José Cabrero-Holgueras* and Sergio Pastrana

SoK: Privacy-Preserving Computation Techniques for Deep Learning

Abstract: Deep Learning (DL) is a powerful solution for complex problems in many disciplines such as finance, medical research, or social sciences. Due to the high computational cost of DL algorithms, data scientists often rely upon Machine Learning as a Service (MLaaS) to outsource the computation onto third-party servers. However, outsourcing the computation raises privacy concerns when dealing with sensitive information, e.g., health or financial records. Also, privacy regulations like the European GDPR limit the collection, distribution, and use of such sensitive data. Recent advances in privacy-preserving computation techniques (i.e., Homomorphic Encryption and Secure Multiparty Computation) have enabled DL training and inference over protected data. However, these techniques are still immature and difficult to deploy in practical scenarios. In this work, we review the evolution of the adaptation of privacy-preserving computation techniques onto DL, to understand the gap between research proposals and practical applications. We highlight the relative advantages and disadvantages, considering aspects such as efficiency shortcomings, reproducibility issues due to the lack of standard tools and programming interfaces, or lack of integration with DL frameworks commonly used by the data science community.

Keywords: Privacy Preserving Computation, Deep Learning, Homomorphic Encryption, Secure Multipary Computation.

DOI 10.2478/popets-2021-0064
Received 2021-02-28; revised 2021-06-15; accepted 2021-06-16.

1 Introduction

Deep Learning (DL) is the term given to a set of algorithms based on Artificial Neural Networks (ANNs). The term *deep* refers to the aggregation of multiple layers of neurons linked together, forming deeper networks (DNNs). These connected layers apply linear and non-linear transformations to the inputs, with the latest layer providing the output.

During training, a dataset is presented to a DL model that *learns* statistical properties and adapts its internal parameters (i.e., weights) to correctly mimic the distribution of the data (given the output). Then, in production or testing time, new inputs are presented to the model, which *infers* the output from the learned correlations. Due to its potential to solve complex problems and to deal with large datasets, DL is popular in multiple disciplines such as finances [1], cybersecurity [2] and medical research [3]. However, there are concerns regarding its security in adversarial environments. For example, since models enclose knowledge about the training dataset, they can be abused to leak private information [4].

Several available DL frameworks allow researchers and practitioners to train models using their data and use them internally in their local business processes [5–7]. In this scenario, privacy is protected by appropriate access control policies and perimetral security. However, DL solutions are costly and often require high computing and memory resources. In some scenarios where it is not cost-effective for data scientists to acquire hardware, they rely upon 'Machine Learning as a Service' (MLaaS). In MLaaS, clients send their data to a cloud infrastructure to build the models and later run the inference. Despite its benefits, this paradigm has multiple privacy implications since the data sent to third parties may get exposed. [3, 8, 9]. Another application where DL suffers from privacy issues is when multiple entities share their data for collaboratively training a DL model. A recurrent yet motivating example is on medical research, where various hospitals would benefit from each others' patient records to learn how to detect and treat rare diseases [10]. However, current technology is not well prepared for the application of DL on shared data while keeping its privacy [11]. Therefore, it is essential to understand the current knowledge, the failures, and the future research lines in the field of Privacy-Preserving Deep Learning (PPDL).

***Corresponding Author: José Cabrero-Holgueras:** CERN/Universidad Carlos III de Madrid, E-mail:jose.cabrero.holgueras@cern.ch
Sergio Pastrana: Universidad Carlos III de Madrid, E-mail: spastran@inf.uc3m.es

There are two main research lines related to PPDL. On the one hand, research focuses on attacks and countermeasures related to classical DL models. On the other hand, investigations looking for privacy-by-design DL architectures, where training and inference phases reveal neither the model nor the data. The former is related to an extensive research line on Adversarial Machine Learning, which aims at disclosing potential threats on classical ML algorithms and providing countermeasures [4, 12]. The latter is the research line covered in this paper. It requires the reformulation and adaptation of inner functions and algorithms to provide a full training or inference pipeline.

In MLaaS, the data is sent to a third party, non-necessarily trusted. In terms of privacy, this party becomes an adversary, so it is needed to protect the data, e.g., through encryption. However, the main strength of DL is precisely its ability to learn statistical properties from the training dataset and to infer these on test samples. Accordingly, the main challenge of PPDL is to meet two *a priori* opposed requirements. On the one hand, the DL process needs access to meaningful data to do either training or inference. On the other hand, a server running the DL process must not learn anything from the data. There are various approaches to achieve PPDL (we provide a high-level overview of them in §2). In this work, we focus on a subset of these, referred to as Privacy-Preserving Computation Techniques, or PPCT,[1] i.e., Homomorphic Encryption (HE) and Secure Multi-Party Computation (SMPC). These techniques allow to perform computations over protected data and thus are suitable for PPDL.

However, they incur a high computational and communication overhead to the already-demanding DL operations. PPCT for DL have attracted the attention of cryptographers and computer scientists in recent years, with an increasing number of proposals published at a fast rate. However, despite the high amount of papers addressing this topic, the technology is still immature, and few actual deployments are being used in privacy-preserving scenarios [11, 13].

In this paper, we study the past and present of the literature on PPCT applied to DL. To this end, we have studied more than 40 papers of impact from different fields (see §3 for a description of the scope and methodology used). Various works have surveyed this topic [8, 9, 11, 13–15]. However, our work differs from these in three main aspects. First, we provide a complete overview of the whole spectrum of privacy requirements, techniques, and settings for PPDL, and then we narrow down the analysis to a subset of these techniques, i.e., PPCT. Second, we provide a multi-disciplinary view of the problem, showing that advances in cryptography and computation are equally important to streamline this technology in efficient and accurate solutions. Finally, to the best of our knowledge, we are the first to analyze and discuss reproducibility and deployment issues of the different works, which, as we show, are key features to foster research and improve the state-of-the-art. We believe our analysis complements previous ones and makes a step-forward towards narrowing down the gap between academic works and deployments in the industry. Overall, the main contributions of this paper are:

1. We provide a basic but solid theoretical background that eases the understanding of current approaches and their limitations. Concretely, we describe some preliminaries on DL (Appendix A), and also both basic and advance topics related to SMPC and HE (§4).
2. We describe the main privacy requirements, techniques, and settings used for PPDL, and propose a generic pipeline that covers both centralized and distributed architectures and the different processing steps (§2). Based on the defined taxonomy, we outline the scope and selection methodology of the article (§3).
3. We provide a systematic description of the literature on PPCT, with a particular focus on the strengths and weaknesses of the different proposals, to analyze how these complement each other (§5).
4. We analyze the current limitations that prevent the deployment of existing solutions in real-world settings, mostly due to deficiencies related to efficiency and usability of the proposals, and discuss research lines that the community should address in the following years (§6)

This work highlights relevant challenges of PPCT for DL that require further attention from both the academy and industry. Also, we aim at easing comprehension of these topics to data scientists from other disciplines, not necessarily having the required mathematical background.

1 We use PPDL to refer to all the techniques aimed at Privacy-Preserving DL, and PPCT for the subset of cryptographic computation techniques.

2 Privacy Settings and Requirements in Deep Learning

PPDL considers a scenario where a set of one or more entities (clients) aim to privately perform DL training or inference. If any of these processes is outsourced to external parties (in the MLaaS paradigm), it will involve network communications with one or more servers. First, a set of *input training samples* is fed to the DL pipeline outputting a *model*. Then, in production time, the *model* is employed over a set of *inference samples*, producing *output predictions*.

In this paper, we focus on settings with the presence of external, non-trusted third parties (in §3, we further detail and justify the scope and methodology of our study). In this section, we describe various properties and settings related to the privacy of DL models and data. We first present the privacy goals and adversarial model. We then describe different techniques proposed to achieve such goals, and finally, a description of the architectures and data processing phases.

2.1 Privacy Goals

The main goal of PPDL is to enable training and inference while preserving the privacy of the associated entities (i.e., data providers, or clients, and service providers). It means that the processing cannot reveal additional information about their data. We identify the following privacy goals, depending on the source of information being protected:

Input Privacy aims to preserve data privacy during training or inference [16]. This requirement is needed when the data is sent to an external, non-trusted party (e.g., a cloud server) that performs the computation.

Output Privacy ensures the non-revelation of private information about the data from the products of the training (i.e., the model) or inference (i.e., the output predictions). This requirement is needed when the model is exposed and used by non-trusted parties, which might obtain information from it [17, 18]. We note that when the training process is outsourced to a non-trusted party, both input and output privacy are required (i.e., the data must be kept secret, and the model must be processed so that it does not reveal information about the input data).

Model Secrecy is the property ensuring the non-revelation of the attributes that define a model (i.e., architecture and weights).

Due to the complexity and effort required to train the models, these have a monetary value [13]. Also, gaining information about how a model works ease inference attacks against it [19]. Consequently, model secrecy is a desirable privacy feature. Such secrecy can refer to two components of the model. First, *Architectural Secrecy* keeps secret the organization of the layers and their internal hyper-parameters. Second, *Weight Secrecy* protects the values of the different weights given to the internal neurons after the training phase. The former is often related to intellectual property (e.g., cloud services not willing to expose how they process the data), and the latter with the output privacy guarantees (since the weights are adjusted to represent statistical properties of the training data).

2.2 Adversarial Model

Security guarantees of the protocols used in distributed systems, where multiple parties interact, often rely on the *ideal-real world simulation paradigm* [20–22]. This paradigm considers an ideal scenario where an ideal functionality (i.e., inviolable third-party) receives data from participants, computes a function in a centralized way, and sends back the result of the execution. In the ideal scenario, an adversary can only tamper with the inputs of corrupted parties. A distributed protocol is considered secure if, in the real world, the information exchanged between the different parties does not reveal more information to an adversary than what the ideal world reveals. According to this paradigm, two adversarial models are defined depending on their capabilities and goals:

Honest-but-curious (HBC) is a passive adversary that complies with the protocol and does not tamper with the data for malicious purposes. However, it tries to learn as much information from information exchanges.

Malicious is an active adversary with stronger capabilities. In addition to having the HBC adversary capabilities, it can tamper with the protocol (e.g., dynamically changing the inputs to the computation, not executing the process, or disconnecting at any point).

2.3 Privacy Techniques

Depending on privacy requirements, the adversarial model faced, and the specific settings, existing solutions for PPDL often use the following different techniques (either in isolation or combined in hybrid approaches):

Data Privacy techniques aim at reducing the amount of sensitive information that data carry. The goal is that content of data released does not reveal private information of the entities behind it. Two main techniques are used for this. First, *Differential Privacy (DP)* provides privacy guarantees for an individual of a larger population [23] (i.e., guaranteeing the privacy of the individual while allowing the calculation of population-wide statistics). DP mechanisms often rely on adding noise to the data reducing its expressiveness. Second, *Data Anonymization* aims at de-identification of data owners through generalization (i.e., removing identifying values from samples), omission (i.e., not including an individual in the dataset), and suppression (i.e., deleting complete identifying entries from the dataset) [24–26]. Data privacy techniques allow carrying out regular operations on the data, making them suitable for classical DL environments. However, they also suffer from utility reduction given the information loss from modifications.

Privacy Preserving Computation Techniques (PPCT) rely on cryptography to hide the information while allowing computation over it. It provides data secrecy (and thus privacy). There are two main approaches for this set of techniques (often used in combinations, as we discuss later), i.e., Secure Multiparty Computation (SMPC) and Homomorphic Encryption (HE). Different from the data privacy techniques, data is hidden not causing information loss. These techniques provide input privacy. However, while the amount of available information is not reduced, the available operations are limited. Another drawback is that these techniques suffer from important performance and usability challenges when they are used in PPDL (see §4).

Trusted Execution Environments (TEEs) are hardware components that allow for the encryption of a portion of a process memory [27–29] and keep the confidentiality and integrity of the data and operations loaded. Sometimes, hardware solutions such as TEE are used to conduct attestation of the executed code. Some approaches rely on TEEs for DL [30–32]. They are considered to preserve input privacy, though their security guarantees are based on hardware and are subject to side-channel attacks [33, 34]. For example, SLALOM relies on TEEs for simple private operations and executes complex computations in an external, not necessarily trusted GPU, using ZKP to attest their correct execution [32]. Also, Chiron proposes a virtual server with a limited instruction set running on top of a TEE so they can be attested [30].

Federated Learning allows to collaboratively train a model using local data from different entities without revealing it to the other parties [35]. The clients use their local data to train a local version of the model to compute the updates (i.e., gradients). Then these updates are sent back to a central server by sharing the resulting weights and parameters [36], which the central server aggregates onto a global model. By itself, this technique suffers from security issues since the generated model and gradients are shared, and they may be abused to breach privacy. Thus, it is often combined with other techniques to preserve input and output privacy, such as HE, SMPC [37] or Differential Privacy [36, 38]. We point out for further information to the works by Kairouz et al. [39], and Bonawitz et al. [40]

2.4 Architectures and Processing Steps

A PPDL pipeline generally involves 3 steps (see Figure 1 in Appendix B). In the *preprocessing* phase, the client and server transform the input data and the model respectively to evaluate the cryptographic protocol (e.g., linearization of activation functions for Homomorphic Encryption evaluation). In the *privacy-preserving processing* phase, the server receives the input data from the client and performs the actual evaluation of the circuit. Finally, the *post-processing* phase consists of either reconstruction of the data from the different pieces or the decryption of the message. Only the preprocessing phase requires modifying the functions used internally in DL. The other two stages merely apply the modified model on the privatized data according to the protocol. Thus, most contributions focus either on improving the adaptation of DL to cryptographic protocols or creating new protocols (or combinations of them) which reduce the changes needed.

The paradigm of MLaaS assumes a distributed protocol where different entities communicate in a network. Based on that, we consider two general architectures, depending on where and how the actual computation is done:

Centralized Architectures load the exigent processing on one party (i.e., a server with enough computational resources). As such, a single server obtains the necessary information from the client and performs most of the computation. Clients are not required to use high computational resources or frequent interaction with the server. In general, in a centralized model, the original model is only held on the server-side and is often present in solutions with Homomorphic Encryption. Centralized

architectures involve complex operations, but less communication.

Distributed Architectures allow several parties to distributively make processing on their data without sharing the actual data. These architectures split the processing among all the participants in a distributed fashion, without a central server holding the entire model. It is often done through Secure Multiparty Computation techniques and requires computing infrastructure on all the parties. Distributed approaches perform less and simpler operations at the cost of more communication and might require client interaction.

3 Scope and Methodology

In this work, the main goal is to understand the landscape of MLaaS in data-sensitive contexts through privacy-preserving cryptographic computation, i.e., when the data sent to third-parties is never decrypted for processing. Privacy-Preserving Computation Techniques (PPCTs) ensure both the privacy and secrecy of the input data. Furthermore, they relieve the client endpoint from heavy workloads. Finally, they can be deployed in collaborative settings, where different stakeholders contribute with their data with a common goal (e.g., various hospitals sharing privately medical information to investigate rare diseases). As a secondary goal, we explore the adaptability of current DL techniques to cryptographic PPCT constructions. PPCT are often combined with other techniques (as explained in §2.3). Only in these crossing points, we detail the use of all PPDL techniques. Accordingly, while Differential Privacy is a widespread technique in the field of PPDL [41–46], we only include proposals that intersect with PPCT (we refer to previous work for details on DP [47]). Also, Federated Learning (FL) is outside the scope of the paper, given its need for computing infrastructure and the reduced use of PPCT construction in the aggregation phase.

PPCT have two main challenges to address for their proper use with Deep Learning, i.e. efficiency and usability.

PPCT can compute a limited set of operations since they suffer from performance issues when dealing with complex computations. While they have been widely deployed in other scenarios, such as private data aggregation or statistics [48–50], their application for DL is not straightforward. We consider that a proposal makes an **Efficiency** improvement if it introduces modifications to previous protocols reducing their runtime on DL workloads.

The second weakness is related to the deployability of these techniques. Many frameworks and tools ease the access for data scientists (not necessarily experts in computing science) to complex DL [5–7]. However, it is complex to adapt these frameworks for the use of PPCT. In this regard, we consider improvements to **Usability** if the proposal meets one of the following two criteria. First, if it simplifies the adaptability of the solution to existing DL frameworks (i.e., by providing tools aimed at reducing the overall programming effort). Accordingly, our study includes works that propose Application Programming Interfaces (APIs), compilers, or relevant practical tools that help to implement and deploy the theoretical solutions into practical applications, and thus foster their usability. Second, when the proposal is accompanied by an open-source implementation. Besides allowing for further improvements on the proposal, the presence of open-source implementations allow for reproducibility of the results. Also, we check whether the papers provides links to open-source implementations, or if these are released apart (e.g., looking in the web page of the authors, or their GitHub repositories). Additionally, we visit such open-source repositories and analyze two properties: i) whether the code matches the theoretical claims (i.e., if it implements the security mechanisms and features described in the paper), and ii) the maintenance of the source code or its integration with other existing frameworks (i.e., if the original implementation has been updated after the initial release).

In summary, for each of the proposals using PPCT for DL, we study the following: (i) the problem addressed, i.e., training or inference, (ii) the architecture proposed, i.e., centralized, distributed, or hybrid, (iii) the privacy goals and adversarial model assumed, (iv) the particular techniques involved, i.e., SMPC, HE and/or others, and, (v) the issues considered regarding efficiency and usability.

To select relevant proposals, we conducted queries in various research repositories and databases, looking for specific keywords (e.g., *Privacy-Preserving*, *Deep Learning*, *Secure [Multiparty] Computation* or *Homomorphic Encryption*). Then, we select those with higher impact (in terms of the number of citations) and also those that published in top venues. We proceed to read their abstracts to check whether they fit in the scope of our study. This gave us an initial set of works, which we carefully analyzed. Then, we apply snowball sampling using the references from the papers in the initial set to

add further relevant works. We know this process has limitations, and we might have left out good research works, since not always quality means popularity. Despite this limitation, we believe our study successfully includes all the relevant works proposing PPCTs for DL.

4 Privacy-Preserving Computation Techniques

This section provides an overview of the techniques used to implement privacy-preserving computation on DL. There are two main approaches: i) to conduct operations over encrypted data (HE), and ii) to split the knowledge of the data among different parties (SMPC). We next describe the basic notions underpinning these techniques, and advances that have enabled their applications to DL.

4.1 Homomorphic Encryption (HE)

Homomorphic Encryption (HE) is a property of encryption schemes that permit operations over encrypted data, guaranteeing that no information is released with the operations and that the decryption of the ciphertext yields the correct result.

Partially Homomorphic Encryption (PHE) schemes allow for a single operation to be performed over the encrypted data. The first notion of HE comes from the RSA cryptosystem, where authors expose the ability of this cryptosystem to multiply numbers secretly [51]. The possibility of performing operations under a given encrypted data opened up a new field and, more advanced algorithms would appear over time, such as ElGamal [52] or Paillier [53] cryptosystems.

Somewhat Homomorphic Encryption (SHE) schemes allow the execution of two different operations, i.e., addition and multiplication, which permit reproducing the behavior of a NAND gate, which is functionally complete.

Learning with Errors (LWE) is the cryptographic problem that forms the basis for most HE schemes. It consists of hiding the secret message on a system of linear equations with noise [54]. Performing operations over those systems causes an increment of the noise. When the amount of noise grows over a certain threshold, the secret becomes indecipherable. Thus, SHE can only evaluate circuits of bounded depth (i.e., it cannot perform any arbitrary number of operations).

Fully Homomorphic Encryption (FHE) was first proposed by Gentry in 2009 [55], supposing a breakthrough for many applications. FHE allows performing an unbounded number of computations. Gentry's main contribution is the notion of bootstrapping, which consists of the homomorphic evaluation of the decryption circuit over a ciphertext, followed by a re-encryption (i.e., resetting the noise to zero). Despite its capabilities and improvements, the use of bootstrapping is still inefficient and suffers from a low runtime performance [56, 57].

Leveled Homomorphic Encryption (LHE) avoids bootstrapping by setting the depth of the circuit beforehand. It relies on performing relinearization and modulus reduction.

4.1.1 Advanced Constructions for DL

Based on the initial ideas proposed by Gentry, multiple encryption schemes have been created using different arithmetic types, like integer operations (e.g. BFV [58] and BGV [59]) or boolean operations (e.g. GSW [60] or TFHE [61]). Of particular interest for the context of DL are floating-point operations. At the time of this writing, we are only aware of one solution that supports this arithmetic, i.e., CKKS [62]. It is also noteworthy to mention the availability of multiple open-source libraries, such as Microsoft SEAL [63], IBM HElib [64], Palisade [65] or TFHE [66].

Multiple projects propose abstraction layers to reduce the complexity of HE for DL.

RAMPARTS [67] relies on Palisade [65] and provides an environment to develop applications (in the Julia language), simplifying the use of HE directives. It uses symbolic execution to automate the construction of optimized circuits and automatically selects the HE parameters and the best encoding for the values. It reduces the complexity and provides an abstraction layer for programmers without cryptographic background.

Armadillo [68] provides a toolchain that compiles C++ code into HE operations, and transforms the arithmetic to work on an FHE backend in modulo 2 (i.e., boolean arithmetic).

4.2 Secure Multiparty Computation

Secure Multiparty Computation (SMPC) is the term used to refer to the techniques that permit a set of n par-

ties to perform computations on input data from each party, without revealing it to the other parties, and to output a shared, common result.

Multiple constructions support SMPC. Each of these has particular requirements and benefits, and thus they are usually combined. We refer to the recent work by Lindell [20] for a detailed description of SMPC.

Oblivious Transfer (OT) is a 2-party cryptographic protocol allowing a receiver to request k out of n pieces from a sender. The protocol ensures that the sender learns nothing about the sent pieces of information. The receiver learns nothing about the pieces of information that he does not receive [69]. This protocol can be extended to apply boolean SMPC [70]. Additionally, it is used as a base for secure data exchange in other protocols.

Yao's Garbled Circuit (YGC) is a 2-party secure computation cryptographic protocol for boolean circuits [71] in presence of HBC adversaries. It allows two parties to perform a computation on their private inputs x, y without neither knowing each others' input, nor the circuit.

In the basic protocol, a garbler (G) owns inputs x and the circuit $\mathcal{C}$, then it garbles the circuit $g(\mathcal{C}) = \mathcal{C}'$ and sends $\mathcal{C}'$ and the garbling of $g(x)$ to the evaluator (E). The inputs of E (i.e., $g(y)$) are sent to the evaluator using OT. The evaluator then blindly follows the protocol based on $g(\mathcal{C})$, the garbling x, and its input (i.e.,y). Several optimizations followed the initial proposal, including the point-and-permute optimization [72], the half gates [73], the free XOR gates [74, 75] or garbled row reduction [76, 77]. To a great extent, the success of GC for DL Inference involves using these optimization techniques.

Zero-Knowledge Proofs (ZKP) are a cryptographic constructions in which a *prover* P verifies that a statement x is part of a language L to a *verifier* V, satisfying completeness (i.e., if $x \in L$, V cannot reject the statement), soundness (i.e., if $x \notin L$, then V only accepts it with 50% probability) and zero-knowledge (i.e, V learns nothing from the statement rather than the commitment and the truth of it) [78]. The probability of accepting a false ZKP can be reduced from 50% by repeating the procedure multiple times. In the scope of private computation techniques, ZKPs are commonly used to protect against malicious adversaries, by requiring the different parties to prove the correct execution of operations.

Secret Sharing (SS) is a cryptographic protocol that permits creating n different 'parts' out of some secret information x, which can only be rebuilt into the original if at least k parties agree. SS is one of the bases to create SMPC protocols with more than two parties. More formally, a (k, n)-secret sharing scheme consists of a pair of algorithms. First, $Share(x)$ produces a tuple of n different shares $(s_1, s_2, ..., s_n)$. Then, $Reconstruction(s_{i1}, s_{i2}, ..., s_{ik})$ computes and produces the secret x out of k shares. Next, we describe three main protocols used for SS.

- **Shamir Secret Sharing** is based on the Lagrange Interpolation, which states that a polynomial $P(x)$ of degree n can be built from $n+1$ points [79]. The sharing consists of generating a polynomial whose independent term is the secret to be shared. Shamir secret sharing is efficient since it does not need any strong preprocessing. Moreover, it is homomorphic for addition and multiplication. Its main inconvenience relies on the fact that the multiplication of two degree-t shares generates a share of degree $2t+1$. Therefore, it requires degree reduction after each multiplication [80], which requires communications, thus incurring an overhead.
- **Additive Secret Sharing** is based on the idea that a given secret x can be decomposed in the sum of n random numbers [81]. The process for generating the share consists of selecting $n-1$ random numbers and computing the n-th share as the sum of the rest. While additive secret sharing is only homomorphic on the addition, Beaver Multiplication Triplets [82] extend it to perform multiplication. This approach is more efficient for computation because it moves the computational delay onto a preprocessing phase (i.e., the multiplication can be performed offline).
- **Verifiable Secret Sharing** schemes use homomorphic operations considering a malicious adversary [83–86]. The main changes to the previous approaches are the sharing of commitments to ensure the order does not interfere; the verification of the correctness of the computations with zero knowledge proofs [87]; the use of agreement schemes [88] and distributed coin-flipping protocols [89].

4.2.1 Advanced Constructions for DL

The basic techniques form the basis for more advanced protocols in terms of security, performance, versatility, and usability. These protocols are nowadays at the core of many of the proposals for PPDL. SPDZ [90] is an SMPC protocol for n-parties secure against the corruption of $n-1$ parties, which highly improves the security

of previous approaches. It relies on a computationally expensive preprocessing phase that reduces the computation of the subsequent processing phase. It combines the following primitives: i) additive secret sharing and beaver multiplication triplets for the computation, ii) SHE for data encryption, iii) ZKP to guarantee the correctness of the information, and iv) commitments to avoid malicious inputs.

In MASCOT [91], authors introduce modifications and remove the complexity from the preprocessing phase, making the processing phase a bit slower than SPDZ. According to their results, the secure protocol is six times slower than its non-secure counterpart. Additionally, Oblivious Transfer can be instantiated using lattice-based primitives, which achieves post-quantum security. The main drawback in MASCOT is the communication delay incurred by the use of OT. Depending on specific settings and the network infrastructure, this delay may be higher or lower. Overdrive [92] improves MASCOT [91] by introducing the use of Beaver Triplet distribution with HE and distributed decryption. Since the most expensive part of the protocol is the execution of ZKP, the approach is to optimize the Schnorr-like ZKP into a more efficient protocol [93].

Other works have attempted to speed up the computation of these protocols. TinyGarble [94] presents a methodology optimizing multiple aspects of Garble Circuits and defining an option for them to be executed on a MIPS I processor instruction set. The optimization permits scaling the amount of computation made.

4.3 Hybrid Techniques

Most PPCTs have a limited instruction set which is relatively efficient to operate in a specific arithmetic domain. However, many problems might require to operate on different arithmetic domains, often forcing to partially approximate the problem to specific arithmetic types. These approximations cause inefficiencies in terms of performance, precision, and flexibility. For example, in the case of DL, linear functions are computed efficiently with floating-point arithmetic, whereas non-linear activation functions require boolean arithmetic. To avoid the use of approximations, some authors have proposed what we define as *Hybrid Techniques*, also referred to as *share/ciphertext conversion protocol* [95]. These techniques permits switching from one PPCT algorithm to a different one, thus adapting to use the required arithmetic type while preserving the privacy of the construction. Hybrid techniques effectively improve the flexibility of the solutions, and preserve the efficiency and accuracy of the computation since the internal function are not approximated.

Hybrid techniques might combine different base cryptographic protocols from a single PPCT, i.e., HE or SMPC, and also propose conversions from one to the other [96]. As we analyze in §5.2.3, due to the arithmetic variety of internal functions applied in DL, various proposals make use of such hybrid techniques.

For HE, CHIMERA [97] presents a framework that allows for switching between three main HE schemes without the need for decryption. Concretely, it proposes using BFV [58], HEAAN (CKKS [62]) and TFHE [61] for integer, floating-point and boolean arithmetic respectively. It has a strong potential for DL since linear functions can be executed in floating-point arithmetic, whereas activation functions rely on boolean arithmetic, thus not needing an approximation.

Similar to HE, SMPC suffers from using a single arithmetic type. ABY [98] (Arithmetic-Boolean and Yao's sharing) gives the programmer access to three different protected data types, namely arithmetic secret shared, boolean secret shared, and Yao's GC. The most important contribution is that they provide efficient cryptographic bridges between the different constructions. ABY is a key contribution that shows how the provision of tools fosters research. Indeed, it serves as a basis for various proposals for PPDL [99, 100]. Even though ABY offers a higher-level abstraction due to the provision of data types, it remains a complex low-level notation, which requires detailed knowledge, and whose optimal use is left to programmers. EzPC [101] partially solves this problem by adding a new layer of abstraction and generating a two-party computation protocol from the high-level description of the language. This layer hides the cryptographic details from the user and selects the parameters automatically. It is implemented in the form of a cross compiler that translates C++ into 2-party secure code by using ABY below it.

5 State of the Art

The interest in DL techniques for solving problems in privacy-sensitive scenarios is growing. However, DL was not designed considering privacy and security goals. While the research area is relatively new, there are multiple precedents on private data analytics and Machine Learning (§5.1) which are the basis of proposals for PPDL inference (§5.2) and training (§5.3).

5.1 Preceding Approaches

The application of PPCT for data mining and machine learning solutions has been addressed before the growth in popularity of DL [12, 102]. Indeed, the idea of secretly evaluating a neural network was first proposed in 2008 by Sadeghi and Schneider [48]. Authors propose a distributed 2-party computation paradigm where the security relies on the secrets each of the parties stores. The authors use SMPC based on OT transforming the NN using Generalized Universal Circuit (i.e., any circuit can be simulated in Boolean arithmetic). In the following paper, TASTY [103] presents a distributed 2-party paradigm proposal where the aim is combining the Paillier PHE cryptosystem with other structures to achieve the execution of other operations such as multiplication. Additionally, the authors present a compiler that simplifies the translation of code written in the Keras framework [5] to the TASTY protocol. It constitutes one of the first attempts to bring together the theoretical concepts with actual deployments. In ML Confidential [104], the authors propose an approach to use linear regression models with LHE. This paper is one of the first attempts that propose a privacy-preserving machine learning solution using encryption and covers various architectural issues and problems such as the polynomial approximations or the fixed-depth of circuits. In CodedPrivateML [105], the authors train machine learning models (i.e., linear and logistic regression), using Shamir's secret sharing and speed it up with the Lagrange Coding. This paper proposes a solution based on cloud computing by distributing the workload to train the algorithms.

Collaborative data analytics have been an area of research for PPCT. The work by Ohrimenko et al. [49] enable different parties to interact through Trusted Execution Environments (TEE) and oblivious access to data structures. In this case, the task to be carried out is known by all the parties, but the access patterns are hidden so no data needs to be released.

These works are examples of the ideas that form the basis for posterior contributions in the area of DL inference and training, which are analyzed in the following sections.

5.2 Privacy Preserving Deep Learning Inference

In this section, we analyze proposals for protecting the privacy of the data sent for inference. Thus, these works do not provide mechanisms that allow for training over privacy-protected data. There are three main approaches, i.e., using centralized (HE), distributed (SMPC), or hybrid architectures. We also analyze proposals that aim to ease the abstraction of these techniques for existing DL frameworks, through programming interfaces and tools (e.g., compilers).

5.2.1 PPDL Inference in Centralized Architectures with Homomorphic Encryption

As mentioned before, the emergence of FHE schemes supposed a step forward for PPDL. Due to the computational complexity incurred by these schemes, the first approaches using FHE are designed for a centralized environment and make use of high-performance hardware.

One of the first solutions that used LHE with DL, named Cryptonets, was proposed in 2017 [106, 107]. It allows training over non-encrypted data, and inference on encrypted data, in a centralized architecture. The main contribution is the conversion of the internal operations and structures of the DL model to use only addition and multiplication. This way, they can be applied to encrypted data using LHE. Concretely, the authors proposed a modification of the ReLU activation function using a square function and the substitution of the Max Pooling layer by an average pooling layer. Faster Cryptonets [108] improves the original proposal using a *quantization* scheme, i.e., a pruning technique for neural networks, and optimal approximations. These optimizations reduce the number of operations performed and thus the width of the circuits.

One of the weaknesses of Cryptonets [107] and its subsequent optimization in Faster Cryptonets [108] is that the activation functions are imprecise. It is due to the ReLU activation functions being a paramount process for the success of DL. Thus, in CryptoDL [109], authors propose different approaches to approximate the ReLU activation using low-degree polynomials. Their solution relies on using the Chebyshev Polynomial approximation of the integral of the sigmoid function.

TAPAS proposes optimizations in the domain of Boolean arithmetic [110]. Authors propose the use of Binary Neural Networks (BNN) and the TFHE library [66]. TAPAS allows to first make the matrix multiplication faster-adapting multiplication functions to the XNOR gate, and then to count the number of 1's in the result for the actual summation. TAPAS BNNs achieve a reasonable speedup for DNN tasks but work less efficiently on CNN tasks. Interestingly, the authors released

this tool open-source, which uses the general-purpose HE evaluation framework SHEEP [111].

In FHE-DiNN [112], the contribution shifts the focus towards using discretized neural networks (i.e., using integer weights) based on boolean arithmetics on top of TFHE. The use of these primitives permits reducing the size of the computation and an increase in performance. Given the native use of boolean arithmetics, they claim a performance improvement over Cryptonets. However, due to the changes performed to the neural networks after training, there is a reduction in the accuracy of the resulting model.

Finally, E2DM [113] is a framework that relies on an encoding that accelerates matrix computations. Due to the improvements in these operations, the authors show an acceleration of DL inference using a specific encrypted CNN model (not trained from encrypted data) to classify images from the well-known MNIST dataset.

5.2.2 PPDL Inference in Distributed Architectures with Secure Multiparty Computation

Due to the limitations of HE in terms of performance and scalability, various works propose the use of distributed architectures (i.e., SMPC) to enable DL inference. These works give continuity to the ideas proposed in TASTY [103] (described in §5.1) for SMPC.

MiniONN [114] is the first proposal using boolean arithmetics for a distributed architecture. It allows private inference (both model and data) through SMPC based on Oblivious Transfer. According to the proposed protocol, any DNN can be transformed into MiniONN. The main disadvantage of this work is the inherent communication delay due to the use of OT for SMPC.

In DeepSecure [115], the authors propose a variation of the preprocessing phase of Yao's GC for the optimization of the DL functions. Since most of the overhead relies on the communication delay and the preprocessing phase of garbled circuits, authors propose an optimized version of DL models reducing the information exchanged. Still, the use of GC implies *regarbling* all the circuit for each input, which entails considerable overhead.

XONN [116] also applies boolean arithmetic, and similar to TAPAS [110], they improve the efficiency for BNNs Inference using a Yao's GC. However, XONN proposes optimized circuits of XNOR gates that permit faster computations of matrix multiplications. Additionally, the authors introduce a pruning algorithm for the neural network to generate a compressed representation for inference and improve the garbling to set a constant number of communication rounds for architectures of up to 21-layers. Finally, they provide a high-level API that permits translating a Python Keras [5] model into their proposed representation.

Dalskov et al. [117] propose a quantization scheme that enhances transmission and speeds up the processing of DNN, considering malicious adversaries. The proposal is evaluated using 16-bit and 8-bit floating-point numbers with a considerable reduction of communications.

5.2.3 PPDL Inference in Hybrid Architectures

In the previous section, we observed how HE is a good technique for computing linear and polynomial operations. However, the conversion of non-linear functions often comes at the cost of precision and efficiency. On the other hand, SMPC scales poorly to many parties due to communication delays. Moreover, the adaptation of a DL model to a single arithmetic type (e.g., integer, fixed-point, floating-point, or boolean) restricts its operations and deteriorates its accuracy. Accordingly, hybrid techniques, where both approaches are combined and that support operations in different arithmetic types, are a research direction that has been also explored.

Chameleon [100] leverages ABY [98] to switch between different arithmetic types during computation. Concretely, it uses fixed-point arithmetic for the linear activation functions and boolean (based on Yao's GC and GMW [21, 118]) for the non-linear ones. This work profits from the contributions from both Cryptonets [107] and CryptoDL [109] due to the importance of precision of the activation functions.

GAZELLE [96] proposes an SMPC protocol based on a hybrid approach for DL inference considering an HBC adversary. On the one hand, they use Packed Additively Homomorphic Encryption (PAHE) for the linear activation functions. PAHE speeds up vector and matrix computations and is designed to avoid bootstrapping. On the other hand, they use Yao's GC for non-linear functions (e.g., ReLU or MaxPool). Due to the need to combine PAHE and GC, they create a cryptographic bridge for switching between these two structures. In this way, they present a speedup of 30x compared to Chameleon [100] and MiniONN [114].

Finally, DELPHI [119] proposes improvements to GAZELLE based on combining GC and quadratic polynomials for the activation functions. The key aspect is to generate several models derived from the origi-

nal using different activation functions. Depending on particular goals, these functions can be substituted by either GC (i.e., accurate approach) or quadratic polynomials (i.e., efficient approach). Since either performance or accuracy is degraded, DELPHI uses a planner to calculate an optimal trade-off that minimizes the loss. They also claim that the protocol permits reducing the amount of information exchanged by 90% with respect to GAZELLE.

5.2.4 Programming Interfaces for PPDL Inference

While it is essential to improve the efficiency of PPDL techniques, it is also fundamental to ease the integration of this technology into existing frameworks. In general, the techniques proposed in the literature require expertise and mathematical background to be appropriately deployed. Thus, while privacy and efficiency are two cornerstones for PPDL deployment, usability and applicability of the techniques are also important and desirable features. In this section, we analyze proposals aimed at bridging the gap between the theoretical proposals and their actual implementations and deployment. We cover two main types of contributions: works that propose compilers from high-level programming languages into privacy-preserving protocols (e.g. by means of libraries), and works that automatically transform outputs from common DL frameworks (e.g., Tensorflow, Keras, PyTorch) to a PPDL protocol.

CHET [120] is a compiler aimed at creating homomorphically encrypted code from a high-level representation. However, it is the first to provide DL primitives. It performs a code translation process efficiently by generating a homomorphically executable code for DNN inference. Nevertheless, it does not use common frameworks and requires specifically designed neural networks. EVA [121] elaborates a parameter selector on top of CHET which processes the circuit to be executed.

Intel nGraph HE Tranformer [122–124] implements a compiler and runtime environment for inference with HE over Tensorflow. A back-end process obtains the graph from Tensorflow, generates encrypted data, and executes the homomorphically encrypted inference. The first version [122] relies on Intel nGraph [125] and implements high-level operations with HE to perform PPDL inference. Additionally, it implements optimizations for ciphertext packing that permit executing single instruction over multiple data (SIMD) operations. However, the activation functions only allowed for non-linear approximations with a low degree. Thus, the second version of the compiler [123], was designed to avoid executing non-linear operations on ciphertexts, by distributing the computations between the server and the clients. Concretely, for those non-linear functions, the information is sent back to the client, who decrypts it, executes the activation, and then sends back the results to the server. Additionally, they further optimize the relinearization operation, reducing the number of executions.

MP2ML [124] uses Yao's GC to support the computation of the activation functions (i.e., in a similar way as GAZELLE [96]). Contrary to Intel nGraph HE, the computation of the activation through GC guarantees the weights secrecy on the model.

PlaidML-HE [126] proposes a new HE compiler for PlaidML (a library for speeding Machine Learning workloads using heterogeneous hardware back-ends). This way, it provides an intermediate abstraction layer agnostic to the machine learning framework being used. TinyGarble2 [127] presents an evolution of TinyGarble [94] that permits having an efficient representation of GCs enabling the execution of Neural Networks in shorter times. For that, they implement an interface to C++ and a DL library containing primitives to build CNNs for inference.

The PySyft framework allows inference and training [128]. It combines PyTorch and SPDZ [90]. Tensors built can be shared among the parties executing the computation and make it seamless. Similarly, TFEncrypted is an open-source library that applies changes to Tensorflow to make it usable with SecureNN and ABY3 as compiler and runtime. We note that, by June'21, PySyft and TFEncrypted have not fully working implementations of the proposed PPCT backends.

Finally, other proposals focus on the usability of SMPC. CrypTFlow is an end-to-end approach that permits translating Tensorflow code to different SMPC protocols [129]. CrypTFlow has three components: i) Athos is an end-to-end compiler from TensorFlow to a variety of semi-honest MPC protocols, ii) Porthos is an improved semi-honest 3-party protocol that is allegedly faster than State of the Art tools, especially for the convolution operation iii) Aramis is a tool that converts any semi-honest MPC protocol into an MPC protocol that provides malicious security based on the use of TEEs.

5.3 Privacy Preserving Deep Learning Training

All the approaches covering DL inference assume the existence of a pre-trained model and do not cover back-

propagation. In various scenarios, we want to outsource the training to a third-party server. For example, if a single entity lacks the necessary hardware. Or, for collaborative learning, when various entities share sensitive data for DL training keeping *input privacy*.

5.3.1 HE for PPDL Training

The first constructions to cover training extend PPDL inference to implement a two-party scenario where the backpropagation is computed with the help of the client.

Hesamifard et al. [130] propose a continuation to CryptoDL [109] for training with LHE. Given the lack of efficient bootstrapping, they opt for sending back the data to the client who re-encrypts it and sends it back to the server for computation. In addition to the feasibility they test their approach on multiple datasets. Nandakumar et al. [131] propose to make the full training on the server side. For that, they adapt the complete DL pipeline including the loss function and stochastic gradient descent function. They show a set of optimizations that thanks to them permit performing an encrypted training in 40 minutes.

5.3.2 SMPC for PPDL Training

Similarly, most approaches relying on SMPC perform training among two parties nodes. Sometimes, these protocols introduce another party which acts as randomness provider.

One of the first attempts to perform PPDL training was proposed in 2017 by Chase et al. [132]. In their work, authors combine Differential Privacy for data protection with SMPC through additive secret sharing for collaborative privacy computation. For their use case, the authors show a detailed comparison between the accuracy loss and the privacy budget. DP shows the problem of having to reduce the privacy budget (i.e.,$\epsilon > 8$) for achieving good accuracy.

In a more complex approach, SecureML firstly proposed a semi-honest 2-party distributed approach to train linear regression, logistic regression, and neural networks [50]. The protocol consists of two phases: an offline preprocessing phase and the private computation online phase. The offline phase is expensive since it requires generating Beaver Triplets for the rest of the computation. Thus, SecureML optimizes this generation using vectorized LHE and OT. Then, the online phase uses Additive Secret Sharing, Beaver Triplets for the shared computation, and Yao's GC for bit-level arithmetics.

QUOTIENT [133] approaches DL training through two-party OT SMPC. For that, they introduce a novel quantization scheme for the values. Additionally, they adapt novel techniques such as layer normalization and adaptive gradient methods to improve training efficiency.

FLASH [134] is a 4-Party Computation with a malicious adversary framework. It extends ASTRA [135], a preceding version focused on ML. It includes Guaranteed Output Delivery, which ensures that the protocol finishes even if the parties are dishonest. For that, they create a bi-convey primitive that evolves from a commitment scheme and permits two honest parties to discover another party who is not being honest. Additionally, it includes mechanisms in the secret sharing and circuit computation to ensure honesty. This protocol is tested and compared with ABY3 and SecureML, claiming efficiency improvements for the training phase.

5.3.3 Hybrid Techniques for PPDL Training

While some SMPC techniques are designed for 3 and more parties, their adaption to DL is not easy. With hybrid techniques, we observe protocols adapting better to DL training.

As we described in §4.3, ABY and EzPC provided abstract layers to allow for standard SPMC computation. Likely, ABY3 [99] focuses on DL training and proposes a similar protocol allowing for 3-party computation. It considers both HBC and a malicious adversary. The most relevant contribution of this work is a compiler solving the need for efficient computation and ciphertext packing; and providing arithmetic switches for different representations of information (GC, boolean, and arithmetic secret sharing). While ABY3 releases the open-source implementation it only provides the semi-honest adversary version of the protocol.

SecureNN [95] improved previous works allowing 3 independent servers to perform privacy-preserving training and prediction. The protocol considers m parties who want to share their data to train a model using 3 servers. Firstly, they use Secret Sharing from the m parties onto the n servers. Then, two of the servers make use of additive secret sharing which is collaboratively performed with the third server. The third server provides beaver triplets for multiplication and participates in the operation in a boolean sharing of some bits of the computation. They also propose modifications and

optimizations for ReLU and MaxPool, reducing up to 8 times the computation overhead.

Finally, POSEIDON [37] proposes a framework for training DNNs based on federated learning and multi-key FHE. Concretely, the use of multi-key FHE permits generating multiple private keys tied to a single public key. This way, each party performs encryption on its share, while decryption requires an agreement between all the parties. Multi-key FHE uses SMPC protocols for key-generation, bootstrapping, and key-switching. POSEIDON modifies federated learning so that training is fully executed under HE under multiple servers and there is a hierarchical gradient aggregation under HE.

6 Current Challenges and Research Directions

The analysis of the current state-of-the-art of PPCT for DL helps to outline the current challenges that need to be addressed by the community. Table 1 summarizes the analysis of the proposals with respect to the attributes defined in §3. We group works depending on whether they address DL training or inference, or if they propose an API or tool to ease the use of the PPDL techniques.We first provide an analysis of these proposals and then summarize some interesting remarks and takeaways from our study. For reference, the links to the open-source repositories are provided in Appendix C.

6.1 Analysis

There are various **problems addressed and challenges** in which PPCT have been used. Regarding the works that address *privacy-preserving inference*, we observe that those using HE uniquely in a centralized architecture are progressively losing interest (the latest proposal is from 2018). They often lose accuracy due to the need for approximations of non-linear functions and are computationally demanding. Other contributions rely on SMPC through distributed architectures, trading off the computational performance and communication overhead. Both HE and SMPC proposals need to modify DL structures to match the corresponding cryptographic protocols, which affects accuracy and hinders efficiency with existing frameworks. Thus, we observe an evolution towards hybrid techniques that adapt cryptographic protocols to the DL structures. As an example, GAZELLE [96] combines efficient constructions for linear computations (e.g., efficient HE schemes) and boolean SMPC for non-linear functions.

We observe that proposals that rely on HE and Hybrid Techniques often provide an open-source implementation (i.e., 5 out of 7 and 2 out of 3 open-source works, respectively), which allows for its reproducibility. However, most of the works using SMPC focus on efficiency improvements, leaving aside their reproducibility and usability. Indeed, only the proposal by Dalskov et al. [117] provides an open-source implementation, which has been integrated into the MP-SPDZ framework [137]. Also, XONN [116] implements an API and compiler to translate from high-level descriptions of neural networks to low-level code, though it is not released open-source.

While works intended for PPDL inference are progressively improving performance and usability, solutions covering **PPDL training are still immature** in these terms. Similar to PPDL inference, the major bottleneck for PPDL training is its high computational overhead, preventing its application to complex, real-world use cases. Indeed, we observe that, to lower the computation, often the proposed test neural network architectures are of reduced complexity (i.e., low number of layers and dimensions). Furthermore, PPDL training in distributed settings requires higher network communications. The comparison of the performance over Wide Area Networks (WAN), where the network bandwidth is lower than Local Area Networks, shows improvable delays, as shown in SecureNN [95]. Finally, most PPDL inference contributions provide specific implementation details of the different DL components (e.g., linear layers and polynomial approximations strategy). In training, we have observed some proposals omit details such as loss function or non-linear derivative approximation. This, together with the fact only 3 out of 10 contributions share the source code, hinders their reproducibility and usability.

An active area of research is the provision of **programming interfaces and compilers** that allow for smooth integration of existing frameworks and DL projects using PPCT. For end-users to become familiar with these technologies, it is essential to enable tools in native languages. In the data science domain, interfaces adapted to DL frameworks such as Tensorflow [6], Keras [5] or Pytorch [7] are fundamental. We highlight Intel nGraph HE, where proposals from various papers are integrated and maintained in an open-source tool [122–124], which provides data scientists with seamless solutions. Additionally, we highlight proposals that elaborate conversion routines (e.g., TASTY [103] or

Objective	Privacy Technique	Name	Reference	Year	Number of Parties	Input Privacy	Output Privacy	Architectural Secrecy	Weights Secrecy	Adv.		Arith.			SMPC				HE			Trusted Execution Environments	Differential Privacy	Efficiency Improvement	Usability Improvement	Open Source Implementation	Paper-Code Matching	Code Maintenance	Tensorflow/Keras/PyTorch
										Honest-But-Curious	Malicious	Integer	Boolean	Floating Point	Oblivious Transfer	Yao's Garbled Circuits	Additive SS & Beaver Triplets	Shamir's SS	Zero-Knowledge Proofs	Commitment Schemes	Partially Homomorphic	Levelled Homomorphic	Fully-Homomorphic							
DL Inference	HE	TASTY†	[103]	2010	2	✓	×	✓	✓	◐		●									●				▲	▲	✓	✓	×	✓
		Cryptonets	[106, 107]	2014	2	✓	–	✓	✓	◐		◇		◆								●			▲	▲	✓	✓	×	×
		CryptoDL	[109]	2017	2	✓	–	✓	✓	◐		◇		◆								●			▲	–	×	?	?	?
		TAPAS	[110]	2018	2	✓	–	✓	✓	◐			●										●		▲	▲	✓	✓	×	×
		Faster Cryptonets	[108]	2018	2	✓	✓	✓	✓	◐		◇		◆									●	■	▲	–	×	?	?	?
		FHE DiNN	[112]	2018	2	✓	–	✓	✓	◐		◆	◇										●		▲	▲	✓	✓	×	×
		Jiang et al.	[113]	2018	2	✓	–	✓	✓	◐		◇		◆									●		▲	▲	✓	✓	×	×
	Hybrid	Chameleon	[100]	2018	2	✓	–	✓	✓	◐		●	●		●	●	●								▲	–	×	?	?	?
		GAZELLE	[96]	2018	2	✓	–	✓	✓	◐		◇	●	◆		●					●				▲	▲	✓	✓	×	×
		Delphi	[119]	2020	2	✓	–	✓	✓	◐		◇	●	◆		●						●			▲	▲	✓	✓	×	×
	SMPC	Reza Sadeghi et al.	[48]	2008	2	✓	–	✓	✓	◐			●		●										▲	–	×	?	?	?
		MiniONN	[114]	2017	2	✓	–	✓	✓	◐			●		●										▲	–	×	?	?	?
		Deep Secure	[115]	2018	2	✓	–	✓	✓	◐			●		●	●									▲	–	×	?	?	?
		XONN†	[116]	2019	2	✓	–	×	✓	◐			●		●	●									▲	▲	×	?	?	✓
		Dalskov et al.	[117]	2020	2	✓	–	✓	✓		●	◇		◆			●		●				■		▲	▲	✓	✓	✓	×
DL Training	HE	Hesamifard et al.	[130]	2018	2	✓	–	×	✓	◐		◇		◆								●			▲	–	×	?	?	?
		Nandakumar et al.	[131]	2019	2	✓	–	×	✓	◐		◇		◆								●			▲	–	×	?	?	?
	Hybrid	ABY3	[99]	2018	3	✓	×	✓	✓		●	●	●	◆	●	●	●			●					▲	▲	✓	×	×	×
		SecureNN	[95]	2018	3	✓	×	✓	✓	◐		◇	●	◆	●	●	●								▲	▲	✓	✓	×	×
		Poseidon	[37]	2020	n	✓	×	×	✓	◐		●		●			●						●		▲	–	×	?	?	?
	SMPC	Chase et al.	[132]	2017	n	✓	✓	×	✓	◐		●					●							■	▲	–	×	?	?	?
		SecureML	[50]	2017	2	✓	×	✓	✓	◐		◇	●	◆	●	●	●					●			▲	▲	✓	✓	×	×
		QUOTIENT	[133]	2019	2	✓	–	×	✓	◐			●		●										▲	–	×	?	?	?
		Coded Private ML	[105]	2019	n	✓	×	✓	✓	◐		●						●							▲	–	×	?	?	?
		FLASH	[134]	2020	4	✓	×	✓	✓		●		◇	◆			●		●	●					▲	–	×	?	?	?
APIs and Compilers	-	TASTY†	[103]	2010	2	✓	–	✓	✓	◐		●									●				▲	▲	✓	✓	×	✓
		PySyft	[128]	2018	n	✓	×	×	✓		●	●			●		●		●	●		●			–	▲	✓	×	✓	✓
		TFEncrypted	[136]	2018	n	✓	×	×	✓		●	●			●		●		●	●		●			–	▲	✓	×	×	✓
		nGraph HE	[122]	2018	2	✓	×	×	✓	◐		●		●								●			–	▲	✓	✓	✓	✓
		XONN†	[116]	2019	2	✓	–	×	✓	◐			●		●	●									▲	▲	×	?	?	✓
		CHET, EVA	[120, 121]	2019	2	✓	–	✓	✓	◐												●			–	▲	✓	✓	✓	×
		nGraph HE 2	[123]	2019	2	✓	×	×	×	◐		●		●								●			–	▲	✓	✓	✓	✓
		MP2ML	[124]	2019	2	✓	–	✓	✓	◐		●		●	●	●						●			–	▲	✓	✓	✓	✓
		PlaidML HE	[126]	2019	2	✓	–	✓	✓	◐		●		●									●		–	▲	✓	✓	×	✓
		CrypTFlow	[129]	2020	3	✓	×	✓	✓		●	●			●	●	●			●				■	–	▲	✓	✓	✓	✓
		TinyGarble2	[127]	2020	2	✓	–	✓	✓		●		●		●	●			●	●					–	▲	✓	✓	×	×

Table 1. Summary of the papers covered in this study, based on different criteria (protocols marked as † are purposely repeated for clarity, due to their contributions to multiple objectives). For notation, we use ✓ and × to denote whether the context of the contribution provides input and/or output privacy, model secrecy, and open-source implementation or adaptation to a DL framework. We also use – when the feature is dependent on the specific implementation. We denote ● when the proposal either assumes a malicious adversary, or it uses a specific arithmetic or technique. We use ◐ when the proposal only assumes a HBC adversary. We use ■ for techniques that are used in the contribution, but we are not covering. We denote ◇ and ◆ when the contribution proposes the use of one arithmetic (◇) to emulate another (◆). We also depict the improvement in Efficiency and Usability with ▲ and – using the criteria described in Section 3.

CrypTFlow [129]) to easily adapt existing trained models to PPDL protocols.

There is a lack of interoperability of the different **APIs and compilers for PPDL training**. While all these interfaces implement approaches that allow for forward propagation, backpropagation is still not covered. Indeed, depending on the protocol, backpropagation may require the approximation of derivatives of the different activation and loss functions, and also the optimization of larger circuits. Furthermore, training often requires multiple epochs, i.e., to perform an arbitrary number of operations. Most APIs and compilers use finite circuit approaches and account for them for performance improvements, i.e., do not allow for unlimited operations to be performed.

Concerning **arithmetic types**, we highlight the need for boolean and floating-point arithmetic types for linear and non-linear activations (respectively). In this line, hybrid approaches have shown a strong impact, and solutions such as CHIMERA [97] open new paths for optimization. While authors have used integer and boolean arithmetic for discretized and binary neural networks, these are rare and difficult to adapt to modern DL solutions for complex problems. Accordingly, CKKS [62] which natively supports floating-point arithmetic, has been used in various proposals that propose different approximations from integer arithmetic.

Privacy goals are those security guarantees that a protocol aims to protect. As explained in Section 2.3, we consider that a proposal provides input privacy if the data processing reveals no user information. All the covered works provide input privacy (indeed, this is a key feature of PPCT). Output privacy guarantees that the result of a DL algorithm does not reveal additional information from the data. It involves the protection of DL models against other attacks, such as membership inference or model inversion. While this belongs to a broader and active area of research (i.e., adversarial machine learning [12]), only two works propose mechanisms to incorporate output privacy guarantees, i.e., Faster Cryptonets [108] and the work by Chase et al. [132]. Note that most of the works on privacy-preserving inference assume that DL models are securely pre-trained (i.e., they do not contain or reveal private information). Additionally, they consider the cloud provider as the model owner or a trusted third party for the training (i.e., it establishes security measures and does not attack the model). Accordingly, while the proposals might not address output privacy, providing such a guarantee would involve combining it with other techniques such as Differential Privacy. Regarding model secrecy, most of the works address both weight secrecy and architectural secrecy. However, the latter is more difficult to guarantee, given the patterns present in common DL layers and activations.

Adversarial Security Models define the threats and adversary capabilities on a cryptographic protocol. Most of the literature works usually assume an HBC adversary, who is limited in its offensive capabilities. Considering this kind of adversary lowers the performance requirements. Multiparty computation protocols can be transformed into the malicious adversary setting through commitment schemes, ZKP, and distributed randomness protocols [80, 83–86]. However, it is unclear whether cryptographic bridges such as the ones implemented in GAZELLE [96] or ABY [98] can be attested, and also what their performance would be on the malicious setting. Indeed, some contributions tackling the malicious adversary setting rely on trusted hardware (i.e., TEE) for code verification and confidentiality [117]. We believe it is important to explore the translation of hybrid protocols to malicious adversary models and efficient ZKP for distributed protocols.

6.2 Take-Aways

We enumerate key take-aways from our work, highlighting lessons learned and points that deserve further research effort:

1. While classical cryptography is relatively easy to use (e.g., due to easy-to-use and curated libraries), modern cryptography such as HE requires more tuning and expertise. It calls for schemes (e.g., CKKS [62]) and solutions (e.g., ABY [98]) that allow for automatic analysis and adaptation to specific circuits and the election of optimal parameters [138]. For efficient use of PPDL, these solutions should implement hybrid techniques to offer protocol conversion mechanisms, which will improve the overall performance of multi-arithmetic systems.
2. Efficiency and usability are confronting goals. Most of the works focused on efficiency improvements modify the original protocols to a great extent, which is detrimental for their integration into other frameworks. Meanwhile, works that focus on usability (i.e., APIs and compilers) involve few adjustments from basic techniques and original protocols, but have significant worse runtime performance than those focused on efficiency. Additionally, as an-

alyzed before, current APIs and compilers are not suitable for PPDL training, being this an area of research deserving more attention in the near future.

3. Even if proposals release their source code openly for reproducibility, the comparison in terms of efficiency and accuracy among contributions is difficult due to different benchmarking across articles. Different works claim different runtimes and accuracy across different DL models and different computing architectures. Additionally, due to ad-hoc optimizations applied on each proposal, we cannot easily measure the overall impact of each optimization on the protocol. It is thus needed to provide standardized trained models and benchmarks for an accurate and fair comparison of the proposals. Again, this would need to ease the integration of new works with said standard benchmarks.
4. To lower down the entry barrier for data scientists, it is essential to provide privacy-preserving extensions for existing DL frameworks, e.g., Tensorflow or PyTorch. Currently, this adaptation requires scientists to i) select a privacy-preserving solution or protocol, ii) implement adaptions to the protocol, and iii) provide an interface or bridge with the used DL framework. That is, implementing a full software stack. It is a challenging yet desirable goal to build standard solutions to adapt existing PPDL solutions by minimally changing the original code.
5. Depending on the scenario, providing only privacy in the inference part is insufficient. For example, output privacy is also needed to prevent fines due to leakage of sensitive data, e.g. by means of membership inference attacks. However, as our analysis suggests, current proposals on PPDL inference leave this part aside. Data scientists and engineers might require to combine techniques that guarantee both input and output privacy on their protocols. But, as mentioned before, integrating and combining these proposals with others is not straightforward.
6. Except for classic Yao's Garbled Circuits, most of the PPCTs reveal the performed operations. As pointed out in §2, architectural secrecy ensures the non-revelation of the architecture to untrusted parties. In contexts where the cloud server is not a trusted third party, the common patterns in neural network operations allow inferring the neural network architecture to the party performing the processing. Therefore, to better protect the privacy and security of the DL models, works not only hiding data but also processing (e.g., using whenever available Indistinguishable Obfuscation [139]).
7. In general, protocols assume participating entities to these protocols are honest-but-curious. However, we note a lack of works considering a malicious setting, limiting the deployment of PPCTs to scenarios where honest-but-curious adversaries cannot be guaranteed. However, previous experience shows that honest-but-curious settings are limited, e.g., due to internal security breaches or insiders.
8. Similar to other Privacy-Enhancing Technologies, there is the potential misuse of PPTC from malicious adversaries. Indeed, since PPCTs ensure the input privacy of users, this protection can conceal adversarial behavior. For example, a user trying to extract model parameters will carefully craft input data to create inference samples that yield knowledge about the model. Detecting these adversarial samples might require monitoring the inputs. In such cases, due to the use of PPCT, the detection mechanisms become unusable. Thus, it would be desirable to explore how to integrate this technology in adversarial settings.

7 Conclusions

In this paper, we provide a comprehensive review of Privacy-Preserving Computation Techniques for DL. First, we describe a high-level overview of the different privacy requirements, techniques, and architectures used in DL. It allows understanding the broad spectrum of PPDL and the benefits and constraints of the various sub-areas depending on the goals and settings. Then, intending to understand how to make MLaaS more attractive and privacy-friendly, we leverage the taxonomy to narrow down the scope of the study into cryptographic techniques for privacy-preserving computation (i.e., HE and SMPC). We study the state-of-the-art and provide an analysis according to attributes, such as the privacy goal, architecture, efficiency, and usability. It allows us to extract recurring insights and flaws from the current solutions. We believe this work will guide future researchers and practitioners towards a better deployment of privacy-preserving solutions for DL.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Phillipp Schoppmann, for their valuable feedback. We also thank Alberto Di Meglio, Marco Manca

and Rosario Cammarota for their helpful comments. This work was partially supported by CERN openlab, the CERN Doctoral Student Programme, the Spanish grants ODIO (PID2019-111429RB-C21 and PID2019-111429RB) and the Region of Madrid grants CYNAMON-CM (P2018/TCS-4566), co-financed by European Structural Funds ESF and FEDER, and Excellence Program EPUC3M17. The opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect those of any of the funders.

References

[1] James B Heaton, Nick G Polson, and Jan Hendrik Witte. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1):3–12, 2017.

[2] Daniel S Berman, Anna L Buczak, Jeffrey S Chavis, and Cherita L Corbett. A survey of deep learning methods for cyber security. *Information*, 10(4):122, 2019.

[3] Eric J Topol. High-performance medicine: the convergence of human and artificial intelligence. *Nature medicine*, 25(1):44–56, 2019.

[4] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P Wellman. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414. IEEE, 2018.

[5] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint:1603.04467*, 2016.

[7] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[8] M Sadegh Riazi, Bita Darvish Rouani, and Farinaz Koushanfar. Deep learning on private data. *IEEE Security & Privacy*, 17(6):54–63, 2019.

[9] Harry Chandra Tanuwidjaja, Rakyong Choi, Seunggeun Baek, and Kwangjo Kim. Privacy-preserving deep learning on machine learning as a service—a comprehensive survey. *IEEE Access*, 8:167425–167447, 2020.

[10] D Julkowska, CP Austin, CM Cutillo, D Gancberg, C Hager, J Halftermeyer, AH Jonker, LPL Lau, I Norstedt, A Rath, et al. The importance of international collaboration for rare diseases research: a european perspective. *Gene therapy*, 24(9):562–571, 2017.

[11] Huili Chen, Siam Umar Hussain, Fabian Boemer, Emmanuel Stapf, Ahmad Reza Sadeghi, Farinaz Koushanfar, and Rosario Cammarota. Developing privacy-preserving ai systems: the lessons learned. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–4. IEEE, 2020.

[12] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[13] Ram Shankar Siva Kumar, Magnus Nyström, John Lambert, Andrew Marshall, Mario Goertzel, Andi Comissoneru, Matt Swann, and Sharon Xia. Adversarial machine learning-industry perspectives. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 69–75. IEEE, 2020.

[14] Monir Azraoui, Muhammad Bahram, Beyza Bozdemir, Sébastien Canard, Eleonora Ciceri, Orhan Ermis, Ramy Masalha, Marco Mosconi, Melek Önen, Marie Paindavoine, et al. Sok: Cryptography for neural networks. In *IFIP International Summer School on Privacy and Identity Management*, pages 63–81. Springer, 2019.

[15] Georgios A Kaissis, Marcus R Makowski, Daniel Rückert, and Rickmer F Braren. Secure, privacy-preserving and federated machine learning in medical imaging. *Nature Machine Intelligence*, pages 1–7, 2020.

[16] Ting Wang and Ling Liu. Output privacy in data mining. *ACM Transactions on Database Systems*, 36(1):1–34, 2011.

[17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[18] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[19] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.

[20] Yehuda Lindell. Secure multiparty computation. *Commun. ACM*, 64(1):86–96, December 2020.

[21] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.

[22] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to solve any protocol problem. In *Proc. of STOC*, 1987.

[23] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[24] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[25] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es, 2007.

[26] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115. IEEE, 2007.

[27] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[28] David Kaplan, Jeremy Powell, and Tom Woller. Amd memory encryption. *White paper*, 2016.

[29] Johannes Winter. Trusted computing building blocks for embedded linux-based arm trustzone platforms. In *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, pages 21–30, 2008.

[30] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint:1803.05961*, 2018.

[31] Nick Hynes, Raymond Cheng, and Dawn Song. Efficient deep learning on multi-source private data. *arXiv preprint:1807.06689*, 2018.

[32] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint:1806.03287*, 2018.

[33] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. Cachezoom: How sgx amplifies the power of cache attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 69–90. Springer, 2017.

[34] Zhao-Hui Du, Zhiwei Ying, Zhenke Ma, Yufei Mai, Phoebe Wang, Jesse Liu, and Jesse Fang. Secure encrypted virtualization is unsecure. *arXiv preprint:1712.05090*, 2017.

[35] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint:1610.02527*, 2016.

[36] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015.

[37] Sinem Sav, Apostolos Pyrgelis, Juan R Troncoso-Pastoriza, David Froelicher, Jean-Philippe Bossuat, Joao Sa Sousa, and Jean-Pierre Hubaux. POSEIDON: Privacy-preserving federated neural network learning. *arXiv preprint:2009.00349*, 2020.

[38] Tao Yu, Eugene Bagdasaryan, and Vitaly Shmatikov. Salvaging federated learning by local adaptation. *arXiv preprint:2002.04758*, 2020.

[39] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint:1912.04977*, 2019.

[40] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint:1902.01046*, 2019.

[41] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

[42] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint:1610.05755*, 2016.

[43] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. Scalable private learning with PATE. *arXiv preprint:1802.08908*, 2018.

[44] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[45] James Jordon, Jinsung Yoon, and Mihaela van der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2018.

[46] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

[47] Damien Desfontaines and Balázs Pejó. Sok: Differential privacies. *CoRR*, abs/1906.01337, 2019.

[48] Ahmad-Reza Sadeghi and Thomas Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *International Conference on Information Security and Cryptology*, pages 336–353. Springer, 2008.

[49] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 619–636, 2016.

[50] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 19–38. IEEE, 2017.

[51] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[52] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[53] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1592, pages 223–238. Springer Verlag, 1999.

[54] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[55] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[56] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[57] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640. Springer, 2015.

[58] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*,

2012:144, 2012.

[59] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[60] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual Cryptology Conference*, pages 75–92. Springer, 2013.

[61] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *international conference on the theory and application of cryptology and information security*, pages 3–33. Springer, 2016.

[62] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer, 2017.

[63] Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL, November 2020. Microsoft Research, Redmond, WA.

[64] Shai Halevi and Victor Shoup. Algorithms in helib. Cryptology ePrint Archive, Report 2014/106, 2014. https://eprint.iacr.org/2014/106.

[65] Yuriy Polyakov, Kurt Rohloff, and Gerard W Ryan. PALISADE lattice cryptography library user manual. *Cybersecurity Research Center, New Jersey Institute of Technology (NJIT), Tech. Rep*, 2017.

[66] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[67] David W Archer, José Manuel Calderón Trilla, Jason Dagit, Alex Malozemoff, Yuriy Polyakov, Kurt Rohloff, and Gerard Ryan. Ramparts: A programmer-friendly system for building homomorphic encryption applications. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 57–68, 2019.

[68] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo: a compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, pages 13–19, 2015.

[69] Michael O Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.*, 2005(187), 2005.

[70] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the 31st ACM Symposium on Theory of Computing*, pages 245–254, 1999.

[71] A. C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[72] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. Secure two-party computation is practical. In *International conference on the theory and application of cryptology and information security*, pages 250–267. Springer, 2009.

[73] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 220–250. Springer, 2015.

[74] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.

[75] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for xor gates that beats free-xor. In *Annual Cryptology Conference*, pages 440–457. Springer, 2014.

[76] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139, 1999.

[77] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513, 1990.

[78] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[80] Rosario Gennaro, Michael O Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 101–111, 1998.

[81] George Robert Blakley. Safeguarding cryptographic keys. In *1979 International Workshop on Managing Requirements Knowledge (MARK)*, pages 313–318. IEEE, 1979.

[82] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pages 420–432. Springer, 1991.

[83] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science*, pages 427–438. IEEE, 1987.

[84] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science*, pages 383–395. IEEE, 1985.

[85] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM (JACM)*, 38(3):690–728, 1991.

[86] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 1–10, New York, NY, USA, 1988. ACM.

[87] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 103–112, New York, NY, USA, 1988. Association for Computing Machinery.

[88] Paul Feldman and Silvio Micali. An optimal probabilistic algorithm for synchronous byzantine agreement. In *International Colloquium on Automata, Languages, and Programming*, pages 341–378. Springer, 1989.

[89] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*,

15(1):23–27, 1983.

[90] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Annual Cryptology Conference*, pages 643–662. Springer, 2012.

[91] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016.

[92] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 158–189. Springer, 2018.

[93] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[94] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *2015 IEEE Symposium on Security and Privacy*, pages 411–428. IEEE, 2015.

[95] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: Efficient and private neural network training. *IACR Cryptol. ePrint Arch.*, 2018:442, 2018.

[96] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, 2018.

[97] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020.

[98] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY-A framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[99] Payman Mohassel and Peter Rindal. mixed protocol framework for machine learning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 35–52, 2018.

[100] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 707–721, 2018.

[101] Nishanth Chandran, Divya Gupta, Aseem Rastogi, Rahul Sharma, and Shardul Tripathi. EzPC: programmable, efficient, and scalable secure two-party computation for machine learning. *ePrint Report*, 1109, 2017.

[102] Jaideep Vaidya and Chris Clifton. Privacy-preserving data mining: Why, how, and when. *IEEE Security & Privacy*, 2(6):19–27, 2004.

[103] Wilko Henecka, Stefan K ögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 451–462, 2010.

[104] Thore Graepel, Kristin Lauter, and Michael Naehrig. ML confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21. Springer, 2012.

[105] Jinhyun So, Basak Guler, A Salman Avestimehr, and Payman Mohassel. CodedPrivateML: A fast and privacy-preserving framework for distributed machine learning. *arXiv preprint:1902.00641*, 2019.

[106] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin Lauter, and Michael Naehrig. Cryptonets: Neural networks over encrypted data. *arXiv preprint:1412.6181*, 2014.

[107] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[108] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint:1811.09953*, 2018.

[109] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint:1711.05189*, 2017.

[110] Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In *International Conference on Machine Learning*, pages 4490–4499, 2018.

[111] Nick Barlow and Oliver Strickson. SHEEP is a homomorphic encryption evaluation framework. https://github.com/alan-turing-institute/SHEEP, 2018.

[112] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, pages 483–512. Springer, 2018.

[113] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1209–1222, 2018.

[114] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017.

[115] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[116] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. XONN: XNOR-based oblivious deep neural network inference. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1501–1518, 2019.

[117] Anders Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies*, 2020(4):355–375, 2020.

[118] Goldreich Oded. Foundations of cryptography: Volume 2, basic applications, 2009.

[119] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. DELPHI: A cryp-

tographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[120] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 142–156, 2019.

[121] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. Eva: an encrypted vector arithmetic language and compiler for efficient homomorphic computation. *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, Jun 2020.

[122] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 3–13, 2019.

[123] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE2: A high-throughput framework for neural network inference on encrypted data. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 45–56, 2019.

[124] Fabian Boemer, Rosario Cammarota, Daniel Demmler, Thomas Schneider, and Hossein Yalame. MP2ML: a mixed-protocol machine learning framework for private inference. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, pages 1–10, 2020.

[125] Scott Cyphers, Arjun K Bansal, Anahita Bhiwandiwalla, Jayaram Bobba, Matthew Brookhart, Avijit Chakraborty, Will Constable, Christian Convey, Leona Cook, Omar Kanawi, et al. Intel ngraph: An intermediate representation, compiler, and executor for deep learning. *arXiv preprint:1801.08058*, 2018.

[126] Huili Chen, Rosario Cammarota, Felipe Valencia, and Francesco Regazzoni. PlaidML-HE: Acceleration of deep learning kernels to compute on encrypted data. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pages 333–336. IEEE, 2019.

[127] Siam Hussain, Baiyu Li, Farinaz Koushanfar, and Rosario Cammarota. Tinygarble2: Smart, efficient, and scalable yao's garble circuit. In *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, pages 65–67, 2020.

[128] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint:1811.04017*, 2018.

[129] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 336–353. IEEE, 2020.

[130] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N Wright. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies*, 2018(3):123–142, 2018.

[131] Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. Towards deep neural network training on encrypted data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[132] Melissa Chase, Ran Gilad-Bachrach, Kim Laine, Kristin E Lauter, and Peter Rindal. Private collaborative neural network learning. *IACR Cryptol. ePrint Arch.*, 2017:762, 2017.

[133] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J Kusner, and Adrià Gascón. Quotient: two-party secure neural network training and prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1231–1247, 2019.

[134] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. Flash: fast and robust framework for privacy-preserving machine learning. *Proceedings on Privacy Enhancing Technologies*, 2020(2):459–480, 2020.

[135] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. Astra: High throughput 3pc over rings with application to secure prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pages 81–92, 2019.

[136] Morten Dahl, Jason Mancuso, Yann Dupis, Ben Decoste, Morgan Giraud, Ian Livingstone, Justin Patriquin, and Gavin Uhma. Private machine learning in tensorflow using secure computation. *arXiv preprint:1810.08130*, 2018.

[137] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. Cryptology ePrint Archive, Report 2020/521, 2020. https://eprint.iacr.org/2020/521.

[138] A. Viand, P. Jattke, and A. Hithnawi. Sok: Fully homomorphic encryption compilers. In *2021 2021 IEEE Symposium on Security and Privacy (SP)*, pages 1166–1182, Los Alamitos, CA, USA, may 2021. IEEE Computer Society.

[139] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. *arXiv preprint:2008.09317*, 2020.

[140] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[141] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[142] Mohamad H Hassoun et al. *Fundamentals of artificial neural networks*. MIT press, 1995.

[143] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[144] Nathan O Hodas and Panos Stinis. Doing the impossible: Why neural networks can be trained at all. *Frontiers in psychology*, 9:1185, 2018.

[145] David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *International Conference on Machine Learning*, pages 342–350. PMLR, 2017.

[146] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[147] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[148] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[149] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[150] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[151] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[152] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

A Deep Learning Background

Deep Learning is a machine learning technique derived from Artificial Neural Networks (ANN) [140, 141]. ANNs are biologically inspired statistical models that resemble the neural connections, where neurons are connected in a network, and they activate each other's based on a stimulus [142]. An ANN consists of a layered network of neurons that are instrumented, through a training process, to learn how to solve complex problems. The inner mathematical function of each neuron consists of a linear part that computes the product of internal weights with the inputs from previous neurons; and a non-linear part used as activation function expressing distinctive behaviors of neurons. In ANNs, neurons are grouped in layers and connected to the following layers in specific patterns. The organization of the layers is also known as architecture. Once the model is defined, the training consists of two steps. Firstly, forward propagation infers the prediction on an input training sample. Secondly, backward propagation evaluates the influence of each neuron on the result, and modifies the weights according to a loss function. Often, ANNs implement Gradient Descent (or variations of it).

Learning Theory states that a single layer would be enough to learn any function [141, 143]. However, it does not account for the computational complexity of such neurons. Advances in computations and also mathematical science have shown that combining various networks in deeper (i.e., with more layers) architectures obtain similar results than making shallower and wider ANNs efficiently [144, 145]. This is what constitutes *Deep* Learning (DL). Moreover, there are software frameworks that facilitate the training and use of DL for data scientists (e.g., Tensorflow [6], Keras [5] or PyTorch [7]). The science in DL consists of finding the best architecture for a particular problem, and training the model to approximate a function that better fits the input data. We next describe the linear and non-linear structures of Deep Neural Networks (DNN).

A.1 Linear Structures

We describe the two most popular common linear layers. **Dense Layer**. In this layer, all the neurons receive the input from the all the neurons of the previous layer and send its output to the next one. The process to compute the output activation is $a = \sigma(w_0 + w_1 \cdot x_1 + ... + w_n \cdot x_n)$ which multiplies the weights w_i by the inputs of the previous layer x_i and applies the activation function σ. **Convolutional Layer**. This layer is based on the concept of image filters and thus only considers a subset of the input [146, 147]. This layer typically applies to image processing applications. It operates over a subset of n by n pixels and reduces it to a single output. The filter is shifted all over the pixels of the input image, resulting in a reduced representation of a feature. The operations carried on the filter are determined by weights that are computed during training. In that way, neural networks can infer about different shapes found in images. Convolutional layers reduce the number of parameters and training time. They are often used in conjunction with batch normalization layers [148] and dropout layers [149] to improve its training convergence.

A.2 Non-Linear Structures

For the non-linear structures, we differentiate those aimed at *activating* and *pooling* the neurons.

A.2.1 Activation Functions

Activation functions are fundamental components that determine the intensity of the output of a neuron [150], and they control the learning factor for the different weights. They are a key aspect of DL training, since otherwise a neural network without activation becomes a linear regression [141]. There are several examples of it, but the most commonly used ones are:

Sigmoid $\sigma(x) = (1 + e^{-x})^{-1}$ This activation shifts towards values being either 0 or 1. It is used as the last layer activation. It enables a faster minimization of the cost function (i.e., values are shifted towards 0 or 1) [151].
ReLU $ReLU(x) = max(0, x)$ This function [152] is used in intermediate layers of the NN. It has the advantage of allowing gradient updates to propagate correctly to the first layers. That is, for very deep architectures, the use of ReLU is essential.
Softmax $\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$ This function has similar behavior to the sigmoid function but for multiclass classification outputting a higher probability of one class and lower for the rest.

A.2.2 Max Pooling and Average Pooling

The max-pooling and average pooling layers are subsampling structures that permit obtaining significant values from the results of previous layers. Pooling layers act on $N \times N$ areas of the matrix and compute the max or average of it. Thus, they further reduce the representation of the input and speed up the computation. In CNN's Max Pooling layers are directly coupled to the success of convolutional layers [141], and thus, they are always used together. In the case of PPDL, the use of max-pooling layers is minimal due to the complexity to approximate the max function, and therefore the average pooling is often used.

B Generic DL Pipeline

Figure 1 shows an schematic view of the generic architecture for Privacy-Preserving Deep Learning.

C Open-Source Repositories

Table 2 shows the URLs for the various open-source repositories analysed in this work.

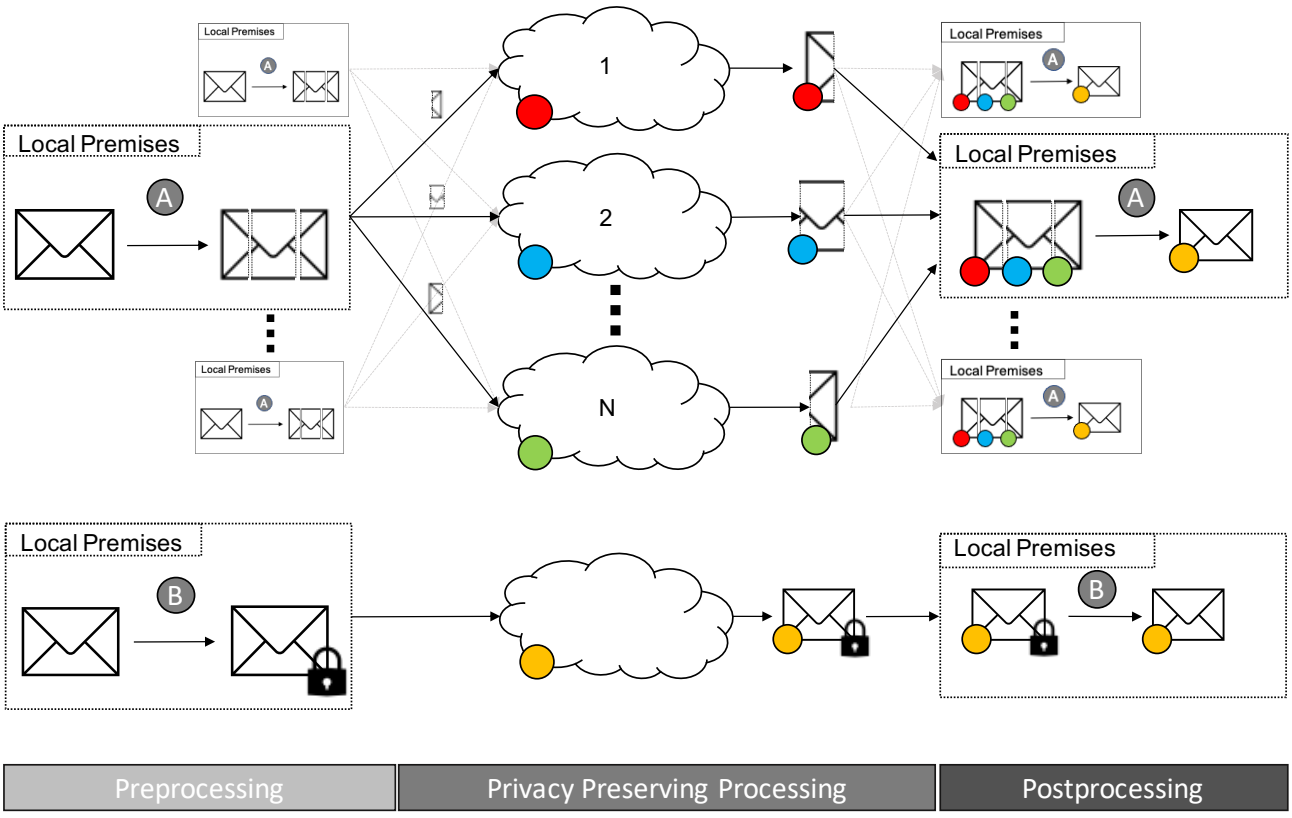

Fig. 1. Generic Architecture for Privacy Preserving Data Processing.

Proposal	Ref.	URL	Last Update
Dalskov et al.	[117]	github.com/data61/MP-SPDZ	27/05/2021
PySyft	[128]	github.com/OpenMined/PySyft	24/05/2021
CrypTFlow	[129]	github.com/mpc-msri/EzPC	12/05/2021
CHET/EVA	[120, 121]	github.com/microsoft/EVA	01/05/2021
ABY3	[99]	github.com/ladnir/aby3	08/03/2021
PlaidML HE	[126]	github.com/plaidml/plaidml	21/11/2020
TinyGarble2	[127]	github.com/IntelLabs/TinyGarble2.0	16/11/2020
MP2ML	[124]	github.com/IntelAI/he-transformer	03/11/2020
nGraph HE 2	[123]	github.com/IntelAI/he-transformer	03/11/2020
nGraph HE	[122]	github.com/IntelAI/he-transformer	03/11/2020
SecureNN	[95]	github.com/snwagh/securenn-public	01/11/2020
Dalskov et al.	[117]	github.com/anderspkd/SecureQ8	06/10/2020
TFEncrypted	[136]	github.com/tf-encrypted/tf-encrypted	19/08/2020
Delphi	[119]	github.com/mc2-project/delphi	14/08/2020
SecureML	[50]	github.com/shreya-28/Secure-ML	01/10/2019
Cryptonets	[106, 107]	github.com/microsoft/CryptoNets	12/09/2019
Jiang et al.	[113]	github.com/K-miran/HEMat	07/08/2019
TAPAS	[110]	github.com/amartya18x/tapas	28/05/2019
GAZELLE	[96]	github.com/chiraag/gazelle_mpc	17/09/2018
FHE DiNN	[112]	github.com/mminelli/dinn	21/11/2017
TASTY	[103]	github.com/tastyproject/tasty	13/11/2014

Table 2. URLs for the open-source repositories of the different contributions analyzed [Last review on May 27th, 2021].