avidan [116] tackled the computational overhead of PPML by reducing the ReLU count in the network. They show that activations can be grouped, and one activation output can be used for all in the group. The accuracy impact on this grouping is low, while it can provide about 30% speedup. They test their system using two and three-party computation. If a similar speedup can be observed on HE systems or if it is even applicable to other activations, then ReLU still needs to be investigated.

Most PPML solutions focus on client data privacy; it is guaranteed if the underlying schemes remain secure. However, if these systems are to be adopted in practice, model providers likely need more assurance that their models are secure. As discussed in Section VII-B, server-side model information can be leaked using HE. To the best of our knowledge, there is no research into mitigating information leakage to the client when using HE-based PPML.

XI. CONCLUSION

In this paper, we reviewed scientific articles and publications in the particular area of Deep Learning Architectures for Privacy-Preserving Machine Learning with Fully Homomorphic Encryption. This specification narrows down the literature to a limited number of publications that are (1) well-known, (2) practical, (3) often library-included, and (4) usually cover a variety of HE techniques and deep learning architectures. The primary motivation behind this work was for the PPML researchers to observe (1) state-of-the-art HE PPML solutions and (2) the HE PPML challenges and their potential solutions. The primary challenges in implementing HE PPML include significant computational, memory, and latency overhead, in addition to incompatible operations and limited network depth. The most efficient approach to address the computational overhead is ciphertext packing. Solutions that use ciphertext packing often achieve much lower runtime and memory requirements. However, finding the best packing strategy is a complex and time-consuming task. Compiler-based solutions can assist with packing scheme selection. The problem of incompatible operations is much more significant for studies that do not use TFHE. However, TFHE-based studies cannot use ciphertext packing, limiting ways to reduce computational overhead. The problem of limited network depth arises from the noise build-up during computation. Authors could address the noise build-up using bootstrapping. However, bootstrapping further increases the computational overhead. Therefore most authors do not use it and rely on leveled HE and try to offset the reduced network depth in other ways. To compare the effectiveness of the resulting solutions, we have proposed evaluation criteria in this study. These proposed evaluation criteria will show a complete picture of a solution's performance and improve the comparability of different approaches.

REFERENCES

[1] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends Theor. Comput. Sci.*, vol. 9, nos. 3–4, pp. 211–407, 2013.

[2] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. Symp. Found. Comput. Sci. (SFCS)*, Nov. 1982, pp. 160–164.

[3] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Symp. Theory Comput. (STOC)*, Bethesda, MD, USA: ACM Press, 2009, p. 169. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1536414.1536440

[4] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *Proc. Theory Cryptogr. Conf.* Cham, Switzerland: Springer, 2011, pp. 253–273.

[5] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, vol. 1, Aug. 2015, pp. 57–64.

[6] D. J. Wu, "Fully homomorphic encryption: Cryptography's holy grail," *XRDS, Crossroads, ACM Mag. Students*, vol. 21, no. 3, pp. 24–29, Mar. 2015.

[7] D. Tourky, M. ElKawkagy, and A. Keshk, "Homomorphic encryption the, 'holy grail' of cryptography," in *Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC)*, Oct. 2016, pp. 196–201.

[8] M. Azraoui, M. Bahram, B. Bozdemir, S. Canard, E. Ciceri, O. Ermis, R. Masalha, M. Mosconi, M. Önen, and M. Paindavoine, ''SoK: Cryptography for neural networks,'' in *IFIP International Summer School Privacy Identity Management*. Cham, Switzerland: Springer, 2019, pp. 63–81.

[9] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, ''SoK: Security and privacy in machine learning,'' in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2018, pp. 399–414.

[10] B. Pulido-Gaytan, A. Tchernykh, J. M. Cortés-Mendoza, M. Babenko, G. Radchenko, A. Avetisyan, and A. Y. Drozdov, ''Privacy-preserving neural networks with homomorphic encryption: Challenges and opportunities,'' *Peer-to-Peer Netw. Appl.*, vol. 14, no. 3, pp. 1666–1691, Mar. 2021, doi: 10.1007/s12083-021-01076-8.

[11] C. G. Mouris, F. Dimitris, and N. G. Tsoutsos, ''New insights into fully homomorphic encryption libraries via standardized benchmarks,'' Cryptol. ePrint Arch., Tech. Rep. 425, 2022. [Online]. Available: https://eprint.iacr.org/2022/425

[12] A. Viand, P. Jattke, and A. Hithnawi, ''SoK: Fully homomorphic encryption compilers,'' in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 1092–1108.

[13] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, ''Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning,'' in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 501–518.

[14] J. Konený, H. Brendan McMahan, D. Ramage, and P. Richtárik, ''Federated optimization: Distributed machine learning for on-device intelligence,'' 2016, *arXiv:1610.02527*.

[15] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, ''A survey on security and privacy of federated learning,'' *Future Gener. Comput. Syst.*, vol. 115, pp. 619–640, Feb. 2021.[Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X20329848

[16] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. Mcmahan, S. Patel, D. Ramage, A. Segal, and K. Seth, ''Practical secure aggregation for privacy-preserving machine learning,'' in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1175–1191.

[17] N. Senanayake, R. Podschwadt, D. Takabi, V. D. Calhoun, and S. M. Plis, ''Neurocrypt: Machine learning over encrypted distributed neuroimaging data,'' *Neuroinformatics*, vol. 20, pp. 1–18, May 2021.

[18] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, ''Deep learning with differential privacy,'' in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 308–318.

[19] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson, ''Scalable private learning with PATE,'' 2018, *arXiv:1802.08908*.

[20] D. Evans, V. Kolesnikov, and M. Rosulek, ''A pragmatic introduction to secure multi-party computation,'' *Found. Trends Privacy Secur.*, vol. 2, nos. 2–3, pp. 246–270, 2018.

[21] O. Goldreich, S. Micali, and A. Wigderson, ''How to play any mental game, or a completeness theorem for protocols with honest majority,'' in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. New York, NY, USA: Association for Computing Machinery (ACM), 2019, pp. 307–328.

[22] M. O. Rabin, ''How to exchange secrets with oblivious transfer,'' Aiken Comput. Lab, Harvard Univ., Tech. Rep. TR-81, 1981. [Online]. Available: https://eprint.iacr.org/2005/187

[23] W.-S. Choi, B. Reagen, G.-Y. Wei, and D. Brooks, ''Impala: Low-latency, communication-efficient private deep learning inference,'' 2022, *arXiv:2205.06437*.

[24] S. Li, K. Xue, B. Zhu, C. Ding, X. Gao, D. Wei, and T. Wan, ''FALCON: A Fourier transform based approach for fast and secure convolutional neural network predictions,'' in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*. Seattle, WA, USA: IEEE, Jun. 2020, pp. 8702–8711. [Online]. Available: https://ieeexplore.ieee.org/document/9156980/

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

[26] D. Beaver, ''Efficient multiparty protocols using circuit randomization,'' in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 1991, pp. 420–432.

[27] D. Beaver, ''Precomputing oblivious transfer,'' in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 1995, pp. 97–109.

[28] A. Sahai and B. Waters, ''Fuzzy identity-based encryption,'' in *Proc. Annu. Int. Conf. theory Appl. Cryptograph. Techn.* New York, NY, USA: Springer, 2005, pp. 457–473.

[29] P. Panzade and D. Takabi, ''Towards faster functional encryption for privacy-preserving machine learning,'' in *Proc. 3rd IEEE Int. Conf. Trust, Privacy Secur. Intell. Syst. Appl. (TPS-ISA)*, Dec. 2021, pp. 21–30.

[30] T. Ryffel, D. Pointcheval, F. Bach, E. Dufour-Sans, and R. Gay, ''Partially encrypted deep learning using functional encryption,'' in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–10.

[31] (2019). *Learn the Architecture—TrustZone for AArch64*. [Online]. Available: https://developer.arm.com/documentation/102418/0101/?lang=en

[32] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, ''Innovative instructions and software model for isolated execution,'' *Hasp@ isca*, vol. 10, no. 1, pp. 1–5, 2013.

[33] D. Lee, D. Jung, I. T. Fang, C.-C. Tsai, and R. A. Popa, ''An off-chip attack on hardware enclaves via the memory bus,'' in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 1–8.

[34] J. Van Bulck, F. Piessens, and R. Strackx, ''SGX-Step: A practical attack framework for precise enclave execution control,'' in *Proc. 2nd Workshop Syst. Softw. Trusted Execution*, Oct. 2017, pp. 1–6.

[35] W. Diffie and M. E. Hellman, ''New directions in cryptography,'' *IEEE Trans. Inf. Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[36] R. L. Rivest, A. Shamir, and L. Adleman, ''A method for obtaining digital signatures and public-key cryptosystems,'' *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, doi: 10.1145/359340.359342.

[37] R. L. Rivest, L. Adleman, and M. L. Dertouzos, ''On data banks and privacy homomorphisms,'' *Found. Secure Comput.*, vol. 4, no. 11, pp. 169–180, 1978.

[38] F. Armknecht, C. Boyd, C. Carr, A. Jaschke, and C. A. Reuter, ''A guide to fully homomorphic encryption,'' *Cryptol. ePrint Arch.*, vol. 45, p. 35, 2015.

[39] P. Paillier, ''Public-key cryptosystems based on composite degree residuosity classes,'' in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1999, pp. 223–238.

[40] T. Elgamal, ''A public key cryptosystem and a signature scheme based on discrete logarithms,'' *IEEE Trans. Inf. Theory*, vol. IT-31, no. 4, pp. 469–472, Jul. 1985.

[41] Z. Brakerski and V. Vaikuntanathan, ''Fully homomorphic encryption from ring-LWE and security for key dependent messages,'' in *Annu. Cryptol. Conf.* Cham, Switzerland: Springer, 2011, pp. 505–524.

[42] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, ''(Leveled) fully homomorphic encryption without bootstrapping,'' *ACM Trans. Comput. Theory*, vol. 6, pp. 1–36, Jul. 2014.

[43] M. V. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, ''Fully homomorphic encryption over the integers,'' in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* New York, NY, USA: Springer, 2010, pp. 24–43.

[44] Z. Brakerski, S. Goldwasser, and Y. T. Kalai, ''Black-box circular-secure encryption beyond affine functions,'' in *Proc. Theory Cryptogr. Conf.* New York, NY, USA: Springer, 2011, pp. 201–218.

[45] Z. Brakerski, ''Fundamentals of fully homomorphic encryption,'' in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. New York, NY, USA: Association for Computing Machinery (ACM), 2019, pp. 543–563.

[46] K. Han and D. Ki, ''Better bootstrapping for approximate homomorphic encryption,'' in *Proc. Cryptographers' Track RSA Conf.* Cham, Switzerland: Springer, 2020, pp. 364–390.

[47] M. J. Flynn, ''Some computer organizations and their effectiveness,'' *IEEE Trans. Comput.*, vol. C-21, no. 9, pp. 948–960, Sep. 1972.

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

[49] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, ''Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE,'' in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland: Springer, 2017, pp. 377–408.

[50] J. W. Bos, K. Lauter, J. Loftus, and M. Naehrig, ''Improved security for a ring-based fully homomorphic encryption scheme,'' in *Proc. IMA Int. Conf. Cryptogr. Coding*. Cham, Switzerland: Springer, 2013, pp. 45–64.

[51] M. Albrecht, S. Bai, and L. Ducas, ''A subfield lattice attack on overstretched NTRU assumptions,'' in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 2016, pp. 153–178.

[52] V. Lyubashevsky, C. Peikert, and O. Regev, ''On ideal lattices and learning with errors over rings,'' *J. ACM*, vol. 60, no. 6, p. 43, 2013, doi: 10.1145/2535925.

[53] J. Fan and F. Vercauteren, ''Somewhat practical fully homomorphic encryption,'' *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, Mar. 2012.

[54] 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. Secur.* Cham, Switzerland: Springer, 2017, pp. 409–437.

[55] (2015). *PALISADE Homomorphic Encryption Software Library—An Open-Source Lattice Crypto Software Library*. [Online]. Available: https://palisade-crypto.org/
[56] S. Halevi and V. Shoup, "Algorithms in Helib," in *Proc. Annu. Cryptol. Conf.* Cham, Switzerland: Springer, 2014, pp. 554–571.
[57] (Nov. 2020). *Microsoft SEAL (Release 3.6)*. [Online]. Available: https://github.com/Microsoft/SEAL
[58] (Apr. 2021). *Snucrypto/HEAAN*. [Online]. Available: https://github.com/snucrypto/HEAAN
[59] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, 2018.
[60] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Proc. Annu. Int. Cryptol. Conf.* New York, NY, USA: Springer, 2018, pp. 483–512.
[61] N. Dowlin, R. Gilad-Bachrach, 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. Machine Learn.*, 2016, pp. 201–210.
[62] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 35, Mar. 2017.
[63] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," 2017, *arXiv:1711.05189*.
[64] S.-X. Zhang, Y. Gong, and D. Yu, "Encrypted speech recognition using deep polynomial networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Dec. 2019, pp. 5691–5695.
[65] Q. Lou and L. Jiang, "SHE: A fast and accurate deep neural network for encrypted data," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 10035–10043.
[66] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster CryptoNets: Leveraging sparsity for real-world encrypted inference," 2018, *arXiv:1811.09953*.
[67] S. Akleylek, N. Bindel, J. Buchmann, J. Krämer, and G. A. Marson, "An efficient lattice-based signature scheme with provably secure instantiation," in *Proc. Int. Conf. Cryptol. Africa*. Cham, Switzerland: Springer, 2016, pp. 44–60.
[68] X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Toronto, ON, Canada, Oct. 2018, pp. 1209–1222, doi: 10.1145/3243734.3243837.
[69] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 812–821.
[70] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30039–30054, 2022.
[71] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1651–1669.
[72] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux, "Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2021, pp. 587–617.
[73] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[74] J.-W. Lee, E. Lee, Y. Lee, Y.-S. Kim, and J.-S. No, "High-precision bootstrapping of RNS-CKKS homomorphic encryption using optimal minimax polynomial approximation and inverse sine function," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2021, pp. 618–647.
[75] K. Mihara, R. Yamaguchi, M. Mitsuishi, and Y. Maruyama, "Neural network training with homomorphic encryption," 2020, *arXiv:2012.13552*.
[76] J. Jang, Y. Lee, A. Kim, B. Na, D. Yhee, B. Lee, J. H. Cheon, and S. Yoon, "Privacy-preserving deep sequential model with matrix homomorphic encryption," in *Proc. ACM Asia Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, May 2022, pp. 377–391, doi: 10.1145/3488932.3523253.
[77] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014, *arXiv:1409.1259*.
[78] A. A. Badawi, L. Hoang, C. F. Mun, K. Laine, and K. M. M. Aung, "PrivFT: Private and fast text classification with homomorphic encryption," *IEEE Access*, vol. 8, pp. 226544–226556, 2020.
[79] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," 2016, *arXiv:1607.01759*.
[80] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Proc. Int. Conf. Sel. Areas Cryptogr.* Cham, Switzerland: Springer, 2018, pp. 347–368.
[81] A. Al Badawi, C. Jin, J. Lin, C. F. Mun, S. J. Jie, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar, "Towards the AlexNet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with GPUs," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1330–1343, Jul. 2021.
[82] R. Podschwadt and D. Takabi, "Classification of encrypted word embeddings using recurrent neural networks," in *Proc. PrivateNLP@ WSDM*, 2020, pp. 27–31.
[83] R. Podschwadt, "Non-interactive privacy preserving recurrent neural network prediction with homomorphic encryption," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, Apr. 2021, pp. 65–70.
[84] M. Bakshi and M. Last, "Cryptornn-privacy-preserving recurrent neural networks using homomorphic encryption," in *Proc. Int. Symp. Cyber Secur. Cryptogr. Mach. Learn.* Cham, Switzerland: Springer, 2020, pp. 245–253.
[85] K. Nandakumar, N. Ratha, S. Pankanti, and S. Halevi, "Towards deep neural network training on encrypted data," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2019, pp. 40–48.
[86] 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. Frontiers*, 2019, pp. 3–13.
[87] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "CHET: An optimizing compiler for fully-homomorphic neural-network inferencing," in *Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design Implement.* New York, NY, USA: Association for Computing Machinery, Jun. 2019, pp. 142–156, doi: 10.1145/3314221.3314628.
[88] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," 2017, *arXiv:1702.03044*.
[89] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[90] M. H. Stone, "The generalized Weierstrass approximation theorem," *Math. Mag.*, vol. 21, no. 5, pp. 237–254, 1948.
[91] T. J. Rivlin, *Chebyshev Polynomials*. Mineola, NY, USA: Courier Dover, 2020.
[92] E. Lee, J.-W. Lee, Y.-S. Kim, and J.-S. No, "Minimax approximation of sign function by composite polynomial for homomorphic comparison," *IEEE Trans. Dependable Secure Comput.*, early access, Aug. 18, 2021, doi: 10.1109/TDSC.2021.3105111.
[93] R. E. Goldschmidt, "Applications of division by convergence," Ph.D. thesis, Dept. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 1964.
[94] G. Meinardus, *Approximation Functions: Theory and Numerical Methods*, vol. 13. Cham, Switzerland: Springer, 2012.
[95] J. L. H. Crawford, C. Gentry, S. Halevi, D. Platt, and V. Shoup, "Doing real work with FHE: The case of logistic regression," in *Proc. 6th Workshop Encrypted Comput. Appl. Homomorphic Cryptogr. (WAHC)*, Feb. 2018, pp. 1–12.
[96] P. Scheunders, "A genetic lloyd-max image quantization algorithm," *Pattern Recognit. Lett.*, vol. 17, no. 5, pp. 547–556, May 1996.
[97] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Cham, Switzerland: Springer, 2021, pp. 648–677.
[98] V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, and R. Kim, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," *J. Amer. Med. Assoc.*, vol. 316, no. 22, pp. 2402–2410, 2016, doi: 10.1001/jama.2016.17216.
[99] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 601–618.
[100] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, "PRADA: Protecting against DNN model stealing attacks," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Jun. 2019, pp. 512–527.

[101] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.
[102] D. Jurafsky and J. H. Martin, "N-gram language models," in *Speech and Language Processing*, 3rd ed. Hoboken, NJ, USA: Prentice-Hall, Jan. 2022, pp. 37–38. [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/
[103] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *Proc. 25th Int. Conf. World Wide Web*, Apr. 2016, pp. 507–517.
[104] A. Krizhevsky and G. Hinton, *Learning Multiple Layers of Features From Tiny Images*. Toronto, ON, Canada: MIT Press, 2009.
[105] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[106] (2016). *Homomorphic Encryption Standardization—An Open Industry / Government / Academic Consortium to Advance Secure Computation*. [Online]. Available: https://homomorphicencryption.org/
[107] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption security standard," HomomorphicEncryption.org, Toronto, ON, Canada, Nov. 2018. [Online]. Available: https://homomorphicencryption.org/standard/
[108] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Mar. 2011.
[109] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implement. (OSDI)*, May 2016, pp. 265–283.
[110] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. Antiga, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–18.
[111] E. Lee, J.-W. Lee, J. Lee, Y.-S. Kim, Y. Kim, J.-S. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 12403–12422.
[112] P. G. M. Alves, J. N. Ortiz, and D. F. Aranha, "Faster homomorphic encryption over gpgpus via hierarchical dgt," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2021, pp. 520–540.
[113] K. Nan, S. Liu, J. Du, and H. Liu, "Deep model compression for mobile platforms: A survey," *Tsinghua Sci. Technol.*, vol. 24, no. 6, pp. 677–693, Dec. 2019.
[114] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: Automl for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 784–800.
[115] E. Aharoni, M. Baruch, P. Bose, A. Buyuktosunoglu, N. Drucker, S. Pal, T. Pelleg, K. Sarpatwar, H. Shaul, O. Soceanu, and R. Vaculin, "HE-PEX: Efficient machine learning under homomorphic encryption using pruning, permutation and expansion," 2022, *arXiv:2207.03384*.
[116] I. Helbitz and S. Avidan, "Reducing ReLU count for privacy-preserving CNN speedup," 2021, *arXiv:2101.11835*.

ROBERT PODSCHWADT (Student Member, IEEE) received the B.S. and M.S. degrees in computer science from Hochschule der Medien Stuttgart, Germany, in 2012. He is currently pursuing the Ph.D. degree with the University of North Texas.

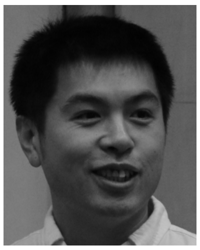
From 2012 to 2017, he worked as a System Developer at Rhode & Schwarz Cybersecurity. He is also at Georgia State University with Prof. Daniel Takabi at the INSPIRE Center. His research interests include PPML using HE and attacks on neural networks using adversarial examples.

DANIEL TAKABI (Member, IEEE) received the B.S. degree in computer engineering from the Amirkabir University of Technology, Tehran, Iran, in 2004, the M.S. degree in information technology from the Sharif University of Technology, Tehran, in 2007, and the Ph.D. degree in information science and technology from the University of Pittsburgh, Pittsburgh, PA, USA, in 2013.

He is currently an Associate Professor in computer science and the Next Generation Scholar with Georgia State University, Atlanta, GA, USA. He is also a Founding Director of the Information Security and Privacy: Interdisciplinary Research and Education Center, which is designated as the National Center of Academic Excellence in Cyber Defense Research. His research interests include various aspects of cybersecurity and privacy, including privacy-preserving machine learning, adversarial machine learning, advanced access control models, insider threats, and usable security and privacy. He is a member of ACM.

PEIZHAO HU (Member, IEEE) is currently an Associate Professor with the Department of Computer Science, Rochester Institute of Technology. He has served as a Technical Program Committee and an Organizing Committee for a number of conferences and workshops, including PerCom and LCN. In addition, he has served as a Reviewer for international journals, including the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY and the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING. His research interests include privacy-preserving cloud data analytics, specifically homomorphic encryption and multiparty computations, and distributed systems, including mobile and pervasive computing and blockchain.

MOHAMMAD H. RAFIEI received the Ph.D. degree in civil engineering focused on machine learning (ML) from Ohio State University (OSU), in December 2016, under Dr. Hojjat Adeli. He continued his ML research as a Postdoctoral Researcher in infrastructure engineering and neuroscience at OSU, between February 2017 and August 2018. Then, he joined Johns Hopkins University (JHU) as a Postdoctoral Fellow, in August 2018, and an Adjunct Assistant Research Scientist, in September 2019, working in ML nano-scale materials simulations. He worked as a ML Course Developer and an Instructor at the Engineering for Professionals Graduate-Level Program, JHU, in 2020. He joined the INSPIRE Center, Georgia State University, in January 2021, as a Postdoctoral Fellow working on the emerging area of privacy-preserving ML.



ZHIPENG CAI (Senior Member, IEEE) received the B.S. degree from the Beijing Institute of Technology and the M.S. and Ph.D. degrees from the Department of Computing Science, University of Alberta. He is currently an Associate Professor at the Department of Computer Science, Georgia State University, USA. Prior to joining GSU, he was a Research Faculty with the School of Electrical and Computer Engineering, Georgia Institute of Technology. His research interests include the Internet of Things, machine learning, cyber-security, privacy, networking, and big data. He was a recipient of an NSF Career Award. He served as a Steering Committee Co-Chair and a Steering Committee Member for WASA and IPCCC. He also served as a Technical Program Committee Member for more than 20 conferences, including INFOCOM, ICDE, and ICDCS. He has been serving as an Associate Editor-in-Chief for *High-Confidence Computing* (HCC) (Elsevier) journal and an Associate Editor for more than ten international journals, including IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.