SoK: Cryptographic Neural-Network Computation

Lucien K. L. Ng
Georgia Institute of Technology

Sherman S. M. Chow
The Chinese University of Hong Kong

2023 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-9336-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/SP46215.2023.10179483

***Abstract*—We studied 53 privacy-preserving neural-network papers in 2016-2022 based on cryptography (without trusted processors or differential privacy), 16 of which only use homomorphic encryption, 19 use secure computation for *inference*, and 18 use non-colluding servers (among which 12 support *training*), solving a wide variety of research problems. We dissect their cryptographic techniques and "love-hate relationships" with machine learning alongside a genealogy highlighting noteworthy developments. We also re-evaluate the state of the art under WAN. We hope this can serve as a go-to guide connecting different experts in related fields.**

I. Introduction

Secure machine-learning (ML) computation, specifically privacy-preserving neural networks (PPNN), has attracted much attention for their broad applicability in often privacy-sensitive scenarios. Model owners who provide inference services or powerful cloud platforms that provide training services might not always be trusted to ensure the data privacy of the inference query or the training data. Cryptographic techniques come in handy in realizing four privacy services (Table I) – oblivious inference, outsourced inference, outsourced training (by one data owner), and private training (for *multiple* data owners).

Oblivious Inference. Many inference applications involve sensitive data in queries. Model owners could give the neural network (NN) to queriers for local evaluation, but they often see the model as a valuable intellectual property [1]. Leaking models also paves the way for attacks [2], [3], [4]. In *oblivious inference*, a querier can learn the inference result (and what is inferrable from it) in a secure two-party computation manner over the query and the model parameters acquired from the training.

Outsourced Inference/Training. Model owners may not have in-house facilities for running inference services. With secure *outsourced inference*, untrusted third parties can be delegated to answer queries without knowing the model, *i.e.*, it is a threat that any server is curious about the model too. Likewise, the data owner may desire secure *outsourced training*. (Frameworks for outsourced training mostly support outsourced inference because handling queries is a sub-routine of NN training.)

Private Training. *Multiple* mutually distrustful data owners can collectively train an NN with privacy using interactive protocols. Typically, the job can be done by servers that none of them trust for data privacy. We use the term *private training* for *multiple* owners, which covers outsourced training (for one data owner).

Early Works focused on the less complex task of inference. Barni, Orlandi, and Piva [5], [6] process model and query by (additive) homomorphic encryption (HE) for linear NN computations and garbled circuits [7] and HE (with polynomial approximation) for non-linear ones. Sadeghi and Schneider [8] rely on secure universal-circuit evaluation [7], [9] solely. It was 2006-08 when NN did not (re-)gain its popularity, and fully-HE [10] and leveled-HE (LHE) were not available. In 2016, a purely LHE-based oblivious inference solution CryptoNets was proposed [11]. Shortly afterward, MiniONN [12] and SecureML [13] appeared. The former uses secure two-party computation techniques beyond LHE for oblivious inference, while the latter is the first 2-server solution of private training.

Dissecting the Development. PPNN has then rapidly developed across different disciplines, such as system research, scientific computing, machine learning, and cryptography (or crypto in short), posing a rich set of challenging problems requiring quite a few different skill sets to tackle. We systematize 53 *frameworks*[1] born in these 8 years, microscopically explore their different technical contributions[2], and categorize them by their problem settings and major approaches, summarized in Table IV. Figure I-D sorts out the branches in the literature for a clear genealogy and easy identification of research gaps/opportunities.

Easier Comparisons. This SoK also explains the commonly-used benchmarks and shed light on how to gauge a framework, which to use for a given setting/goal, and which frameworks to compare with, *e.g.*, to avoid unfair comparison between one assuming non-colluding servers while the other does not. For the state-of-the-art frameworks assumed to run on LAN, we re-evaluate them on WAN. We found 1000Mbps is needed for peak performance, but lower bandwidth ($10 \sim 100$Mbps) suffices for some of them in non-real-time applications.

Fostering Interdisciplinary Research. Many issues in PPNN may look complicated at first glance to "outsiders." We explain the challenges and tricks for addressing them. Together with the categorization of contributions in Table IV, experts of different fields can quickly identify the papers related to their interested (sub-)topics, dive right into the crux, and avoid reinventing the wheel. Finally, we share some outstanding challenges and less-explored open problems at the end of a (sub-)section.

Scope. We focus on purely-crypto-based approaches and exclude those that use a trusted execution environment, *e.g.*, [16]. As crypto literature, membership inference [17] and model inversion [18][3] are not in our scope; and neither is federated [23] or coopetitive learning [24][4]. Finally, we refer

Sherman Chow (corresponding, 0000-0001-7306-453X) is supported by GRF 14210319 from RGC. Lucien Ng contributed to this work mostly while at CUHK. An interactive full version is available at https://sokcryptonn.github.io.

[1]We call NN software frameworks (*e.g.*, PyTorch) by libraries instead.

[2]Prior surveys focus on HE-based works [14] or an overlapping subset of $\leq$13 frameworks [4], [15] without aiming for a clear lineage of ideas.

[3]Differential privacy [19], [20], [21], [22] can serve as defenses.

[4]Some may use crypto techniques [25], [26] but mostly for aggregation [27].

DOI 10.1109/SP46215.2023.00198

to a recent SoK [28] for longer expositions of the basics, with its scope confined to open-sourced projects and MPC libraries.

II. Preliminary

A. Deep Neural Network (DNN)

DNN (or just NN) imitates how biological NNs learn by inferring an underlying model from samples, which can be used for prediction/classification. DNNs simplify the imitation to a stack of linear and non-linear layers $f_1, \ldots, f_n$; each (except the first input layer) takes the outputs from its previous layer[5]. One may see it as a chained function $\mathsf{NN} = f_n \circ \cdots \circ f_1$ if inputs other than weights are seen as "hardwired" parameters. The description in this section supposes the NNs are for visual tasks.

Linear Layers. Common examples include dense/fully-connected layers (matrix multiplication) and 2D convolution. The weight, *i.e.*, matrices of the model and kernels of the convolution, are learnable parameters that should be kept private. A linear layer outputs $y_i = \sum_j w_{ij} \cdot x_{ij}$ for weight w_{ij} and input x_{ij}. Its evaluation can be reduced to dot products.

Common Kinds of Non-Linear Layers:
i) Activation layers apply an activation function to all inputs, *e.g.*, popular softmax, rectified linear unit ReLU, sigmoid, *etc.*
ii) Pooling layers aggregate a window from the 2D-output from their previous layers, *e.g.*, maxpool, meanpool, *etc.*
iii) Normalization layers regularize the inputs into a smaller range, *e.g.*, batch normalization (BatchNorm) [29] maps each pixel to a standard normal distribution across the input batch.
iv) The output layer computes the inference output based on all its inputs, *e.g.*, softmax or argmax. For training, it further computes a *loss value*, judging how good/bad the NN is.

The architecture of an NN specifies the type and dimension of each layer. The dimension of a layer means the size and shape of the inputs and output. Layers may include other specifications, *e.g.*, convolutional and pooling layers take specifications on the shape of the kernels or the pooling windows, respectively.

(Supervised) Training takes a labeled training data set and an architecture and outputs the weights (or learnable parameters) of an NN. Most NN training adopts *stochastic gradient descent* (SGD), which updates all weights $\mathbf{w}$ iteratively over the training data batch-by-batch. During training, besides computing the outputs, the output layer computes a loss value $L = \mathcal{L}_{\mathbf{w}}(y', y)$ respective to $\mathbf{w}$, where y' and y are the labels and the inference outputs of that batch, respectively. Let w_i be a weight tensor of the i-th layer f_i, and Δw_i be its corresponding derivative, a.k.a. the *gradient*. The weight is updated via $w_i' = w_i - \alpha \Delta w_i$, where α is a learning rate to regulate the weight change. With a reasonable α, the loss will gradually decrease, meaning the model can make the right inference according to the training data.

Backward propagation derives Δw_i. The derivative of a layer can be computed with that of the next layer recursively via $\Delta w_i = f'_{i,w}(\Delta x_{i+1}, x_i)$ and $\Delta x_i = f'_{i,x}(\Delta x_{i+1}, w_i)$, where $f'_{i,v}$ are functions for computing the derivative for v in the i-th layer. Their computation is similar to the layer's original operations. So, it can happen recursively and in backward order from $\Delta w_n = \partial_y L$ in the loss layer to the input layer.

[5]More advanced NNs may have multiple paths of layers from inputs to outputs; each may take inputs from more previous layers.

Machine-learning Libraries (*e.g.*, Caffe [30], Keras, PyTorch, TensorFlow [31], and Theano) mostly take care of tedious computations of NN inference and training for developers to conveniently realize an NN by just specifying its architecture. They translate NN specifications into *computation graphs*, which are then relayed to compilers to generate the machine code. Some PPNN frameworks aim to provide similar convenience.

Common Benchmarks. Oblivious inference aims to reduce the *latency*, *i.e.*, the time between the initiating query and the return of the result. Another factor, *throughput*, counts how many queries or training data one can process for a given time. They are jointly affected by the choices of secure protocols, NN architectures, network conditions, and hardware components. Private training also aims to improve the accuracy of the trained model. It depends on the convergence rate, indicating the needed steps to attain saturated accuracy.

GPUs feature an *instruction throughput* orders of magnitude higher than CPU when fully utilized: SIMD (*single instruction multiple data*) performs the same operation across vectorized data simultaneously, while *parallelism* divides a task for many GPU cores to process. GPUs are mainly optimized for floating-point numbers. They are popular choices for NN computations. Crypto primitives, however, cannot benefit from GPUs by default as they mostly work on fixed-point numbers. Leveraging GPUs for privacy-preserving NN poses challenges [32], [16].

B. Problem Formulation and Computation and Threat Models

We consider an abstract threat model in typical crypto literature[6]. Oblivious inference can be cleanly abstracted as the secure two-party computation of $f(model, query) \rightarrow (\perp, result)$. The model owner and the querier are also called the *server* and the *client*, respectively. This paper refers to it as *server/client* or *S/C* (to avoid confusion with 2PC frameworks using two non-colluding servers on top of two-party computation (2PC) techniques). In practice, some metadata, such as the dimension of the query as a feature vector, might be considered public knowledge (see Section VIII-D).

Outsourced inference refers to the same computation, while the $model$ is outsourced in an encrypted form to a single server or secret-shared (Section II-C2) among different non-colluding servers. The querier may or may not interact with servers during the computation. Finally, n-party private training can be abstracted as the secure multi-party computation of $f(\langle dataset \rangle_1, \ldots, \langle dataset \rangle_n) \rightarrow (\langle model \rangle_1, \ldots, \langle model \rangle_n)$. Each participant is a non-colluding server, inputs a secret share $\langle \cdot \rangle$ of the training dataset received from the data contributors, and outputs a secret share $\langle \cdot \rangle$ of the trained model. The trained parameters in NN are secret protected by secret shares.

Non-colluding assumption means the participants, *e.g.*, *helper servers* [33], will not reveal their secrets to any other parties, *e.g.*, competing cloud servers, prestigious agents, a subset of model owners (in inference), and/or a subset of data contributors (in training). The number of servers required by existing frameworks ranges from 2 to 4. An adversary just needs to compromise any two servers for a total break against most of these frameworks covered in this SoK. A 4-party setup is easier to be

[6]*e.g.*, the honest party's hardware and software are not compromised.

compromised than a 3-party setup as there is one more server as an attack target. The same applies to 4-/3- vs. 2-party setups.

Many frameworks adopt *offline/online computation* to reduce the *online latency/throughput*. Consider oblivious inference; in the offline phase, one can prepare, probably in batch, pre-computed results without knowing the query, using techniques that might be heavier than those in online computation. In the online phase, where the query is known, the pre-computed results can speed up the online computation.

C. Cryptographic Primitives

1) Secure Multi-Party Computation (MPC) allows mutually distrustful parties to jointly compute a known function over their respective secret inputs via interactions. The computation results are often secret-shared in some forms among the parties. We refer to the MPC for n parties by nPC (*e.g.*, 4PC). We also refer to frameworks with non-colluding servers by MPC frameworks.

2) (t, n) *Secret Sharing (SS)* distributes a secret x into shares $\{\langle x\rangle_i\}$ for n parties; x can be recovered from any $t \geq n$ shares, *e.g.*, $\langle x\rangle_0 \in \mathbb{Z}_p$ and $\langle x\rangle_1 = x - \langle x\rangle_0 \bmod p$ are $(2, 2)$ *additive* SS of $x \in \mathbb{Z}_p$. When $p \gg 2$, we also call them *arithmetic SS*. Addition and constant multiplication can be done locally. Multiplication over SSs $\langle x \cdot y\rangle$ from $\langle x\rangle, \langle y\rangle$ can be done using Beaver's trick [34], which requires both parties to have prepared offline (via HE, oblivious transfer, or a non-colluding server) additive SS of u, v, z (*Beaver triplets*), where $z = u \cdot v$.

Boolean SS $(2, 2)$-shares a bit-string via bit-wise exclusive OR (XOR) over $\mathbb{Z}_2$. GMW protocol of Goldreich *et al.* [9] can evaluate logic gates (*e.g.*, AND, XOR), simple circuits (*e.g.*, CMP for comparison), and a composed circuit over them. To save communication rounds, we may run GMW in parallel for some gates, but its runtime is still linear in the circuit depth.

3) Yao's Garbled Circuit (GC) [7] also supports oblivious 2PC circuit evaluation over "encrypted" truth tables of gates, with decryption keys named *Yao's shares*. GC only takes constant communication rounds. Moreover, the encrypted truth table can be sent in the offline phase. In the online phase, the garbler only obliviously sends the evaluator the shares for input. In contrast, GMW requires interaction for each AND gate. Thus, GC is thus a better choice than GMW for 2PC [35, Fig.3].

4) Oblivious Transfer (OT) allows the receiver to pick the i-th item obliviously (hiding i) from the sender's table of items $A_1, \ldots, A_n$, *i.e.*, it realizes 2PC $f(\{A_j\}, i) \to (\perp, A_i)$. In OT extension [36], online crypto operations are solely symmetric-key ones, while public-key operations are shifted offline. OT is a building block for GC and Beaver triplets [34]. It also realizes oblivious *table lookup* with $O(n)$ communication cost [37].

With $\geq$3 non-colluding servers, GMW becomes a more popular choice than GC due to the cheap offline phase and the low storage requirement. The third server can cheaply distribute the beaver triplets instead of running the expensive OTs. After the offline phase, GC requires E to store $\approx$256 bits for each AND gate. Yet, GMW only requires the parties to store 3 bits in total. It thus reduces the parties' storage pressure stemming from offline preparation.

5) Homomorphic Encryption (HE) supports operations over encrypted messages. Encryption $[x] = \mathsf{Enc}_{\mathsf{pk}}(x)$ of x can be decrypted by secret key sk when $(\mathsf{pk}, \mathsf{sk})$ is a matching key pair. *Additive-HE (AHE)*, a.k.a. linear-HE, supports additive homomorphism like arithmetic SS. *Leveled-HE (LHE)*, a.k.a. somewhat-HE, supports a limited number of multiplications corresponding to the depth of an *arithmetic circuit*. The number is a tunable parameter that affects the ciphertext size and efficiency of the homomorphic operations (Section VIII-B).

LHE like BFV-HE [38], [39], BGV-HE [40], and CKKS-HE [41] are SIMD-friendly, *i.e.*, a ciphertext has multiple slots, each fits a message in ring $\mathbb{Z}_p$, and a homomorphic operation ($\star$, $*$, and $\circ$) applies on all slots simultaneously. Namely, addition: $\mathsf{Dec}([(x_i)_i] \star [(y_i)_i]) = (x_i + y_i)_i$, plaintext-ciphertext multiplication: $\mathsf{Dec}((x_i)_i \circ [(y_i)_i]) = (x_i \cdot y_i)_i$, and ciphertext-ciphertext multiplication: $\mathsf{Dec}([(x_i)_i] * [(y_i)_i]) = (x_i \cdot y_i)_i$. (When the context is obvious, we may replace $\circ$ and $*$ by $\cdot$ and $\star$ by $+$.)

For security, they introduce *noises* to ciphertexts. Fresh encryption starts with very little noise. Noises accumulate upon evaluating $\star$, $*$, and $\circ$. If they exceed the *noise quota*, decryption will fail. The quota implicitly defines the number of $\star$, $*$, and $\circ$ allowed to take. Yet, one can ask the secret key holder to help "denoise" the ciphertext by adding an additive mask (via $\star$) without leaking the encrypted message (see Section VI-E).

Two SIMD Layouts. Consider the central part of a photo is of interest; one ciphertext thus stores the same part of many photos. Such kind of batch-query processing via SIMD is relevant only when the client always has a batch of queries. An alternative usage is to handle one single (feature-vector) query at a time, but with its features spread across SIMD slots of a ciphertext.

One may want to homomorphically evaluate over multiple slots of a query (*e.g.*, for a dot product). A homomorphic *rotation* over the encrypted query is required. Rotation is more computationally expensive than plaintext-ciphertext multiplication and addition. It also increases the noise additively [40], [42].

TFHE [43] is torus-based fully-HE (FHE) with much faster bootstrap procedures but slower addition/multiplication than BFV-HE and BGV-HE. TFHE is mainly used to evaluate *binary gates* [44], [45], [46] with very little communication cost (proportional to the input size) when compared to GC/GMW (the whole circuit size), but it takes a longer computation time.

BFV-HE, BGV-HE, and TFHE are all based on ring-LWE. A ciphertext under one scheme can be switched to that of another with the message inside unchanged [47].

Microsoft SEAL implements BFV-HE and CKKS-HE. HElib implements BGV-HE. TFHE library is on GitHub.

6) General Comparison of the Gadgets. Arithmetic SS operations can be as cheap as a few CPU instructions, but each multiplication takes a communication round. HE could be a better choice in general in a high-latency network as no communication is needed in between. This motivates pure-HE frameworks (Sections IV and V-A) and differentiates them from the rest.

LHE and arithmetic SS are born to handle linear layers. Boolean/circuit-based primitives, including TFHE, boolean SS, and Yao's shares, support arbitrary functions, including non-linear ones. This motivates the mixed framework (Section VI).

Arithmetic SS, boolean SS, and Yao's shares (A/B/Y) can be converted to each other with a few communication rounds (except Y to B, which is free) such that one can use *mixed-protocol computation* with the most-fit form for each kind of computa-

tion. ABY [35] is proposed for this purpose. This leads to interesting interbreeds in mixed and MPC frameworks (Section VII).

7) Struggles in Fixed-Points and Limited Bitwidth. Most crypto primitives operate over $\mathbb{Z}_p$ for a large prime p, which we call *fixed-points*, while NN runs over floating-point numbers, which we call *floats*. A float x_f can be quantized as fixed-point $x_Q = \lfloor x_f \cdot P_x \rceil$ for the scaling factor P_x. We can then avoid *overflow* in, say, dot product by computing over fixed points. The resulting fixed y_Q can then be dequantized by $y_f = y_Q / P_y$.

Bitwidth is the number of bits needed to represent the operands, which is logarithmic in the largest magnitude. To fully emulate single-precision floats, one may need 256-bit fixed points. More bitwidth means more transferred bytes, CPU/GPU instructions, larger circuits for GMW/GC, and collectively much higher communication and computation costs. As a rather fundamental issue, different classes of frameworks tackle a slightly different form of it in their own ways.

III. Systematizing Machine learning × Cryptology

Crypto operations usually incur high overhead and tend to attain a lower accuracy for operating over fixed-points (*e.g.*, see Section VI-C). We highlight the initial struggles and some milestones. A query on MNIST takes CryptoNets [11] 300s. Two years later, Gazelle [48] takes only 0.03s. Building upon the success of mixed frameworks, GForce [49] can process a query on CIFAR-10, a harder dataset, within 0.3s online latency using the popular VGG-16 NN. For the same dataset, Gazelle takes 3.56s with a much smaller, custom NN.

For private training, SecureML [13] over 2 non-colluding servers achieves an online training throughput of $\sim 3.3 \times 10^5$ images/hour on MNIST at 93% accuracy. Trident [50] over 4 servers attained an online throughput $\sim 1.4 \times 10^7$ on the same NN. To support training on more realistic datasets and popular NNs, CryptGPU [51] integrates more advanced ML techniques, *e.g.*, BatchNorm and softmax, and achieves 83.7% accuracy on CIFAR-10 with VGG-16 at a $\sim 4 \times 10^4$ online throughput.

PPNN researchers look into the nature of the operations at various levels and their interactions, and tailor-make protocols optimized under constraints due to many reasons, *e.g.*, scientific computation for the traditional setting (of CPU/GPU instead of crypto layers), the ML expectation, and crypto tools. At the layer level, different protocols handle each layer type separately. Such modular designs allow flexibility over many architectures.

Roadmap. Sections IV-VII discuss pure-LHE-based, boolean-circuit-based, mixed, and MPC-based frameworks, respectively. Section VIII discusses special focuses, which might interest experts outside crypto communities. Section IX compares the performance of different frameworks and puts them on the Pareto frontier. Appendix I recommends the state of the arts for specific functions under different settings/assumptions.

IV. Pure-LHE Frameworks

Pure-LHE frameworks (*e.g.*, CryptoNets [11]) use LHE as the only crypto tool throughout the entire evaluation of oblivious inference. The client first encrypts the query x and sends it to the server. For linear layers with weights $\{w_{ij}\}$, the server computes $[y_i] = \sum_j w_{ij} \cdot [x_{ij}]$ for encrypted layer input $[x_{ij}]$. Some mixed frameworks (Section VI) also handle linear layers in this way. (We thus defer to Section VI-E for LoLa [52].) Below, we discuss "BNormCrypt" [53], CryptoDL [54], FasterCryptoNets [55], HCNN [56], E2DM [57], and Glyph [58]. (HE compilers/optimizers are deferred to Section VIII-B.)

TABLE I
Privacy Services Supported by Different Paradigms

Framework Type	Oblivious Inference	Outsourced Inference	Outsourced/Private Training
Pure-LHE (§IV)	●	◐	○
TFHE-based (§V-A)	●	◐	◐
Pure-GC (§V-B)	●	◐	○
Mixed (§VI)	●	○	○
MPC-based (§VII)	●	●	◐

●: All (frameworks of the type support the service), ◐: Some, ○: None

A. Major Developments for Pure-LHE-based Inference

Non-linear layers pose a challenge to pure-LHE frameworks. They tend to choose specific functions that can be either supported or approximated by polynomials. CryptoNets is a polynomial-network [59] approach, which uses only squaring $[x] \cdot [x]$ as the activation function. For pooling layers, it only supports meanpool by $[y_i] = \sum_j [x_j]$. The client internally maintains the output's scale after computing each layer (*e.g.*, the output is scaled up by d for the d-size window meanpool). Eventually, the server sends the output layer's $[y]$ to the client, who decrypts it (and divides it by the scale) to obtain the query result.

Batch normalization [29] is an ML trick that helps ensure accuracy by normalizing the range of the inputs since polynomial approximation is only accurate within a small range. "BNormCrypt" [53] and subsequent works insert *BatchNorm layers* before activation layers. BatchNorm in inference can be efficiently done with HE ciphertexts since the computation is basically point-wise (plaintext-ciphertext) multiplication and addition. BNormCrypt approximates ReLU by $0.2[x]^2 + 0.5[x] + 0.2$.

Speeding up Multiplication. CryptoDL [54] exploits various methods for approximation by low-degree (2∼4) polynomials, including numerical analysis, Taylor series, and Chebyshev polynomial, since ciphertext-ciphertext multiplication is not cheap. FasterCryptoNets [55] uses power-of-2 coefficients for the polynomials to enable more efficient plaintext-ciphertext multiplication. It also quantizes the weights of linear layers to power-of-2 for more efficient multiplication.

Pruning sets some weights in a trained NN to 0. Pure-LHE frameworks above dedicate each HE ciphertext to a specific entry across all queries in a batch; each entry is then multiplied by the same weight. These frameworks (*e.g.*, [55]) can benefit from pruning by skipping the multiplication of the ciphertext and the zeroed weights, which can skip up to >90% of those.

GPU for HE. HCNN [56] is like CryptoNets. They both use BFV-HE for oblivious inference, but HCNN implements A*FV [60], a variant that better utilizes GPU's capability.

B. Outsourced Inference and Training without non-collusion

Notably, HE can support outsourced inference and training without non-colluding assumptions. For example, E2DM [57] uses only cipher-cipher multiplications over private weights. It

only demonstrated reasonable performance on NNs with square activation and a few fully-connected layers. Deeper NNs with convolutions require more costly bootstrapping and rotations.

Glyph [58] supports outsourced training. It adopts transfer learning to reduce the costly homomorphic operations. It trains many layers close to the input layer of an NN with publicly available data and only uses HE for training the remaining layers with private data. Glyph does not consider (multi-party) private training as the single secret key owner, effectively *the* data owner, can decrypt the ciphertexts and acquire all training data. One could use heavyweight multi-key FHE. We thus ask:

Open Problem 1. *How to realize high-throughput and accurate private training without non-colluding assumptions?*

Section VII will explore such "non-colluding" frameworks.

To sum up, the distinctive feature of the pure-LHE framework is the use of polynomial networks. This begs the question:

Open Problem 2. *What positive/negative results can we establish on the sample complexity, expressiveness, and (the practical) training time for networks based on polynomials?*

Linvi *et al.* [59] provided some results. Understanding more about deep learning still needs more research [61]. In contrast, Sections VI-B and VII-C explore non-polynomial approaches.

V. Boolean-Circuit-based Approaches

A. TFHE-based (Inference) Frameworks

To avoid the higher cost of homomorphic arithmetic operations, an alternative is TFHE [43], optimized for binary gates. Glyph [58] switches between different kinds of ring-LWE-based HE schemes. It uses BGV-HE [40] for linear arithmetics and TFHE for de-quantization and non-linear layers.

Binarized neural network (BNN) [62] confines its weights and inputs of linear layers to $\{-1, 1\}$. It uses $\mathsf{Sign}(x) = |x|/x$ as the activation function, which is easy to implement in boolean circuits. FHE-DiNN [44] implements BNN on TFHE [43]. It can perform inference on MNIST in $<$1s, but its accuracy is much lower than CryptoNets since it only supports small BNNs.

TAPAS [45] also uses TFHE over BNN. It implements ripple carry adders for linear layers in a recursive manner to avoid errors due to accumulated noise in TFHE. However, to maintain a reasonable accuracy level, BNN often requires scaling the architecture, *i.e.*, having more outputs in each layer, which enlarges the circuits and cancels out part of the performance gain.

Unlike TAPAS, SHE [46] supports most common NNs using TFHE. It implements ReLU and maxpool as binary circuits. It also uses the power-of-2 idea [55], and running shifting circuits in TFHE is efficient. To efficiently handle the addition after the multiplication, it reduces the circuit size via a mixed-bit-width accumulator, whose size is smaller when the input size is smaller. SHE adopts an intra-layer mixed-bitwidth design for smaller circuits, which also allows the bitwidth to vary across layers to balance the performance and accuracy.

B. Pure-GC (Inference) Frameworks

The major hurdle of circuit-based approaches is that the circuits (even for linear layers) are bulky and incur a large overhead. DeepSecure [33] uses only GC. It decomposes the input (linear) layer into lower-rank matrices and releases one of them to the client for computing a low-rank projection (of the input). It also uses circuit optimization tools to approximate activation with fewer gates and skip computation for pruned neurons (in a pruned NN). XONN [63] is dedicated to BNN with pruning applied. It implements XNOR ($\odot$) in GC since BNN uses it to count the number of bits after multiplication.

"GarbledNN" [64] aims to show that GC can be applied to not only BNN but also "off-the-shelf" NNs, *e.g.*, the TensorFlow model. It uses arithmetic garbling techniques and optimizes for the sign function (and ReLU). Namely, it uses a *stochastic approach* that simplifies comparison gates (CMP) in GC by tolerating an error probability $<0.1\%$. Yet, this simplified circuit can only work on $\mathbb{Z}_N$, where N is a product of many distinct small primes. GarbledNN outperforms DeepSecure, but it still takes $\sim$15 minutes to run an NN with only 6 linear layers on CIFAR-10. As an approach orthogonal to XONN (for BNN), one might consider combining optimization tricks of both.

Open Problem 3. Tailoring ML techniques for compatibility with cryptographic techniques (vs. the other way round, *e.g.*, HE for linear and GC for non-linear layers, which many earlier PPNN works aim at) is a promising direction. We ask:

"Apart from BNN (and its generalization QNN to be discussed later), any ML techniques can help cryptographic NN?"

VI. Mixed Frameworks for Oblivious Inference

Many oblivious inference frameworks use both HE and SS and/or GC for the best of both worlds, *e.g.*, reduced runtime. We first explore SS-based computation (Delphi, CrypTFlow2, Circa, GForce) before revisiting HE (Gazelle, GALA, "FalconI").

A. Offline/Online Share Computation for 2PC (over GPU)

MiniONN [12] uses the offline/online trick for computing linear layers $\langle y_i \rangle = \sum_j w_{ij} \cdot \langle x_{ij} \rangle$ over arithmetic SS and weights w_{ij} known to the server. In the offline phase, the client sends $\{[\langle x_j \rangle_1]\}_j$ encrypting random $\{\langle x_j \rangle_1\}$ to the server, which then returns $\sum_j w_j \cdot [\langle x_j \rangle_1] - r'$ for a random r'. The client decrypts it to get $\langle y \rangle_0 = \sum_j w_j \cdot \langle x_j \rangle_1 - r'$. In the online phase, the client with input $\{x_j\}_j$ sends $\{\langle x_j \rangle_0 = x_j - \langle x_j \rangle_1\}_j$ to the server, which then computes $\langle y \rangle_1 = \sum_j w_j \cdot \langle x_j \rangle_0 + r'$. We call it *secure online/offline share computation (SOS)*. Note that $\{\langle x_j \rangle_1\}$ can be "reused" for different weights.

SOS-style multiplication of ℓ-bit integers [13], [65], [66] can also be done by OT. The client picks r'_i and sets $A_i = (r - r'_i, -r'_i), \forall i \in [0, \ell - 1]$ for random $\langle x \rangle_1 = r$. For each bit w_i of w in binary, the server gets $A_i[w_i]$ via OT and computes $\langle w \cdot r \rangle_0 = \sum_{i=0}^{\ell-1} 2^i A_i[w_i] = w \cdot r - \langle y \rangle_1$. The client then computes $\langle y \rangle_1 = \langle w \cdot r \rangle_1 = \sum_{i=0}^{\ell-1} 2^i r'_i$. (During the online phase, the server computes $\langle y \rangle_0 = w \cdot \langle x \rangle_0 + \langle w \cdot r \rangle_0$.) Since each OT sends $\approx \ell$ bits (after using correlated OTs and packing multiple instances for cost amortization), a multiplication takes $O(\ell^2)$ bits. Generalizing to matrices/tensors, the client/server obtains $\langle Y \rangle_0 = W \otimes R - R'$ or $\langle Y \rangle_1 = W \otimes (X - R) + R'$, respectively. For repeated dot products, multiple OTs are needed for varying weights, which takes more communication than using HE.

GPU for SS. In SOS, the client can generate an AHE of an additive mask offline. Once the input arrives, an SS-based

protocol is run on the client's masked input $\langle X \rangle$. The server in Delphi [67] and GForce [49] can use GPU to compute $W \otimes \langle X \rangle$ with performance close to plaintext computation. This observation generalizes SOS to the *AHE-to-SOS* trick [49] for many compatible operations (*e.g.*, truncation and GForce's wrap-around protocol below) and non-linear layers. Also, the offline protocol can be optimized separately (see Section VI-E).

B. Comparison Functions for sign, max, and ReLU

Non-linear layers in many popular NNs, *e.g.*, VGG and ResNet, only use ReLU and maxpool, which essentially compute $\langle \mathsf{max}(x,y) \rangle = \mathsf{CMP}_{x \leq ? y}(\langle x \rangle, \langle y \rangle) \cdot \langle y - x \rangle + \langle x \rangle$ (expressed in SS) using a *secure comparison protocol* CMP. CMP is also called the derivative of ReLU [68], [69] or *bit-extraction* for its equivalence to extract the MSB of $(y-x)$.

CMP via GC. ReLU and maxpool can be implemented by circuits with the comparison gate for CMP and multiplexer via the Beaver's trick after the Yao-to-Arithmetic (Y2A) conversion of ABY [35]. MiniONN computes $\langle \mathsf{ReLU}(x) \rangle = \mathsf{CMP}(\langle x \rangle, 0) \cdot \langle x \rangle$. Gazelle [48] computes the entire $\langle \mathsf{ReLU}(x) \rangle$ in GC. Many later works focus on optimizing the expensive CMP.

Selective Approximation. The online latency due to linear layers is <0.1 seconds by using GPU in Delphi. The bottleneck lies in non-linear layers. Delphi computes some ReLU layers by GC but others (strategically decided by a planner) by x^2-approximation. Accuracy is still degraded [49], albeit confined.

Stochastic Approaches. Circa [70] simplifies and thus accelerates CMP in GC, and truncates its input to suppress its error probability, which is linear in the input's absolute value.

DGK for GPU. DGK protocol of Damgård *et al.* [71] compares two HE-encrypted integers and outputs a result bit. Its naïve implementation is slower than GC/GMW's CMP. GForce [49] transforms DGK into a bunch of linear operations and fits them with its AHE-to-SOS trick, which outperforms Gazelle [48] and Delphi [67], also without any approximation.

An OT-based approach is proposed by CryptFlow2 [65]. We discuss a simplified version: For $x, y \in \mathbb{Z}_{2^\ell}$ respectively held by party P_0 and P_1, P_1 can *obliviously* select $\langle x \leq_? y \rangle_1$ from P_0 with $O(2^\ell)$ communication for bitwidth ℓ. To reduce it, CryptFlow2 decomposes x and y into k equal parts, *e.g.*, $x = x_k || \cdots || x_1$, and uses OT for oblivious selection over each pair of (x_i, y_i). Finally, they run a GMW-like CMP over $\{\langle x_i \leq_? y_i \rangle\}_1^{\ell/k}$ and $\{\langle x_i =_? y_i \rangle\}_1^{\ell/k}$ to get $\langle x \leq_? y \rangle$. Now the new cost is $O(2^{\ell/k} \cdot k)$. With a proper k, the cost can be lower than the typical GC/GMW approach, as OT is generally cheaper. This protocol does not support preprocessing like GForce, but the overall communication cost is modest ($90 \sim 120$ bits per input bit).

Below we pose a cryptographic challenge on the surface, which could be a scientific computation problem in disguise.

Challenge 1. *How to implement an even more efficient* CMP*?*

C. Bitwidth and Overflow Issues, Truncation and Wrap-Around

Dot products over fixed-points would easily overflow, *i.e.*, the absolute value exceeds $p/2$ for a p-sized plaintext space. As the most significant (~ 8) bits matter more empirically, *naïve truncation* is done by MiniONN, SecureML, and some subsequent (3PC) frameworks using SS (*e.g.*, [48], [67]). To truncate m bits, the server and the client locally divide their respective arithmetic SS $\langle x \rangle_S = x - r \bmod p$ and $\langle x \rangle_C = r \bmod p$ by 2^m.

However, an error occurs when a wrap-around happens in the secret share, *i.e.*, $x < r$, so $\langle x \rangle_0 = x - r + p$. In this case, $\langle x \rangle_0 / 2^m + \langle x \rangle_1 / 2^m = (x+p)/2^m \neq x/2^m$. The error probability is proportional to the message's magnitude and p. p needs to be large to suppress the error probability. The increased bitwidth (*i.e.*, $\lceil \log_2 p \rceil$) leads to a higher communication cost and bulky circuits in GC/GMW for non-linear layers.

Truncation protocol of GForce [49] outputs $\langle x/2^m \rangle = \langle x \rangle / 2^m + \langle b \rangle \cdot \lfloor p/2^m \rceil$, which offsets the wrap-around error by a wrap-around detection sub-protocol that outputs $\langle b \rangle = 1$ if $x < r$; 0 otherwise. There still could be an off-by-one error, but the impact on inference accuracy is negligible [49], [72].

GForce proposes an efficient sub-protocol that computes $\langle b \rangle$ with a constant multiplication (W) over additive SS ($\langle X \rangle$). It leverages the fact that wrap-around happens when the following conditions are both met: $\langle x \rangle_0 > p/2$ and $\langle x \rangle_1 < p/2$, assuming that $x < p/2$ [73]. This assumption can be met by swapping the ReLU layer before the truncation or offsetting x by a large public constant L first and then subtracting the truncated value by $L/2^m$. The server generates a random SS $\langle x \rangle_1 = r$ offline and eventually derives $\langle b \rangle = W \langle X \rangle$ where $W = (\langle x \rangle_1 <_? p/2)$ and $\langle X \rangle = \langle \langle x \rangle_0 >_? p/2 \rangle$ using the AHE-to-SOS trick. Since the client knows $\langle x \rangle_0$, it can locally compute X and send $\langle X \rangle_1$ to the server with an additive mask.

Most MPC-based frameworks [68], [74], [72], [75], [76], [77] (in Section VII) also face the fixed-point issues and need truncation after dot products. As they propose their own special kind of SS, they also come with their truncation protocols.

D. Quantized Neural Network (QNN)

Inputs, (intermediate) outputs, and weights of a QNN [78] are all fixed-points. QNN can be trained from scratch or post-training quantization (quantizing a trained DNN in floats), possibly fine-tuned using training data. QNNs are promising alternatives [55], [63], [74], [72], [49] for crypto-processing as they tolerate limited-precision ($\sim$8-bit) computation.

We highlight two works at two ends of the spectrum in a sense. "QuantizedNN" [72] directly adopted a quantized inference scheme [79] (originally designed for reducing the model size) to simplify the arithmetics and activations. GForce [49] adopts SWALP (stochastic weight averaging in low-precision training) [80] that trains a quantized model for inference with <1 percentage point (pp) of accuracy drop, further abstracts the (de)quantization to "SRT layers," with a more efficient truncation protocol, and analytically proves that the output distribution is close to such stochastic rounding after truncation. Interestingly, SRT is also GPU-friendly using the AHE-to-SOS trick.

With what the two previous subsections discussed, we ask:

Challenge 2. *How to utilize GPU for better performance, possibly with tricks to deal with the low-bitwidth constraint?*

Challenge 3. *Is there any other secure protocol for non-linear functions that can be made to fit with the AHE-to-SOS trick?*

E. Improving Efficiency of HE Operations

SOS computation reduces only online latency. Many mixed frameworks use various HE tricks to reduce the overall latency.

Denoising. Gazelle uses tight HE parameters to reduce latency, imposing a tight noise quota over HE operations (also see Section VIII-B). After each linear layer, it "refreshes" $[y+r]$, where y is the intermediate result, and r is a mask from additive SS, by sending it to the client to decrypt and re-encrypt it to clear the noise (vs. running the entire NN without the client). This requires a good network condition, and the client must remain online. A tempting trick is to let the client receive y, possibly with "noise" (but not masks) injected for efficient non-linear computation. Such heuristics were quickly proven insecure as a malicious client can average out the noise with multiple queries [81].

Optimizing/Reducing Rotations. SIMD is widely adopted, *e.g.* [12], but its rotation is not cheap (Section II-C5).

Gazelle handles linear layers by dedicating all SIMD slots in an HE ciphertext to a single inference query to reduce its latency. To align the slots for summation, it proposes a slot encoding and rotation scheme to reduce the number of ciphertexts the server sends back. LoLa [52] (a pure-LHE framework) also uses SIMD in the same way while minimizing the rotation cost according to the overall NN architecture.

GALA [82] further reduces the use of rotations. First, some can be cheaply replaced by rotating the plaintext weights during encoding. Second, the server can skip some rotation-and-sum on the ciphertexts by sending intermediate encrypted results with an additive mask to the clients, who will then decrypt them and (cheaply) sum the additively masked values during denoising.

"FalconI" [83] applies Fast Fourier Transform (FFT) on both weight tensors and input ciphertexts. Convolution then becomes point-wise multiplication. This approach avoids rotations. However, it is only suitable for (interactive) S/C frameworks and not pure-HE frameworks because it requires the client to perform inverse FFT on the resulting SS (converted from HE ciphertexts) for the subsequent non-linear layers. Interestingly, this CVPR paper [83] did not emphasize that it is motivated by avoiding rotation. This inspires the challenge and open problem below, which probably needs insights into the NN computations.

Challenge 4. *How to reduce the costs of HE SIMD/rotations?*

Open Problem 4. *We see various improvements in the mixed approach. Are there other scientific computation/machine learning techniques with cryptographic implications to explore?*

VII. MPC-based Inference/Training Frameworks

Many frameworks employ multiple non-colluding servers to run MPC. SS operations for MPC are very efficient. In general, they have the best performance. Many support private training.

A. Major Developments for MPC-based Inference

To support outsourced inference, SecureML [13] distributes the arithmetic shares of the weights to 2 servers to evaluate the matrix multiplication or convolution via Beaver's trick. It evaluates non-linear layers via GC or piecewise linear approximation (see Section VII-C). It also supports training over SS. ABY2.0 [84] supports more (online-)communication-efficient and higher fan-in multiplication and CMP by using a new kind of additive SS.

Say No to Beaver. Quotient [74] uses OT and GC to avoid (the expensive offline phase for) Beaver triplets, which improves the overall throughput. It is specifically designed for ternarized NNs with weights confined to $\{-1, 0, 1\}$. However, Quotient's efficiency and accuracy are only demonstrated on small NNs with a few linear layers (using matrix multiplication).

More Non-colluding Servers. Having more servers allows more options, *e.g.*, a third server can aid in preparing triplets without interactions between two servers. But it increases the risk. They might use 4 servers (see Table III), but an adversary still only needs 2 servers for a total break [50], [75], [85].

Chameleon [86] is a mixed framework that speeds up ABY with various tricks. It computes some gates over boolean SS (by GMW) and more advanced protocols by exploiting 3 servers, *e.g.*, a communication-cost-saving variant of Du–Atallah protocol [87] for efficient multiplication over additive SS.

ABY3 [88] extends ABY to 3PC over 3 servers. It proposes a new kind of 2-out-of-3 additive shares for less communication cost, which is generated via a pseudorandom function by exchanging short secret seeds. No Beaver's triplets are used.

SecureNN [68] and "FalconN" [69] propose 3PC variants of DGK comparison protocol using additive SS to outperform (non-colluding 3PC) GC/GMW-based ABY3 [88].

Trident [50] introduces the 4-th non-colluding server, which allows more SS kinds, enabling more efficient secure multiplication and comparison, and moving much computation and communication offline. With 32-CPU-core parallelism, Trident exhibits a much higher online throughput than ABY3, SecureNN, and FalconN in evaluation by orders of magnitude. In principle, many protocols can also be implemented in parallel.

Flash [75], Blaze [76], and Swift [89] also employ 4 non-colluding servers to enhance the throughput. They aim to defend against malicious adversaries, to be discussed in Appendix I-D.

B. Fluctuating Weights in Limited-bitwidth Training

Private and outsourced training frameworks run backward propagation with millions of iterations beyond forward propagation in inference. The only purely-crypto frameworks demonstrating reasonable training time (hours/days for common NNs) use SS under non-colluding assumptions [51], [69], [90].

Fluctuating weights make private training suffers even worse from fixed-point issues since increasing magnitudes (of weights, gradients, and intermediate inputs) in a low-bitwidth environment makes overflow happens easier. Meanwhile, the weights may become zeros when their magnitudes decrease, nullifying the training as the scaling factor may be too small. Most frameworks are yet to demonstrate running large NNs, *e.g.*, VGG, and more complicated datasets, *e.g.*, CIFAR-10/100.

Approaches for inference discussed in Section VI-C do not apply: With a higher overflow chance, fixed-scale truncation damages the training easily. Yet, we note that some underlying principles might remain relevant – One may aim for quantized NN (QNN) using low-bitwidth training. BatchNorm "for" non-linear function in Section VII-C might also "helps."

1. High Bitwidth. The private training frameworks usually use the 64-bit ring ($\mathbb{Z}_{2^{64}}$) for their underlying SS. In contrast to the 32-bit (or less) ring used by S/C frameworks (for oblivious inference), SS in $\mathbb{Z}_{2^{64}}$ doubles the communication and computation costs as it halves the throughput of vectorized operations on CPU/GPU, *e.g.*, Intel AVX. High bitwidth also leads to bulk circuits for the non-linear layers, harming the performance.

2. BatchNorm layers in private training are supported by FalconN [69] using 3 servers. Both the inputs in the forward propagation and the gradients in the backward propagation are normalized in a smaller range to cater to the low-bitwidth environment. Unlike inference, BatchNorm in training requires division and inverse square root where the divisors should be private and unknown prior to training, while it is public and fixed in inference. FalconN thus resorts to a costly iterative method (see Section VII-C) while leaking the input magnitude.

3. Adam. "AdamInPrivate" [90] adopts adaptive moment estimation (Adam), a more advanced SGD algorithm for training. To stabilize the gradient fluctuation between training batches, it scales the gradient by $1/\sqrt{G}$, where G is the moment, *i.e.*, the sum of the squares of the previous gradients. AdamInPrivate computes $1/\sqrt{G}$ using the techniques in Section VII-C.

4. Low-bitwidth QNN Training with "Small" Fixed-point Gradients. BatchNorm and Adam cannot fully resolve the bitwidth hurdles. When a DNN attains a certain level of accuracy, Δw (for updating the weight to $w - \alpha \cdot \Delta w$) may become so small that it has to be accumulated for many iterations before making noticeable changes to the weights. The training ceases to progress because these small gradients are discarded.

Quotient [74] adopts a QNN training scheme WAGE [91] (constraining weights, activations, gradients, and errors) to update the weights stochastically according to the magnitude of the gradient, such that small gradients still have an impact on the big weight. Yet, its attained accuracy is still inferior to floating-point training [91]. We note that perhaps deterred by the expensive GC for stochastical updates, other private training frameworks are yet to integrate low-bitwidth training schemes.

5. Softmax. Unlike inference, where the output layer usually outputs the index of the maximal element of its inputs, training requires the output layer to map its input via a differentiable function to enable backward propagation. A common choice is $\mathsf{softmax}(\mathbf{x}) = (e^{x_i}/\sum_j e^{x_j})_i$. Most schemes compute it with piecewise linear approximation. CrypTen [77] proposes a more efficient and accurate approach combining private division and the limit characterization of e^x (Section VII-C).

6. GPU for Training. Consider the dot product $\sum_i \langle w_i \rangle \cdot \langle x_i \rangle$, the bitwidth of the SSs cannot exceed 26 bits, not to mention redundant bitwidth to cater for additions. CryptGPU [51] utilizes GPUs for private training. It picks the 64-bit ring ($\mathbb{Z}_{2^{64}}$) for SS, which exceeds the bitwidth limit of GPUs. It thus decomposes an SS $\langle X \rangle$ into $2^{48}\langle X_4 \rangle + 2^{32}\langle X_3 \rangle + 2^{16}\langle X_2 \rangle + \langle X_1 \rangle$. Multiplying the low-bitwidth sub-SSs $\{\langle X_i \rangle\}_1^4$ requires many cross-term multiplications, offsetting some performance gain from GPU.

Short Summary. The high bitwidth training requirement conflicts with the low-bitwidth crypto environment. BatchNorm (Trick 2) normalizes the operands into a smaller range. Still, the gradients may become too small and easily underflow as the training goes on (4). The loss function softmax also requires accurate approximation (5). With the current techniques, we still need relatively high bitwidth representation (1), harming the training throughput, and the resulting accuracy could be shy of >30pp: CryptGPU [51] only attains $\sim$60% accuracy in CIFAR-10, while GForce [49] gets to >90% in oblivious inference [49]. Private training still desires more ML research, tackling Challenges 5-6.

Challenge 5. *How to cater dynamic weights in secure training?*

Challenge 6. *How to privately, efficiently, and accurately evaluate (both linear and non-linear) layers in low-bitwidth?*

C. Non-Linear Layers for Training and RNNs

In training, BatchNorm needs *private division* $1/y$, *inverse square root* $1/\sqrt{y}$; and softmax needs *exponential function* e^x.

SiRnn [92], as an oblivious-inference-only framework, also proposes protocols for these functions to support recurrent NN (RNN) [93], *e.g.*, with gated recurrent unit [94] or LSTM cells, which is good at sequential data like voice records and articles. RNN requires e^x for sigmoid, tanh, and normalization, but their piecewise linear approximation [12], [13] unbearably fails.

Piecewise Approximation partitions non-comparison-based functions, *e.g.*, sigmoid, into many linear functions, and uses CMP via GC to locate the piece of the input. SecureML and MiniONN adopt it. The accuracy increases with more dissections, but more CMP and multiplexers harm the performance.

Private Division is known to be more expensive as it frequently invokes the truncation protocol (cheaper as the divisor is public). FalconN [69] computes $\langle 1/y \rangle$ with Goldschmidt's method. With an initial guess $\langle w_0 \rangle$, it computes the approximation $\langle w_i \rangle = \langle w_{i-1} \rangle \cdot (1 + \langle \epsilon_{i-1} \rangle)$ with the error correction term $\langle \epsilon_i \rangle = \langle \epsilon_{i-1} \rangle^2$. (Here, we omit the scaling up and truncation.) Each iteration can double the precision (*i.e.*, quadratic convergence) – the absolute error $|w_t - 1/y|$ is less than $|w_0 - 1/y|^{2^t}$, but it takes 1 more communication round for each SS multiplication. It outputs $\langle w_t \rangle \approx \langle 1/y \rangle$ after t iterations.

A good initial guess reduces the iterations needed. FalconN adopts $\langle w_0 \rangle = 2.9142 - 2 \cdot \langle y \rangle / 2^\alpha$ from the prior art in private division [95]. α is the (unprotected) magnitude of y ($2^\alpha \le y \le 2^{\alpha+1}$). The fixed-point representation of w_0 and y are scaled up to precision $\alpha + 1$ to avoid value underflow. CrypTen [77] computes $\langle w_0 \rangle = 3 \cdot \langle e^{0.5-y} \rangle + 0.003$ (see below for $\langle e^y \rangle$). CryptGPU [51] assumes y is bounded in $[1, Y]$, which is applicable to softmax in private training but may not to sigmoid and tanh in RNN. The initial guess is set to $w_0 = 1/Y$.

AdamInPrivate [90] proposes to privately extract magnitude α by a circuit, use it to shift y to $1 - y' \in [0.5, 1)$, and compute $\langle 1/(1-y') \rangle = \langle 1 + y' + y'^2 + \ldots \rangle = \langle 1 + y' \rangle \cdot \langle 1 + y'^2 \rangle \cdot \langle 1 + y'^4 \rangle \cdots$ by $O(\log_2 t)$ squaring and multiplications for a t-degree approximation, and scale $\langle 1/(1-y') \rangle$ back to $\langle 1/y \rangle$.

Inverse Square Root. $\langle 1/\sqrt{y} \rangle$ can be evaluated by Newton's iterative method $w_i = \langle w_{i-1} \rangle \cdot (3 - \langle y \rangle \cdot \langle w_{i-1} \rangle^2)/2$ with quadratic convergence. AdamInPrivate privately extracts in a circuit $\lfloor \alpha/2 \rceil$ and hides w_0's value. FalconN picks $w_0 = 2^{\lfloor \alpha/2 \rceil}$.

Exponential Function. For $\langle e^x \rangle$, CrypTen [77] approximates $\langle e^x \rangle$ more efficiently by computing the limit characterization

$(1 + \langle x \rangle / 2^t)^{2^t}$. It takes t iterations of SS multiplications to reduce relative error to $e^{O(-x^2/2^t)}$ for $|x| < 2^t$.

Lookup Table. Borrowing memory-saving tricks in embedded systems, SiRnn [92] extends a provably precise lookup approach for getting a good initial guess before using iteration to improve upon the approximation. Namely, it obliviously looks up d-digit $\langle \mathsf{exp}(2^{d\cdot(i-1)} x_i) \rangle$ over $\langle x_i \rangle$ for $x = x_k || \cdots || x_1$ and multiplies the shares as $e^x = \mathsf{exp}(2^{d\cdot(k-1)} \cdot x_k + \cdots + 2^d \cdot x + x_1)$. An upside is that the oblivious lookup can be parallelized, while iterative approaches are inherently sequential.

The above approaches only work on additive shares. Glyph [58] uses the lookup table implemented by TFHE and BGV-HE to compute sigmoid and softmax. Yet, it is impractically slow due to the overhead caused by the FHE schemes. Lately, Heath and Kolesnikov [96] propose one-hot garbling that allows efficient privacy-free lookup tables in garbled circuits. It would be interesting to know if it can speed up non-linear layers, especially in a high-latency communication environment where GC trumps OT.

To sum up, many tricks are inspired by numerical approximations, possibly with a new angle, such as the lookup approach for iterative computations. Their accurate approximations are also crucial for private training. We pose two challenges:

Challenge 7. *How to efficiently and accurately approximate* x/y, $1/\sqrt{y}$, e^x, $\mathsf{sigmoid}(x)$, *and* $\mathsf{tanh}(x)$ *for secret* x *and* y*? More broadly, is there any other crypto-friendly numerical method for mathematical functions useful for NN?*

Challenge 8. *Can we generalize the lookup approach or extend other low-bitwidth tricks of embedding systems to other non-linear functions for more efficient approximations?*

VIII. Practical Deployment Issues

A. Neural Network Architecture Optimization

Tuning NN architecture can cut the cost of cryptographic NN computation with moderate inference accuracy degradation.

Pruning *outsourced* inference would let the server learn the number of zeroed weights from how much computation is saved. Such leakage is not a concern in non-outsourced inference. Still, a custom implementation of SS operations is needed to take advantage of zeroed weights as they are hidden by SS, abandoning off-the-shelf optimized math libraries. This asks for dedicated customization and secure programming effort to ensure no potential side-channel leakage.

Open Problem 5. *How to build a leakage-free data-oblivious library exploiting data-dependent optimization like pruning?*

Cost Optimizers tune the architecture to optimize an objective function, factoring in the inference accuracy and the cost of cryptographic primitives. NASS [97] performs an automatic search for NN architecture and HE parameters to attain the best performance and accuracy over Gazelle [48]. COINN [66] optimizes over the bit-width of operands in each linear layer.

Weight Clustering. Observing $\sum w_i \cdot x_i$ over N weights $\{w_i\}$ can be factorized into $\sum_{i\in[1,V]} w_i \sum_{j\in S_i} x_j$ over V unique weights after clustering, where S_i is the index set, COINN uses N OTs to select and accumulate $\sum_{j\in S_i} X_j$ for $i \in [V]$, resulting in $\approx V N \ell$ bits of communication over ℓ-bit arithmetic shares (vs. $\approx N\ell^2$, as in Section VI-A). Then, it computes the shortened dot product over size-V vectors with $V\ell^2$ bits of communication. The total cost is reduced to $V\ell(N + \ell)$ with COINN's bit-width optimization ($N \gg \ell$) when $V < \ell$.

Reducing ReLU**.** With the huge performance gap between polynomial approximation and CMP, Delphi's planner [67] searches for NNs that replace ReLU layers by quadratic approximation with mild accuracy degradation. SafeNet [98] provides a better tradeoff by a more fine-grained replacement of ReLU by approximation on some channels instead of a whole layer.

CryptoNAS [99] reduces the number of ReLU's by observing that a ReLU layer in ResNet may skip some or even all inputs, and just outputs identical values. DeepReDuce [100] further subsamples the inputs to ReLU layers. It adopts knowledge distillation [101] to compensate for the accuracy loss, which retrains the ReLU-reduced NN with the input-output pairs of the original NN. Surprisingly, the retrained NN may even attain higher accuracy than the original NN.

Despite their empirical performance, they do not appear to be the optimal search for ReLU reduction. We thus ask:

Challenge 9. *How to "optimally" approximate* ReLU *layers while mildly affecting (or even improving) accuracy?*

Cost optimization can be seen as a pursuit complementary to Open Problem 2, which explores the use of polynomials as crypto-friendly operations to approximate non-linear layers. ML research for broader classes of crypto-friendly computations (beyond polynomials) remains largely unexplored. Despite this, this SoK discussed two different kinds of fusing from the system (nGraph-HE [102]) and crypto (GForce [49]) communities. The latter can minimize non-crypto-friendly computations. Broadly:

Open Problem 6. *Are there other "fusing" opportunities that can reduce cryptographic operations in PPNN?*

Hyperparameters affect how ML algorithms perform. Hyperparameter tuning aims to find out the best configuration of the training algorithm that will output a model generalizing to new data well. There is growing attention to hyperparameter optimization (HO) since tuning procedures could be tedious.

Open Problem 7. *Are there any HO opportunities beneficial to PPNN, say, minimizing the cost of cryptographic NN computation while maintaining an "acceptable" accuracy level?*

Open Problem 8. *Can we realize "private training with HO"?*

B. Deploying Optimized HE Frameworks

To sustain the magnitude growth in iterated ciphertext operations, pure-LHE frameworks using BFV-HE or BGV-HE, *e.g.*, CryptoNets and FasterCryptoNets [55] deal with the overflow issue by picking large plaintext space. CKKS-HE allows discarding the least significant bits by rescaling; each such operation requires large HE parameters. These enlarged HE parameters increase the runtime and communication cost. It explains why pure-LHE frameworks are not efficient enough for deep NNs.

HE parameters include the finite-field size for the plaintext (*i.e.*, bitwidth), the degree of the polynomial representing the ciphertexts (often also the numbers of SIMD slots), and

the coefficient's moduli of the polynomial (determining the number of "levels" of HE operations and noise tolerance). Each operation accumulates noise in a ciphertext, which may eventually destroy its plaintext. HE parameters determine how much noise it can withstand.

It is not easy for non-cryptographers to pick the optimal parameters under constraints with cryptographic implications. Large parameters lead to larger ciphertexts and more expensive computations. For a good balance of security and efficiency, one should carefully choose *tight parameters* if possible.

Challenge 10. *How to guide non-cryptographers to select "tight" HE parameters or avoid the troubles of the noise?*

Compilers are proposed to do it w.r.t. NN architecture. We defer them to Section VIII-C for their multiple purposes [102].

Selecting tight HE parameters is critical for performance. A few frameworks automatically pick just-enough choices to prevent overflow with operational costs confined. They also select scaling factors for the fixed-point representation. CHET [103] derives HE parameters based on the training dataset and the computation graph (for the NN architecture). SEALion [104] adopts a similar approach. nGraph-HE2 [105] selects small parameters (32-bit moduli) to reduce CPU instructions for HE.

Optimizing the data layout can lower rotation costs (*e.g.*, for dot product, see Section VI-E). Most frameworks automatically encode (plaintext/encrypted) tensors into the HE polynomials, without developers noticing the underlying data layout. CHET [103] further optimizes the data layout to minimize the overall rotation cost, as different rotations cost differently.

Fusing or skipping operations is common in compiler optimization. nGraph-HE [102] parallelizes HE operations (on CPU), exploiting the SIMD of HE over the batches of inputs, fusing the multiplications in BatchNorm layers, and skipping some HE operations when the plaintext is $0, 1$, or -1. nGraph-HE2 [105] groups up or skips expensive denoising.

Optimizing over the hardware intrinsics is done by PlaidML-HE [106] for more efficient HE operations and linear layers. It covers CPUs and GPUs of various brands and CKKS-HE [41] for its residue number system.

If we had an HE scheme that could be instantiated with small parameters, it immediately reduces the communication cost. Likewise, cutting the bootstrapping time (*e.g.*, of TFHE) also reduces the latency. Cryptographers' research efforts are mostly needed for the problems/challenges below.

Open Problem 9. *How to improve HE's performance?*

Challenge 11. *How to perform low-latency inference under limited-bandwidth conditions without non-colluding servers?*

Challenge 12. *How to efficiently realize secure outsourcing of inference for deep NNs without non-colluding assumptions?*

C. Compilers for Rapid Development and Optimization

Developer-Friendly Compilers. Many works proposed a specific compiler that compiles a (plaintext) NN, including the architecture and the weights, into a PPNN, often with optimized cryptographic computation. This allows coding with a standard machine-learning library or high-level APIs without knowing much about (optimizing) cryptographic computation, reducing laborious hand-engineering and possible coding errors.

Graph-Based Compilers transform an NN representation of an NN library, *e.g.*, TensorFlow or PyTorch, into the "cryptographic version" for oblivious inference. They include nGraph-HE/2 [102], [105], PlaidML-HE [106], and CrypTFlow [107].

Custom APIs are proposed by some frameworks to help developers adopt them. EzPC [108] is a general compiler for 2PC that converts its Python-like source code to C++, calling ABY API functions for oblivious inference with 2 non-colluding servers. XONN [63] supports high-level API for BNN layer specification and translates an NN description from other DNN libraries into its API. SEALion [104] uses Keras-like syntax for implementing PPNN, while CHET [103] uses TensorFlow-like syntax. CrypTen [77] proposes PyTorch-style APIs for 2PC protocols (mainly from FalconN's protocol [69]).

GPU Platform for Arithmetic SS. Piranha [109] is a platform for MPC from arithmetic SS to run on top of GPU. Its modular design allows easier integration of new SS-based protocols (for a certain layer) than re-implementing the entire framework.

These compilers bring convenience to developers. However, PPNN researchers often still need much engineering effort to implement specific tricks they devised, let alone re-implement existing ones for comparison when they are often implemented with different languages on top of different libraries[7] with slightly different low-level optimization. To let researchers focus on what they are good at and save resources from incessant replications of old results (*e.g.*, 96 cores [56], Nvidia V100 GPU [49], [51], [56], [109], processing 1TB of data for an encrypted query [55]), a rapid benchmarking platform is desired:

Challenge 13. *Can we build a universal compiler that enables rapid prototyping and allow uniform experimental comparison?*

Optimization. EzPC [108] optimizes performance over the choices of A/B/Y SS for oblivious (program evaluation and) inference. Most others optimize arithmetic HE:

Challenge 14. *Can we build an end-to-end compiler like PlaidML-HE over other major HE schemes such as TFHE?*

D. Leakages of Metadata and Difficult-to-Hide Information

Convolution Parameters. Most frameworks leak the stride, padding, and kernel size in convolution layers. When using circuit-based primitives in the outsourced setting (*e.g.*, TFHE, GC, and GMW), adversaries can derive them from the circuit's topology. Likewise, they can be leaked from the computation steps by an outsourced server unless they are locally done.

Dimensions of Layers refer to the input and output size of the layers. For outsourced or server/client frameworks, they can be derived from the intermediate SSs or HE ciphertexts. For pure-HE inference frameworks, the client does not see the intermediate ciphertexts and hence does not see the leakage. **Types of Layers** are similar to dimensions since PPNN frameworks do not hide the computed functions in general.

[7] *e.g.*, CrypTen: https://github.com/facebookresearch/CrypTen, Delphi: https://github.com/mc2-project/delphi, EzPC: https://github.com/mpc-msri/EzPC, GForce: https://github.com/Lucieno/gforce-public

Total Number of Layers. Leaking the dimension also leaks the total number of layers. LHE frameworks still leak this information because the client can deduce the number of multiplications performed. Usually, a linear or activation layer requires one.

E. Are Practitioners Willing to Compromise?

The state-of-the-art [51] private training using 3 GPUs is still far from the performance of plaintext training. Meanwhile, the ML community has started considering solutions using trusted execution environments and lightweight cryptographic outsourcing protocols for leveraging (untrusted) GPUs [16]. With the performance gap from plaintext training, some might care less about trust assumptions or malicious security.

Open Problem 10. *Is there any gap between theoretical security guarantees and what practitioners are (un)willing to trust? How should those perspectives inform PPNN research efforts?*

The answer depends on the application scenarios and deployment settings. On one hand, there are highly sensitive scenarios, *e.g.*, child exploitation imagery detection, in which one might expect the highest security level. On the other, it can be a life-saving effort to speed up the progress in, *e.g.*, cancer diagnosis [16], genomic precision medicine, drug-target interactions [110], and reliance on the non-colluding assumption could become a secondary concern [111]. One may have faith in a privacy-respecting government or privacy advocates for the needed trust. Meanwhile, industry players might not welcome non-colluding assumptions since the idea of trusting competitors can be illusory to some.

IX. Status Quo in Accuracy/Latency/Throughput

We integrate evaluation results presented by the papers covered by this SoK in the same plots for general trends. Each framework shines in its way in its targeted settings. We highlight noteworthy points in the space. We then present re-evaluations of the most promising frameworks we identified.

Below, we discuss typical benchmark datasets in the literature. More complicated tasks often require a more complex NN architecture with more internal parameters for high accuracy.

MNIST is a dataset for classifying black-and-white handwritten digitals over 28×28 images, which can be handled via a multilayer perceptron (MLP) NN with 1 fully-connected layer at $>99\%$ accuracy. It can no longer demonstrate the superiority of new advancements in rapidly developing PPNN research.

CIFAR-10/100 became popular for benchmarking. They are datasets consisting of 10/100 classes of 32×32 colorful images of vehicles and animals, *e.g.*, dog, ship, and truck, which are harder to classify than the simpler MNIST, and require convolution layers for better accuracy. VGG-16 [112], having 16 linear layers, is a commonly chosen architecture to handle CIFAR-10/100. ResNet [113] can attain a higher accuracy but also incurs longer inference latency as it has more linear layers.

ImageNet is an even more challenging object recognition dataset. It consists of >14 millions 224×224 colorful images with >20000 classes. Handling this dataset requires capacity beyond current PPNN frameworks. Still, one [65] may process 224×224 (ImageNet-scale) images to estimate its performance on such large images and claims that it loses no accuracy given enough bitwidth. A few frameworks [55], [69] shrink the dataset into Tiny ImageNet with $120,000$ 64×64 images under 200 classes to demonstrate their ability beyond CIFAR-10/100. Unfortunately, they rarely provide accuracy, probably because fixed-point issues are yet to be overcome for the large NNs needed. It is hard to have a fair comparison.

To put accuracy and latency into perspective, we may also factor in: i) the problem setting (oblivious/outsourced inference or outsourced/private training), ii) the chosen dataset, iii) the network condition (*e.g.*, LAN or WAN), iv) the number of non-colluding servers, and v) online versus total performance.

A. Oblivious Inference

We summarize the performance of 23 frameworks marked with CIFAR-10 in Table IV (except AdamInPrivate with only training benchmarks). The tests used CIFAR-10 and LAN.

- 7 pure-HE frameworks: CHET [103], HCNN [56], FasterCryptoNets [55] (or Faster-Crypt in Tables/Figures), CryptoDL [54], nGraph-HE [102], LoLa [52], and SHE [46];
- 13 server/client (S/C) frameworks: CrypTFlow2 [65], XONN [63], Delphi [67], GForce [49], FalconI [83], DeepSecure [33], EzPC [108], Gazelle [48], MiniONN [12], GarbledNN [64], COINN [66], SafeNet [98], and CryptoNAS [99];
- 3 non-colluding frameworks: FalconN [69], CryptGPU [51], Piranha [109]'s GPU-instantiation of FalconN and Fantastic 4.

We first provide an integrated report of their accuracy, latency, and throughput. Our figures aim to give readers a sense of differences among different paradigms (marked with different colors) and the significance of GPU-friendly protocols. We also aim to identify the state of the art in each paradigm. Thus, the figures only name the Pareto fronts of each paradigm, which also avoids overcrowded name tags. Multiple *data points* in each figure can be contributed by a single framework over different NN architectures (all tested with CIFAR-10).

Most existing tests are on LAN, which may not be realistic. To better understand the identified state of the art, we re-evaluate them on WAN with various bandwidths to unveil which framework performs the best under a non-ideal network. It gives us clues on deployment requirements, particularly the minimal bandwidth needed for their peak performance.

For readability, we omitted 8 outliers, from the following 3 frameworks, due to their low accuracy: nGraph-HE (62%), COINN (68.1% – but it has *another* data point on the Pareto frontier for throughput), and Piranha (40% and 55% – two data points nevertheless on the Pareto frontier for 3.25×10^6 Images/hr throughput and 131ms latency).

1. Online Latency. Figure 1 shows the inference accuracy with respect to the online latency, which is more crucial than total latency for user experiences. GForce [49] strikes the best balance and surprisingly outperforms non-colluding MPC frameworks like CryptGPU [51] and FalconN [69] for inference (albeit supporting training). It may be attributed to its adoption of QNN and GPU-friendly comparison protocol.

Pure-HE frameworks perform the worst in this setting. For CIFAR-10, deep NNs are usually needed for reasonable accuracy, leading to huge HE parameters and worse performance.

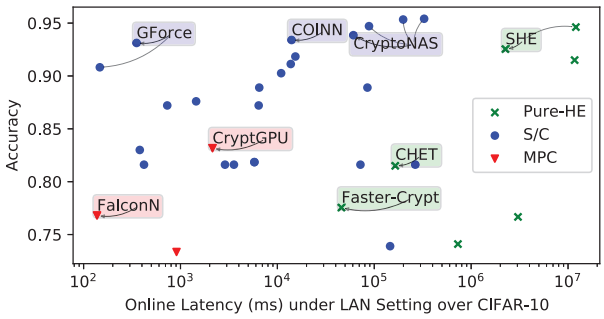

Fig. 1. Online Latency of Inference on CIFAR-10 (*top-left* ones are better)

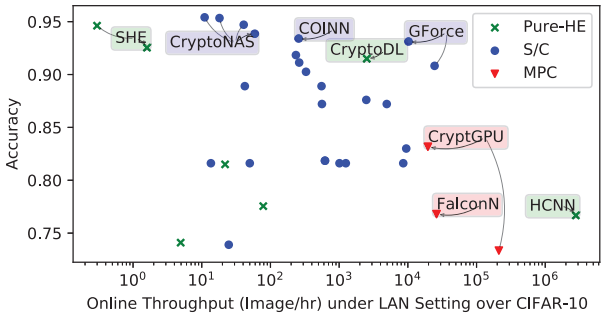

Fig. 2. Online Throughput of Inference on CIFAR-10 (*top-right* ones are better)

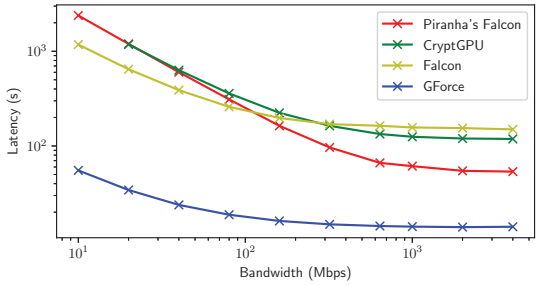

Fig. 3. Our Re-evaluation for Latency over WAN (The *lower* the better)

2. Online Throughput. Figure 2 shows that CryptGPU defeats GForce in throughput, probably for amortizing the communication cost between the 3 servers over its batched queries.

CryptoDL [54], using only LHE, achieves a throughput close to GForce [49] and CryptGPU, the representative of the other approaches. It encodes all batched queries in an HE ciphertext to avoid expensive rotations.

For accuracy, circuit-based SHE [46] is the best, albeit the worst in throughput. Other pure-HE frameworks, *e.g.*, CryptoDL, attain accuracy inferior to GForce and CryptGPU, the representative of the other 2 approaches. It shows that polynomial approximation bound to HE frameworks harms accuracy.

3. GPU Trumps. Figures 1 and 2 show the important role of GPU. CryptGPU, GForce, and HCNN [56] all utilize GPUs and achieve the shortest latency or the highest throughput among the frameworks of their respective major approach. We expect to see future frameworks will continue to involve GPUs.

Open Problem 11. *What can other cryptographic primitives (for non-linear operations) be made GPU/TPU-friendly?*

4. Re-evaluation over WAN. The representative S/C and MPC frameworks are evaluated on LAN. We re-evaluate GForce [49] (for S/C), CryptGPU [51], FalconN [69], and the Piranha [109] version of FalconN on GPU (P-FalconN). (for MPC) to understand their performance under non-ideal network conditions. For LAN, they are state-of-the-art – GForce has the lowest latency, and all of them have (almost) the highest throughput. For Piranha, a GPU platform for MPC frameworks, we pick its instantiation for FalconN as it demonstrates the best performance among all existing instantiations. The evaluation is done on a Google Cloud machine with 4 Nvidia V100 GPUs, an Intel Xeon 2.2GHz CPU with 32 cores, and 64GB RAM. We set the communication latency to 50ms because it is the latency for data centers communicating within a continent.[8]

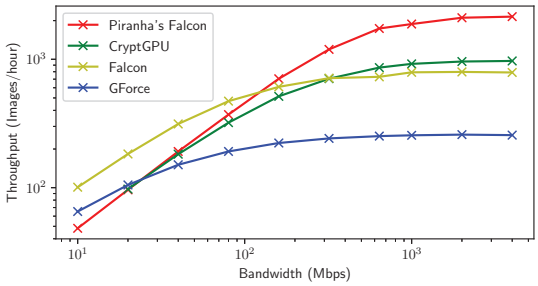

Fig. 4. Our Re-evaluation for Throughput over WAN (The *higher* the better)

Figure 3 shows their online latency in a WAN setting with 50ms communication latency using the Linux tc command for VGG-16 over CIFAR-10. GForce demonstrates the lowest latency, an order of magnitude lower than others. Its latency is $\sim$10s for $\geq$100Mbps bandwidth and remains $<$1 minute even under $\sim$10Mbps, which seems good enough for non-real-time usage. Notably, these frameworks still have shorter latency than the pure-HE frameworks in WAN. With only 10$\sim$100Mbps, the re-evaluated MPC and S/C frameworks attain $10^2\sim10^3$s latency, which outperforms Faster-Crypt (with the shortest latency of $\sim10^5$s among pure-HE frameworks in Figure 1).

Figure 4 shows their online throughput under the same experimental setting. They show a decrease from the order of 10^4 images/hr in LAN to $10^2\sim10^3$ in WAN. Similarly, their inference latency is increased by 100$\times$. When the bandwidth is high ($\geq$100Mbps), their relative order remains similar to Figure 5. Compared to pure-HE frameworks, these MPC and S/C frameworks have lower throughput ($10^2\sim10^3$ images/hr) than HCNN ($\sim10^4$ images/hr) and CryptoDL ($>10^3$ images/hr). It is expected because pure-HE is insensitive to network conditions and can leverage SIMD for higher throughput.

All frameworks appear to need $\sim$1000Mbps to reach their peak performance. Notably, FalconN is less sensitive to low bandwidth as it starts to outperform Piranha and CryptGPU in extremely low bandwidth of 10 $\sim$ 100Mbps.

5. MNIST over WAN. Pure-HE frameworks often perform the best in WAN settings for not requiring interaction and hence less affected by the communication latency. Yet, they support a limited depth of NNs and limited choices of non-linear layers, so they usually only handle MNIST, an easier dataset. We did

[8]https://www.cloudping.co/grid

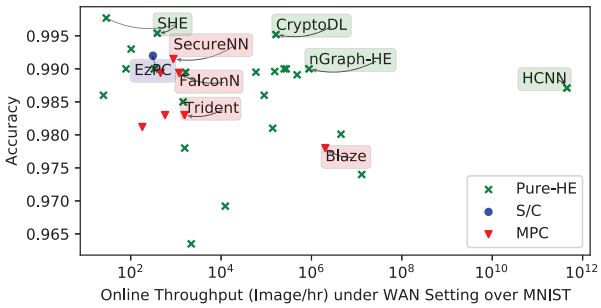

Fig. 5. Online Throughput of Inference on MNIST (*top-right* ones are better)

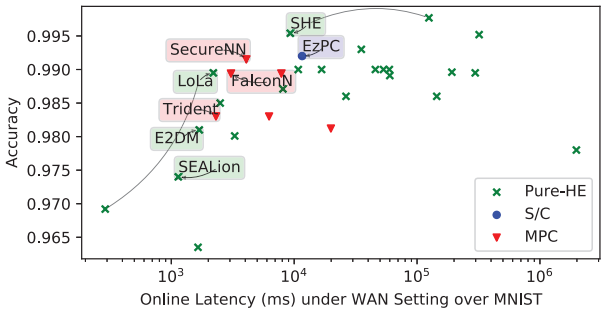

Fig. 6. Online Latency of Inference on MNIST (*top-left* ones are better)

not perform re-evaluations for MNIST being an unrealistically easy dataset to reflect performance for real tasks.

The definition of WAN varies across papers. Their network traffic latency ranges from 40 to 150ms, and 40 to 320Mbps for their bandwidth. Figure 5 and 6 includes 21 frameworks:

- 13 pure-HE frameworks: CryptoNets [11], CryptoDL [54], E2DM [57], Faster-Crypt [55], FHE-DiNN [44], TAPAS [45], CHET [103], nGraph-HE [102], SEALion [104], SHE [46] LoLa [52], Glyph [58], and HCNN [56].
- 7 MPC frameworks: FalconN [69], Quotient [74], Blaze [76], SecureNN [68], Trident [50], SecureML [13], and ABY3 [88].
- 5 outliers, namely, ABY3, FHE-DiNN, SecureML, Trident, and SecureNN, are omitted as their accuracy is only $\sim$93%.

EzPC [108] is the only S/C framework included since most S/C frameworks skip their WAN evaluation.

Figure 5 shows their online throughput in WAN. HCNN (using 4 GPUs) has the highest throughput. CryptoDL and nGraph-HE also strike a good balance between accuracy and throughput. These three frameworks use HE's SIMD for batch processing. SecureNN, FalconN, Trident, and Blaze are the Pareto frontiers of the MPC frameworks. Yet, they are all inferior to the frontiers of the pure-HE frameworks as they require more interactions.

For online latency, LoLa reduces the rotation cost of SIMD, and Figure 6 shows its good balance with accuracy. In general, pure-HE remains better than MPC, but their gap is much narrower since SIMD does not help as much. (SecureNN is also on the Pareto frontier for 130ms latency. Yet, it is omitted from Figure 6 due to its low accuracy (93.4%).)

B. Private Training

Among the 9 frameworks for private (and outsourced) training, CryptGPU [51], FalconN [69], Piranha [109], and AdamInPrivate [90] are the only fours tested on CIFAR-10 or any harder datasets using deep NNs (VGG-16). As shown in Figure II, CryptGPU has the highest accuracy of 82.3%, while P-FalconN (Piranha's instantiation on FalconN) has the highest throughput. Yet, CryptGPU may still be the winner since accuracy is usually more important in NN training, when CryptGPU's throughput (9489 images/hr) is only slightly lower than P-FalconN's (15152 images/hr).

TABLE II
REPORTED ACCURACY AND THROUGHPUT ON LAN

Training Framework	Accuracy	Throughput (Image/hr)
FalconN	76.8%	1482
CryptGPU	82.3%	9489
P-FalconN	55.1%	15152
AdamInPrivate	75%	4171

We note that frameworks for oblivious inference can attain >90% accuracy with the same architecture (VGG-16) adopted by CryptGPU. It implies the potential for improvements.

Challenge 15. *How to attain a higher level of accuracy in private training? For instance, what other (normalization) layers can we use with what other approximation approaches?*

X. Discussion

We give some brief guidance on what PPNN frameworks to use. The threat models and performance are the main concerns. If the users trust some non-colluding servers, they can outsource the NN computation for the best-in-the-class performance since non-colluding MPC frameworks often feature efficient outsourced training and inference.

If the users do not trust non-colluding servers, they have two choices left: 1) For inference expecting nearly real-time results, *e.g.*, facial recognition, they can run S/C frameworks, *e.g.*, GForce [49], with a reasonably good network ($\leq$ 50ms network latency and $\geq$ 100Mbps) and using pre-computation to save online inference latency. 2) For applications that can wait or involve many sub-queries, *e.g.*, medical diagnosis and financial data analysis. These applications can use pure-HE frameworks, *e.g.*, CryptoDL, to achieve high throughput (but high inference latency) using SIMD, *i.e.*, running the queries in batches.

Without non-colluding servers, E2DM and GarbledNN support outsourced inference, and Glyph supports outsourced training. Yet, their performance is far from the S/C and pure-HE frameworks. For detailed discussions about the state-of-the-art frameworks in different settings, see Appendix I and Fig. 7.

We end with remarks on two popular application areas. Natural language processing often runs on RNNs, which require heavy computation due to recurrent calls to some gigantic NN that contains complex non-linear functions. SiRnn [92] is the only efficient (non-colluding) framework for oblivious inference. CryptGPU, FalconN, and AdamInPrivate, being non-colluding frameworks, also support those complex non-linear functions and thus should be able to train and infer over RNNs. Graph NNs, which apply NNs for graphs, can use the techniques in this SoK in oblivious settings. Yet, it poses new challenges on how to hide the topology of the graphs.

REFERENCES

[1] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *Usenix Security*, 2016.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR*, 2015.

[3] N. Papernot, P. D. McDaniel, A. Sinha, and M. P. Wellman, "SoK: Security and privacy in machine learning," in *EuroS&P*, 2018.

[4] M. S. Riazi, B. D. Rouhani, and F. Koushanfar, "Deep learning on private data," *IEEE Secur. Priv.*, vol. 17, no. 6, pp. 54–63, 2019.

[5] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *MM&Sec*, 2006.

[6] C. Orlandi, A. Piva, and M. Barni, "Oblivious neural network computing via homomorphic encryption," *EURASIP J. Info. Sec.*, 2007.

[7] A. C. Yao, "How to generate and exchange secrets," in *FOCS*, 1986.

[8] A. Sadeghi and T. Schneider, "Generalized universal circuits for secure evaluation of private functions with application to data classification," in *ICISC*, 2008.

[9] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *STOC*, 1987.

[10] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009, PhD thesis available at https://crypto.stanford.edu/craig.

[11] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *ICML*, 2016.

[12] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *CCS*, 2017.

[13] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *S&P*, 2017.

[14] H. C. Tanuwidjaja, R. Choi, and K. Kim, "A survey on deep learning techniques for privacy-preserving," in *ML4CS*, 2019.

[15] M. Azraoui and *et al.*, "SoK: Cryptography for neural networks," in *Summer Sch. on Pri. & Id. Mgt. (Data for Better Living)*, 2019.

[16] L. Ng, S. Chow, A. Woo, D. Wong, and Y. Zhao, "Goten: GPU-outsourcing trusted execution of neural network training," in *AAAI*, 2021.

[17] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *S&P*, 2017.

[18] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information & basic countermeasures," in *CCS*, 2015.

[19] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006, pp. 265–284.

[20] F. Mireshghallah, M. Taram, P. Vepakomma, A. Singh, R. Raskar, and H. Esmaeilzadeh, "Privacy in deep learning: A survey," arXiv, 2020.

[21] M. Du, X. Yue, S. S. M. Chow, and H. Sun, "Sanitizing sentence embeddings (and labels) for local differential privacy," in *The Web (WWW)*, 2023, pp. 2349–2359.

[22] M. Du, X. Yue, S. S. M. Chow, T. Wang, C. Huang, and H. Sun, "DP-Forward: Fine-tuning and inference on language models with differential privacy in forward pass," in *CCS*, 2023, to appear.

[23] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *Trans. Intell. Sys. Tech.*, vol. 10, no. 2, 2019.

[24] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *S&P*, 2019.

[25] M. Naor, B. Pinkas, and O. Reingold, "Distributed pseudo-random functions and KDCs," in *EUROCRYPT*, 1999, pp. 327–346.

[26] M. Chase and S. S. M. Chow, "Improving privacy and security in multi-authority attribute-based encryption," in *CCS*, 2009, pp. 121–130.

[27] K. A. 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 *CCS*, 2017, pp. 1175–1191.

[28] J. Cabrero-Holgueras and S. Pastrana, "SoK: Privacy-preserving computation techniques for deep learning," *PoPETs*, no. 4, 2021.

[29] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.

[30] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM MM*, 2014.

[31] M. Abadi and *et al.*, "TensorFlow: A system for large-scale machine learning," in *OSDI*, 2016.

[32] F. Tramèr and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *ICLR*, 2019.

[33] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "DeepSecure: scalable provably-secure deep learning," in *Design Auto. Conf. (DAC)*, 2018.

[34] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *CRYPTO*, 1991.

[35] D. Demmler, T. Schneider, and M. Zohner, "ABY - A framework for efficient mixed-protocol secure two-party computation," in *NDSS*, 2015.

[36] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *CRYPTO*, 2003.

[37] G. Dessouky, F. Koushanfar, A. Sadeghi, T. Schneider, S. Zeitouni, and M. Zohner, "Pushing the communication barrier in secure computation using lookup tables," in *NDSS*, 2017.

[38] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical GapSVP," in *CRYPTO*, 2012.

[39] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," Cryptology ePrint, 2012/144, 2012.

[40] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, 2014.

[41] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *AsiaCrypt Part I*, 2017.

[42] D. J. Wu and J. Haven, "Using homomorphic encryption for large scale statistical analysis," Stanford, Tech. Rep., 2012.

[43] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *J. Crypt.*, vol. 33, 2020.

[44] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *CRYPTO III*, 2018.

[45] A. Sanyal, M. J. Kusner, A. Gascón, and V. Kanade, "TAPAS: Tricks to accelerate (encrypted) prediction as a service," in *ICML*, 2018.

[46] Q. Lou and L. Jiang, "SHE: A fast and accurate deep neural network for encrypted data," in *NeurIPS*, 2019.

[47] C. Boura, N. Gama, M. Georgieva, and D. Jetchev, "Chimera: Combining ring-LWE-based fully homomorphic encryption schemes," in *Number-Theoretic Methods in Cryptology (NuTMiC)*, 2019.

[48] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "Gazelle: A low latency framework for secure neural network inference," in *Usenix Security*, 2018.

[49] L. K. L. Ng and S. S. M. Chow, "GForce: GPU-friendly oblivious and rapid neural network inference," in *Usenix Security*, 2021.

[50] R. Rachuri and A. Suresh, "Trident: Efficient 4PC framework for privacy preserving machine learning," in *NDSS*, 2020.

[51] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the GPU," in *S&P*, 2021.

[52] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *ICML*, 2019.

[53] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," Cryptology ePrint 2017/035, 2017, also presented at Real World Crypto.

[54] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *PoPETs*, no. 3, 2018.

[55] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and F.-F. Li, "Faster CryptoNets: Leveraging sparsity for real-world encrypted inference," arXiv:1811.09953, 2018.

[56] A. A. Badawi and *et al.*, "Towards the AlexNet moment for homomorphic encryption: HCNN, the first homomorphic CNN on encrypted data with GPUs," *IEEE Trans. Emerg. Top. Comput.*, vol. 9, no. 3, pp. 1330–1343, 2021.

[57] X. Jiang, M. Kim, K. E. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *CCS*, 2018.

[58] Q. Lou, B. Feng, G. C. Fox, and L. Jiang, "Glyph: Fast and accurately training deep neural networks on encrypted data," in *NeurIPS*, 2020.

[59] R. Livni, S. Shalev-Shwartz, and O. Shamir, "On the computational efficiency of training neural networks," in *NIPS*, 2014.

[60] A. A. Badawi, B. Veeravalli, C. F. Mun, and K. M. M. Aung, "High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA," *TCHES*, no. 2, pp. 70–95, 2018.

[61] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Commun. ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[62] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *NIPS*, 2016.

[63] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. E. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *Usenix Security*, 2019.

[64] M. Ball, B. Carmer, T. Malkin, M. Rosulek, and N. Schimanski, "Garbled neural networks are practical," PPML, 2019, ia.cr/2019/338.

[65] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-party secure inference," in *CCS*, 2020.

[66] S. U. Hussain, M. Javaheripi, M. Samragh, and F. Koushanfar, "COINN: Crypto/ML codesign for oblivious inference via neural networks," in *CCS*, 2021.

[67] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *Usenix Security*, 2020.
[68] S. Wagh, D. Gupta, and N. Chandran, "SecureNN: 3-party secure computation for neural network training," *PoPETs*, no. 3, pp. 26–49, 2019.
[69] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *Proc. Priv. Enhancing Technol. (PoPETs)*, no. 1, 2021.
[70] Z. Ghodsi, N. K. Jha, B. Reagen, and S. Garg, "Circa: Stochastic ReLUs for private deep learning," in *NeurIPS*, 2021.
[71] I. Damgård, M. Geisler, and M. Krøigaard, "Efficient and secure comparison for on-line auctions," in *ACISP*, 2007.
[72] A. P. K. Dalskov, D. Escudero, and M. Keller, "Secure evaluation of quantized neural networks," *PoPETS*, no. 4, pp. 355–375, 2020.
[73] T. Veugen, "Improving the DGK comparison protocol," in *WIFS*, 2012.
[74] N. Agrawal, A. S. Shamsabadi, M. J. Kusner, and A. Gascón, "Quotient: Two-party secure neural network training and prediction," in *CCS*, 2019.
[75] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, "Flash: Fast and robust framework for privacy-preserving machine learning," *Proc. Priv. Enhancing Technol. (PoPETs)*, no. 2, pp. 459–480, 2020.
[76] A. Patra and A. Suresh, "Blaze: Blazing fast privacy-preserving machine learning," in *NDSS*, 2020.
[77] B. Knott, S. Venkataraman, A. Y. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "CrypTen: Secure multi-party computation meets machine learning," in *NeurIPS*, 2021, poster.
[78] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *JMLR*, vol. 18, 2017.
[79] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018.
[80] G. Yang, T. Zhang, P. Kirichenko, J. Bai, A. G. Wilson, and C. D. Sa, "SWALP : Stochastic weight averaging in low precision training," in *ICML*, 2019.
[81] H. Wong, J. Ma, D. Wong, L. Ng, and S. Chow, "Learning model with error - exposing the hidden model of BAYHENN," in *IJCAI*, 2020.
[82] Q. Zhang, C. Xin, and H. Wu, "GALA: Greedy computation for linear algebra in privacy-preserved neural networks," in *NDSS*, 2021.
[83] S. Li, K. Xue, B. Zhu, C. Ding, X. Gao, D. S. L. Wei, and T. Wan, "Falcon: A Fourier transform based approach for fast and secure convolutional neural network predictions," in *CVPR*, 2020.
[84] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: improved mixed-protocol secure two-party computation," in *Usenix Sec.*, 2021.
[85] A. P. K. Dalskov, D. Escudero, and M. Keller, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *Usenix Security*, 2021.
[86] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *AsiaCCS*, 2018.
[87] W. Du and M. J. Atallah, "Protocols for secure remote database access with approximate matching," in *E-Commerce Security & Privacy*, 2001.
[88] P. Mohassel and P. Rindal, "ABY^3: A mixed protocol framework for machine learning," in *CCS*, 2018.
[89] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "Swift: Super-fast and robust privacy-preserving machine learning," in *Usenix Sec.*, 2021.
[90] N. Attrapadung, K. Hamada, D. Ikarashi, R. Kikuchi, T. Matsuda, I. Mishina, H. Morita, and J. C. N. Schuldt, "Adam in private: Secure and fast training of deep neural networks with adaptive moment estimation," *PoPETs*, no. 4, 2022.
[91] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *ICLR*, 2018.
[92] D. Rathee, M. Rathee, R.-K.-K. Goli, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi, "SiRnn: A math library for secure RNN inference," in *S&P*, 2021.
[93] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[94] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *SSST@EMNLP*, 2014.
[95] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *Financial Cryptography and Data Security*, 2010.
[96] D. Heath and V. Kolesnikov, "One hot garbling," in *CCS*, 2021.
[97] S. Bian, W. Jiang, Q. Lu, Y. Shi, and T. Sato, "NASS: optimizing secure inference via neural architecture search," in *ECAI*, 2020.
[98] Q. Lou, Y. Shen, H. Jin, and L. Jiang, "SafeNet: A secure, accurate and fast neural network inference," in *ICLR*, 2021, poster.

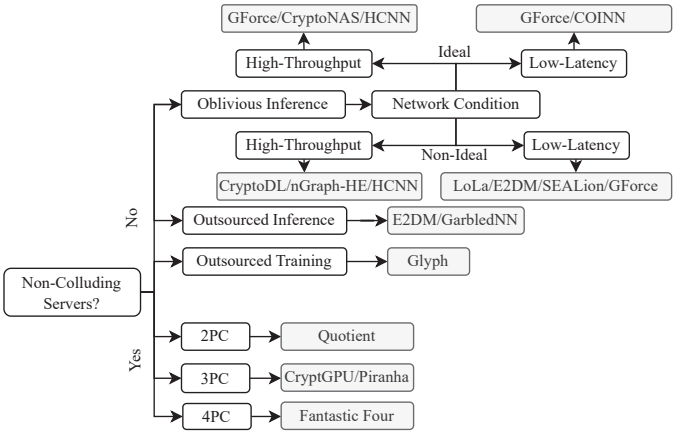

Fig. 7. The State of the Art with Different Features/Setups

[99] Z. Ghodsi, A. K. Veldanda, B. Reagen, and S. Garg, "CryptoNAS: Private inference on a ReLU budget," in *NeurIPS*, 2020.
[100] N. K. Jha, Z. Ghodsi, S. Garg, and B. Reagen, "DeepReDuce: ReLU reduction for fast private inference," in *ICML*, 2021.
[101] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *ICCV*, 2019.
[102] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data," in *Computing Frontiers*, 2019.
[103] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. E. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, "CHET: an optimizing compiler for fully-homomorphic neural-network inferencing," in *PLDI*, 2019.
[104] T. van Elsloo, G. Patrini, and H. Ivey-Law, "SEALion: a framework for neural network inference on encrypted data," arXiv:1904.12840, 2019.
[105] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "nGraph-HE2: A high-throughput framework for neural network inference on encrypted data," in *Enc. Comp. & App. Homo. Crypt. (WAHC)*, 2019.
[106] H. Chen, R. Cammarota, F. Valencia, and F. Regazzoni, "PlaidML-HE: Acceleration of deep learning kernels to compute on encrypted data," in *Intl' Conf. on Comp. Design (ICCD)*, 2019, invited paper.
[107] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure TensorFlow Inference," in *S&P*, 2020.
[108] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "EzPC: Programmable and efficient secure two-party computation for machine learning," in *EuroS&P*, 2019.
[109] J.-L. Watson, S. Wagh, and R. Popa, "Piranha: A GPU platform for secure computation," in *USENIX Security Symposium*, 2022.
[110] B. Hie, H. Cho, and B. Berger, "Realizing private and practical pharmacological collaboration," *Science*, vol. 362, no. 6412, pp. 347–350, 2018.
[111] S. S. M. Chow, J. Lee, and L. Subramanian, "Two-party computation model for privacy-preserving queries over distributed databases," in *NDSS*, 2009.
[112] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.
[113] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.
[114] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "Muse: Secure inference resilient to malicious clients," in *Usenix Security*, 2021.

APPENDIX I
THE STATE OF THE ART FOR VARIOUS SETUPS/FEATURES

This section shows which framework best fits a given scenario. Figure 7 serves as a decision tree guiding the choice.

A. Oblivious Inference without Non-colluding Servers

Figure 1 shows the accuracy and latency in a LAN setting of $>$1Gbps and $<$5ms communication latency.

TABLE III
FRAMEWORKS UNDER NON-COLLUDING ASSUMPTIONS (#: NUMBER OF SERVERS) OR WITH MALICIOUS SECURITY

Framework	#	Guarantee
SecureML [13], Quotient [74], ABY2.0 [84]	2	–
CrypTen [77], Piranha [109]	$\geq$2	–
QuantizedNN [72]	2/3	Abort
Chameleon [86], CrypTFlow [107]	3	–
CryptGPU [51]	3	–
ABY3 [88], SecureNN [68]	3	Abort
FalconN [69], AdamInPrivate [90]	3	Abort
Blaze [76]	3	Fair
Swift [89], Fantastic 4 [85]	3/4	G.O.D.
Flash [75]	4	G.O.D.
Trident [50]	4	Fair
GarbledNN [64], XONN [63]	–	Abort
Muse [114]	–	Client

1) Low-Latency Inference. COINN [66] and GForce [49] are on the *Pareto frontier*, *i.e.*, not superseded by any other in both accuracy and latency. Also, their latency is low ($0.01\sim$1s) for real-time usage. Note that the servers and the clients may need offline preparation before the queries arrive.

2) High-Throughput Inference. Online throughput shows the volume of queries answered in a given time. GForce still demonstrates high throughput, and COINN has its superiority in its high accuracy. HCNN [56] exploits parallelism to achieve the highest throughput but attains relatively low accuracy (80%).

3) Oblivious Inference under Nonideal Network Conditions. Oblivious inference may go under a WAN (vs. the "ideal" LAN) setting, *i.e.*, $<$320 Mbps and $>$40ms latency. If large NNs for complex tasks and accuracy are paramount, S/C frameworks still seem to be the best, *e.g.*, GForce [49] still performs an inference within one minute, as shown in Figure 3.

For smaller NNs and easier tasks, *e.g.*, MNIST, pure-HE frameworks perform better for their constant communication round. For throughput, CryptoDL [54], nGraph-HE [102], and HCNN [56] haVE the highest throughput and accuracy, as shown in Figure 5. For latency, LoLa [52], E2DM [57], and SEALion [104] are on the Pareto frontier in Figure 6.

4) Outsourced Training/Inference. Glyph [58] is the only option for outsourced training (but not private training). It also supports outsourced inference, but E2DM [57] has better performance for using LHE and does not leak the weights of the first few layers to the client like Glyph.

B. Do Non-colluding-based Frameworks Always Win?

For oblivious inference, Figure 1 shows that GForce has the lowest latency, outperforming non-colluding frameworks. CryptGPU strikes the best balance of throughput and accuracy. Only Glyph can support secure training without non-colluding servers. Note that it requires public datasets for transfer training.

C. Non-colluding MPC Settings

1) 2 Non-colluding Servers. SecureML and Quotient [74] are the only two. (ABY2.0 does not provide accuracy.) Quotient is reported to produce NNs with 6pp higher accuracy on MNIST and $>5\times/50\times$ training throughput in the LAN/WAN setting.

2) 3 Non-colluding Servers. CryptGPU [51] reports the highest accuracy, lower latency, and higher throughput than FalconN, SecureNN, and CrypTen in CIFAR-10. Yet, its throughput and latency are slightly worse than Piranha [109], as discussed in Section IX. Other 3PC solutions (*e.g.*, Blaze [76] and Swift [89]) only support inference and are tested on MNIST but not harder datasets. We remark that some of them can defend against malicious adversaries.

3) 4 Non-colluding Servers. Fantastic Four [85] reports that it has slightly better performance than Swift [89] and Flash [75]. How it [85] compared to Trident [50] is not clear because they have not compared themselves with each other but just emphasized their speed-up over ABY3 [88] as a reference point. Notably, they do not provide their training or inference accuracy over CIFAR-10.

D. Defending against Malicious Adversaries

A recent pursuit is to ensure security even when one server deviates from the protocol (but it still cannot collude with other servers) by fully exploiting non-colluding assumptions. In the basic malicious setting (ABY3 [88], SecureNN [68], and FalconN [69]), honest servers would *abort* if a corrupt server deviates from the protocol. For *fairness* (*Malicious Security* in Table IV), all honest parties can get the output (of internal protocols) if the corrupted can receive it (Blaze [76] and Trident [50]), even if the corrupted is malicious (*i.e.*, *guarantee of delivery* (G.O.D.) of Flash [75] and Swift [89]). Muse [114] features "abort security" against clients with higher efficiency than pure-GC approaches [63], [64].

TABLE IV
FRAMEWORK SUMMARY (▽: OBLIVIOUS INFERENCE, ▼: OUTSOURCED INFERENCE, □: OUTSOURCED TRAINING, ■: PRIVATE TRAINING)

	Framework	Basic Info.		Fixed-Point			Non-Linear			Optimization						Datasets				Crypto Tools			
		Reference	Year	Privacy Service	Trunc. & Wrap	Bitwidth	B/QNN	Poly. Approx. CMP		Num. Method	Offline/Online	HE SIMD	Dyna. Weights	GPU	Optimize Arch. Compiler	MNIST	CIFAR-10	CIFAR-100	ImageNet	GC/GMW	OT	SS	HE
Pure-HE	CryptoNets	[11]	16	▽	○	H	-	●	○	-	○	●	○	○	○	○	●	○	○	○	○	○	○	L
	BNormCrypt	[53]	17*	▽	○	H?	-	●	○	-	○	◐	○	○	○	○	●	○	○	○	○	○	○	L
	CryptoDL	[54]	17	▽	○	H?	-	●	○	-	○	◐	○	○	○	○	●	●	○	○	○	○	○	L
	Faster-Crypt	[55]	18*	▽	○	H	Q	●	○	-	○	◐	○	○	●	○	●	●	○	◐	○	○	○	L
	HCNN	[56]	21	▽	○	L	-	◐	○	-	○	◐	○	●	○	○	●	●	○	○	○	○	○	L
	E2DM	[57]	18	▼	○	H	-	◐	○	-	○	●	○	○	○	○	●	○	○	○	○	○	○	L
	nGraph-HE	[102]	19	▽	○	H	-	◐	○	-	○	◐	○	○	○	●	●	●	○	○	○	○	○	L
	nGraph-HE2	[105]	19	▽	○	H	-	○	○	-	○	◐	○	○	○	●	●	○	○	◐	○	○	○	L
	PlaidML-HE	[106]	19	▽	○	H?	-	◐	○	-	○	◐	○	●	○	●	○	○	○	○	○	○	○	L
	CHET	[103]	19	▽	○	L	-	◐	○	-	○	◐	○	○	○	●	●	●	○	○	○	○	○	L
	SEALion	[104]	19*	▽	○	H?	-	◐	○	-	○	◐	○	○	○	●	●	○	○	○	○	○	○	L
	LoLa	[52]	19	▽	○	H	-	◐	○	-	○	●	○	○	○	○	●	●	○	○	○	○	○	L
	FHE-DiNN	[44]	18	▽	○	L	B	○	●	-	○	○	○	○	○	○	●	○	○	○	○	○	○	T
	TAPAS	[45]	18	▽	○	L	B	○	●	-	○	○	○	○	○	○	●	○	○	○	○	○	○	T
	SHE	[46]	19	▽	○	M	Q	○	●	-	○	○	○	○	○	○	●	●	○	○	○	○	○	T
	Glyph	[58]	20	□	○	L	Q	○	◐	T	○	◐	○	○	○	○	●	○	○	○	○	○	○	L,T
Server/Client	DeepSecure	[33]	18	▽	○	L	-	○	◐	-	○	○	○	○	○	○	●	○	○	○	●	○	○	-
	XONN	[63]	19	▽	○	L	B	○	●	-	○	○	○	○	●	●	●	●	○	○	●	○	○	-
	GarbledNN	[64]	19*	▼	○	L	-	○	●	-	○	○	○	○	○	○	●	●	○	○	●	○	○	-
	MiniONN	[12]	17	▽	●	H	-	○	●	-	●	●	○	○	○	○	●	●	○	○	●	○	●	L
	EzPC	[108]	19	▽	○	H	-	○	◐	-	◐	○	○	○	○	●	○	◐	○	○	●	○	●	L
	Gazelle	[48]	18	▽	○	L	-	○	◐	-	◐	●	○	○	○	○	◐	◐	○	○	●	○	●	L
	FalconI	[83]	20	▽	○	L	-	○	◐	-	◐	●	○	○	○	○	●	●	○	○	●	○	●	L
	GALA	[82]	21	▽	○	L	-	○	◐	-	◐	●	○	○	○	○	●	◐	○	○	●	○	●	L
	Delphi	[67]	20	▽	◐	L	-	●	◐	-	●	◐	○	●	●	○	○	●	●	○	●	○	●	L
	GForce	[49]	21	▽	●	L	Q	○	●	-	●	◐	○	●	○	○	○	●	●	○	○	○	●	L
	Muse	[114]	21	▽	◐	L	-	◐	◐	-	●	◐	○	○	○	○	◐	◐	○	○	●	○	●	L
	Circa	[70]	21	▽	◐	L	-	○	●	-	◐	◐	○	○	○	○	○	○	●	○	●	○	●	L
	NASS	[97]	20	▽	○	L	-	○	◐	-	◐	◐	○	○	●	○	●	●	○	○	●	○	●	L
	CryptoNAS	[99]	20	▽	◐	L?	-	○	◐	-	◐	◐	○	○	●	○	○	●	●	○	●	○	●	L
	SafeNet	[98]	21	▽	◐	L	-	◐	◐	-	◐	◐	○	○	●	○	○	●	●	○	●	○	●	L
	DeepReDuce	[100]	21	▽	◐	L?	-	◐	◐	-	◐	◐	○	○	●	○	○	●	●	○	●	○	●	L
	CrypTFlow2	[65]	20	▽	●	L	-	○	●	-	◐	◐	○	○	○	○	○	●	○	◐	○	●	●	L
	SiRnn	[92]	21	▽	●	M	-	○	◐	T	○	○	○	○	○	○	○	○	○	○	○	●	●	-
	COINN	[66]	21	▽	●	L	Q	○	◐	-	●	○	○	○	●	○	○	●	○	●	●	●	●	-
Non-Colluding MPC	SecureML	[13]	17	■	●	H	-	○	●	-	●	○	○	○	○	○	●	○	○	○	●	●	●	L
	Quotient	[74]	19	■	●	L	B	○	◐	-	●	○	●	○	○	○	●	○	○	○	●	●	●	-
	ABY2.0	[84]	21	■	●	H	-	○	●	-	●	○	○	○	○	○	◐	○	○	○	●	●	●	-
	Chameleon	[86]	18	▼	○	L	-	○	◐	-	●	○	○	○	○	○	●	○	○	○	●	○	●	-
	CrypTFlow	[107]	20	▼	●	H	-	○	◐	-	○	○	○	○	○	●	◐	◐	○	●	●	○	●	-
	ABY3	[88]	18	■	●	H	-	○	●	-	●	○	○	○	○	○	●	○	○	○	●	○	●	-
	Flash	[75]	20	▼	●	H	-	○	●	-	●	○	○	○	○	○	●	○	○	○	○	○	●	-
	Blaze	[76]	20	▼	●	H	-	○	●	-	●	○	○	○	○	○	○	○	○	○	○	○	●	-
	Swift	[89]	21	▼	●	H	-	○	●	-	●	○	○	○	○	○	●	◐	○	○	○	○	●	-
	Trident	[50]	20	■	●	H	-	○	●	-	●	○	○	○	○	○	●	○	○	○	○	○	●	-
	Fantastic 4	[85]	21	■	●	H	-	○	●	-	○	○	○	○	○	○	●	○	○	◐	○	○	●	-
	QuantizedNN	[72]	20	▼	●	H	Q	○	◐	-	○	○	○	○	○	○	○	◐	○	●	○	○	●	-
	AdamInPrivate	[90]	22	■	●	L	Q	○	○	I	●	○	●	○	○	○	●	●	○	○	○	○	●	-
	SecureNN	[68]	19	■	●	H	-	○	●	-	●	○	●	○	○	○	●	○	○	○	○	○	●	-
	FalconN	[69]	21	■	●	L	-	○	●	I	●	○	●	○	○	○	●	●	○	○	○	○	●	-
	CrypTen	[77]	21	■	●	H	-	○	◐	I	◐	○	○	◐	○	●	◐	○	○	◐	●	●	●	-
	CryptGPU	[51]	21	■	◐	H	-	○	◐	I	○	○	●	●	○	○	●	●	○	○	●	○	●	-
	Piranha	[109]	22	■	◐	H	-	○	◐	-	◐	○	○	●	○	●	●	●	○	○	○	○	●	-

H: >32-bit, L: ≤32, M: mixed, ?: unspecified; I: iterative, T: table lookup; L: LHE, T: TFHE; * marks the earliest appearance of the preprint;
◐: adopter of existing techniques / w/o acc. or end-to-end inf. results; ●: original contributor / with performance & accuracy;

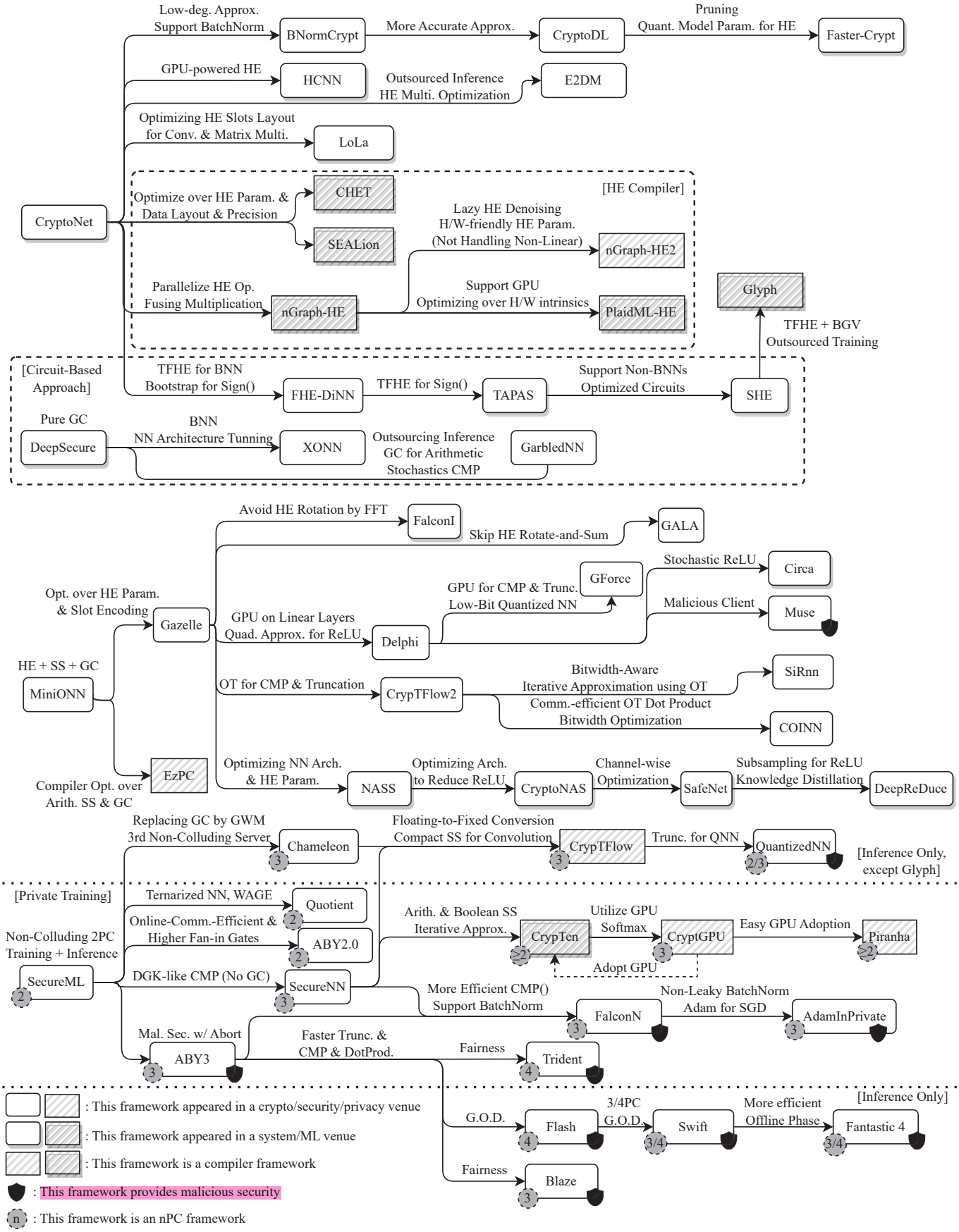

Fig. 8. The Genealogy of Frameworks (Names we came up with: BNormCrypt, GarbledNN, and FalconI/N for Inference/Training)