TASK Quarterly 28 (2) 2024

A Survey on Privacy-Preserving Machine Learning Inference

Stanislaw Baranski
stanislaw.baranski@pg.edu.pl
Faculty of Electronics, Telecommunication and Informatics, Gdansk University of Technology,
Gabriela Narutowicza 11/12, 80-233, Gdańsk, Poland

https://doi.org/10.34808/tq2024/28.2/b

Abstract

This paper examines methods to secure machine learning inference (ML inference) so that sensitive data remains private and proprietary models are protected during remote processing. We review several approaches ranging from cryptographic techniques like homomorphic encryption (HE) and secure multi-party computation (MPC) to hardware solutions such as trusted execution environments (TEEs) and complementary methods including differential privacy and split learning. Each method is analyzed in terms of security, efficiency, communication overhead, and scalability. Use cases in healthcare, finance, and education show how these techniques balance privacy with practical performance. We conclude by outlining open challenges and future directions for building robust, efficient privacy-preserving ML inference systems.

Keywords:

Privacy-Preserving Machine Learning, Oblivious Neural Networks, Homomorphic Encryption, Secure Multi-Party Computation, Trusted Execution Environments, Differential Privacy, Split Learning

1. Introduction

Machine Learning as a Service (MLaaS) has become increasingly popular, but it raises significant privacy concerns when users must send sensitive data to a remote server for inference [1], [2]. For example, a healthcare provider might want to use a cloud-hosted model for diagnosis, yet patient data is protected by regulations like HIPAA and GDPR [3]. Likewise, companies may deploy proprietary models that they do not wish to reveal to users, creating a need to protect the model itself [4]. Privacy-Preserving Machine Learning (PPML) inference addresses these issues by enabling inference on encrypted or otherwise protected data, so that the server learns nothing about the client's input and, optionally, the client learns nothing about the model beyond the output. The solution to these challenges requires knowledge to be drawn from four interconnected fields: machine learning, computation theory, digital systems theory, and cryptography.

Prior research has demonstrated that without PPML, sensitive information can leak during inference. Adversaries might recover aspects of the input data or even properties of the training set via inference attacks (membership inference, model inversion, etc.) [4], [5]. Initially, fully homomorphic encryption (FHE) was proposed to allow computations on encrypted data, and early works like CryptoNets (2016) showed the feasibility of neural network inference on encrypted inputs [2]. Since then, a variety of approaches have emerged, each with trade-offs in security, efficiency, and accuracy.

This survey is organized as follows: Section 2 defines the PPML inference problem and threat models. Section 3 reviews major categories of PPML inference techniques: homomorphic encryption (HE), secure multi-party computation (MPC), trusted execution environments (TEEs), and privacy-aware data perturbation (e.g., differential privacy). Section 4 details practical use cases and recommendations. Section 5 discusses practical implementations and the state of the art, including performance metrics and current tools. Section 6 considers the big picture, alternative approaches like federated or split inference, industry solutions, and key open challenges. Finally, Section 7 concludes with observations on the future of PPML inference.

2. Prerequisites

A typical machine learning pipeline (Fig. 1) consists of two phases:

1. The training phase, in which the algorithm trains the model w using a training dataset.
2. The inference phase, in which the prediction algorithm $f(\cdot)$ makes a prediction $f(x,w)$ based on the input data x (called the parameter vector) and the previously trained model w. This prediction can either be a continuous value (regression) or a category assignment (classification).

We focus solely on the inference phase, assuming the model w is already trained.

In a typical PPML inference scenario, there are two parties: a client (data owner) with private input x (e.g., an image or personal record) and a server (model owner) with a machine learning model $f(\cdot)$ that may be proprietary. The goal is for the client to obtain $f(x,w)$ (the inference result) without revealing x to the server. Optionally, the server may also wish to keep the model parameters (w) secret from the client to protect intellectual property. This is often referred to as two-party privacy: input privacy for the client and model privacy for the server. In some settings, only the client's data privacy is required (the model can be public), while in others both must be preserved.

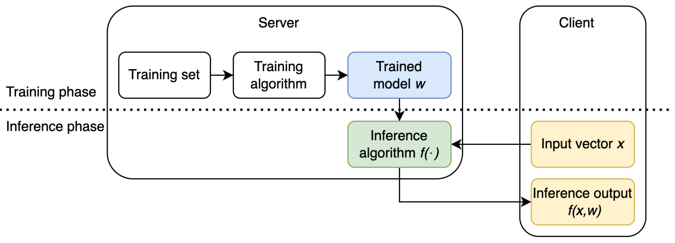

Figure 1: Diagram illustrating phases and parties involved in a typical machine learning pipeline.

In order to better understand the problem, let's define the function $f(\cdot)$ a bit more concretely.

2.1. Neural Networks

A neural network is a pipeline of layers. Each layer receives an input signal, processes it, and generates an output signal, which then serves as the input for the next layer. The first layer receives the input data x, and the output signal of the last layer is the prediction result $f(x,w)$.

A typical neural network layer performs a linear transformation (matrix multiplication and addition) followed by a nonlinear transformation (activation function, and sometimes also *pooling* to reduce data resolution).

Predictions using neural networks can be represented as a pipeline of transformations:

$$x \to f_1 \to a_1 \to \dots \to f_n \to a_n \to y$$

where:

- x is the input vector;
- y is the inference output;
- f_i is the linear transformation of layer i;

- a_i is the nonlinear transformation of layer i;
- n is the total number of layers in the neural network.

The goal of these transformations is to distort the input data space so that it becomes linearly separable, meaning that a line (if the input data is in two dimensions), a plane (in three dimensions), or a hyperplane (in $n+1$ dimensions) can be created to divide the input set into two subsets. Fig. 2 illustrates the transformations performed by each layer of a sample neural network. Linear separability of the data (into green and red regions) is achieved through a sequence of linear and nonlinear transformations. Linear transformations allow for rotation, tilting, and stretching of the space, while nonlinear transformations enable spatial deformation.

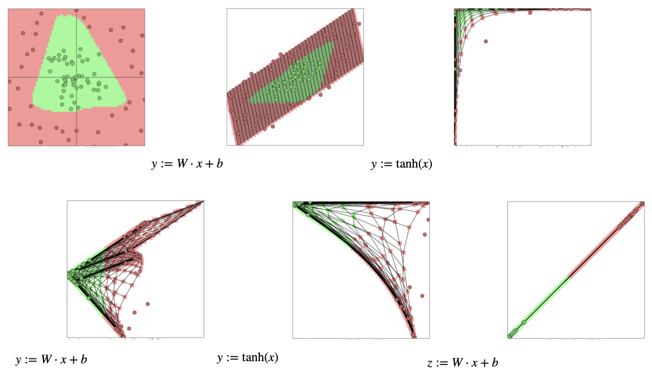

Figure 2: Visualization of linear and nonlinear transformations to achieve linear separability

2.2. Linear Transformations

Matrix Multiplication and Addition are the most common linear transformations used in neural networks:

$$y := W \cdot x + b \tag{1}$$

where:

- x is the input vector;
- W is the weight matrix;
- b is the bias vector;
- y is the output vector.

Convolution is a linear transformation that computes the dot product between a weight tensor (a filter, also called a kernel) and an element of the input matrix along with its neighboring elements. This process is repeated as the filter moves across the input matrix. In practice, convolutions are reformulated as matrix multiplication and addition to improve efficiency [6], similar to equation (1), with the difference that the input data and bias term are matrices: $Y := W \cdot X + B$.

2.3. Nonlinear Transformations

Neural networks utilize nonlinear transformations to model complex relationships between input and output spaces.

Activation Functions. There are three main categories of activation functions:

- *Piecewise Linear Activation Functions.* These functions can be represented as a set of n linear functions $f_i(x) = a_i x + b_i$, where x is bounded by lower and upper limits for a given interval. Examples include:
  - Identity function: $f(x) = x$;
  - *Rectified Linear Unit (ReLU)*: $f(x) = \max(0, x)$;
  - *Leaky ReLU*: $f(x) = \max(0, x) + a \min(0, x)$;
  - *Maxout*: $f(x) = \max(y_1, \ldots, y_n)$.
- *Smooth (Regular) Activation Functions.* These are differentiable functions, defined a certain number of times in their domain. The most popular smooth activation functions include:
  - *Sigmoid (logistic)*: $f(x) = \frac{1}{1+e^{-x}}$;
  - *Hyperbolic tangent (tanh)*: $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$;
  - *Softplus*: $f(x) = \log(e^x + 1)$.

  Sigmoid and *tanh* functions are known as sigmoidal functions, with the relationship:

$$\tanh(x) = 2 \cdot \operatorname{sigmoid}(2x) - 1 \tag{2}$$

- *Softmax.* This function is commonly used as the final layer in a neural network to determine a probability distribution for classification. It is defined as

$$\operatorname{softmax}_i(x) = \frac{e^{x_i}}{\sum_k e^{x_k}} \tag{3}$$

Pooling is an operation that reduces the resolution of a matrix. Also known as downsampling, it involves organizing input data into subgroups and aggregating each subgroup while reducing the dimensionality of the input data. The most common aggregation methods are:

- *Mean pooling* (averaging values);
- *Max pooling* (taking the maximum value).

2.4. Conclusions

The prediction phase of a neural network consists of a sequence of linear and nonlinear transformations.

- **Linear transformations** reduce to matrix multiplication and addition.
- **Nonlinear transformations** reduce to activation functions and pooling operations.

Therefore, the entire prediction process can be performed in a privacy-preserving manner using privacy-preserving matrix multiplication, addition, activation functions, and pooling operations.

2.5. Threat Model

The threat model typically considered is *honest-but-curious* (semi-honest): both client and server follow the protocol but may try to infer the other's private information from the messages they see [7]. Stronger threat models (malicious adversaries who deviate from protocol) require additional safeguards like zero-knowledge proofs or verifiable computation, which can add overhead [8]. Most current PPML inference research targets the semi-honest model for efficiency [7]. We also consider potential collusion in multi-party settings (e.g., if multiple servers are involved for MPC). The adversaries of interest include an outside attacker compromising the server or any party not authorized to learn the data.

Privacy in inference has two aspects: (1) Protecting the **input and output privacy**, so that the server learns nothing about the client's input (and ideally, the client learns nothing beyond the intended output). Cryptographic techniques excel at this aspect. (2) Protecting **training data privacy** from inference outputs. Even if the server never sees raw inputs, a curious client could potentially infer information about the model's training data by querying the model repeatedly (model inversion or membership attacks) [9]. Techniques like differential privacy can mitigate this by ensuring the model or its outputs do not reveal individual training examples.

3. Approaches to Privacy-Preserving ML Inference

Various strategies have been developed for PPML inference, each with different assumptions and performance. We categorize them into: (A) Homomorphic Encryption, (B) Secure Multi-Party Computation, (C) Trusted Execution Environments, and (D) Privacy-Preserving Data Perturbation. Often, hybrid approaches combine these to balance their strengths [7].

3.1. Homomorphic Encryption (HE) Approaches

Homomorphic encryption allows computation on ciphertexts. In a fully homomorphic encryption scheme, an arbitrary function (such as a neural network) can be evaluated on encrypted data, producing an encrypted result that, when decrypted, matches the result of the plaintext computation [2]. This powerful property directly enables private inference: the client encrypts input x with their public key pk and sends $Enc(x)$ to the server; the server evaluates f homomorphically to obtain $Enc(f(x))$ and returns it; the client decrypts to get $f(x)$. In this process, the server sees only encrypted values, so x remains confidential. If only the client holds sk (secret key), the server cannot decrypt any intermediate or final value.

Advantages: HE-based inference provides strong privacy of data (based on cryptographic hardness assumptions). Only one round of communication is needed (send encrypted input, receive encrypted output), which is appealing for high-latency networks. The privacy guarantee is information-theoretic for the data (given properly chosen parameters and secure schemes) – the server learns zero information about the plaintext. Another advantage is that HE does not require a trusted third party; the security relies on the math of encryption (e.g., lattice problems for schemes like BFV/CKKS). Crucially, modern HE schemes support packing multiple values in one ciphertext (SIMD operations) to amortize costs [10]. This yields high throughput in batch processing. For example, the CryptoNets work achieved 99% accuracy on MNIST digits with HE and processed 57,000 encrypted images per hour on a single PC [2], using packing to get amortized latency of a few tens of milliseconds per image (though the first inference had a much larger latency of several minutes).

Limitations: The downside of HE is its computational overhead. Fully homomorphic encryption was long deemed impractical due to enormous slowdown factors. Even with improvements, evaluating deep neural networks under HE is extremely slow compared to plaintext. Early demonstrations like CryptoNets required non-linear activations to be replaced with polynomial approximations to fit within HE's capabilities [2]. This can degrade accuracy or increase the polynomial degree (thus increasing runtime). Moreover, each ciphertext is large (often KBs), and arithmetic on them is expensive (multiplication operations can take tens of milliseconds). HE schemes have a noise budget that limits the depth of computation before requiring bootstrapping (a costly refresh operation). Although "leveled HE" can be used to avoid bootstrapping for shallow circuits by choosing large parameters in advance, this again increases runtime and ciphertext size.

In practice, pure HE inference was orders of magnitude slower. For instance, the CryptoNets network was relatively small (a few layers for MNIST). On more complex models, purely homomorphic inference might take tens of seconds or more per input. The Gazelle system noted that a purely HE approach (like CryptoNets) is over $1000\times$ slower than their hybrid method for CIFAR-10 models [10]. Recent work on FHE (fully homomorphic encryption with bootstrapping) has started to reduce these costs. For example, Chillotti et al. (Concrete library) demonstrated FHE inference on certain models, but at significant cost (seconds per inference) [11].

Thus, while HE ensures strong privacy, it often needs to be combined with optimizations or other techniques to be practical. Efforts like Cheetah (2021) focus on accelerating HE with algorithmic tuning and hardware acceleration, achieving ∼198 ms latency for ResNet-50 inference using a custom ASIC design [7]. These results suggest that with massive parallelism, HE could approach real-time for large models. However, on general-purpose hardware, current HE inference is typically limited to smaller models or low-throughput settings.

3.2. Secure Multi-Party Computation (MPC) Approaches

Secure multi-party computation allows joint computation on private inputs from two or more parties such that each learns only the designated output and nothing else. In the two-party case (client and server), MPC protocols can implement the function $f(x)$ as a series of cryptographic operations (secret shares or garbled circuits) between the parties [4].

In contrast, homomorphic encryption (HE) enables computation on encrypted data without interaction. The client encrypts the input using a public key and sends it to the server, which can then compute directly on the ciphertext. Only the client, who holds the private key, can decrypt the final result. This makes HE a public-key one-party computation technique, since the server performs the computation independently after receiving the encrypted input. MPC, however, usually involves interaction: both client and server interactively engage in a protocol exchanging messages.

Common MPC paradigms include Yao's Garbled Circuits (a two-party protocol for Boolean circuits) and secret-sharing based arithmetic MPC (which can involve two or more servers). For PPML inference, a widely used approach is to secret-share both the input and the model between two non-colluding servers and then use additive secret sharing for linear operations and garbled circuits for non-linear ones, as demonstrated in protocols such as SplitNN or SecureML [9], [12]. In this setting, the client's data and the server's model are split into shares that are distributed between two servers. Each server performs local computations on its share during the linear operations (e.g., matrix multiplications and additions), while the non-linear operations (such as activation functions) are executed via garbled circuits.

The advantage of this approach lies in the efficiency of secret sharing. In additive secret sharing, a secret is split into multiple shares so that no single share reveals any information about the secret. In a two-party setting (with only a client and a server), each party would hold one share. Although privacy can be maintained under honest-but-curious assumptions using protocols like Yao's Garbled Circuits with Oblivious Transfer (OT), such a setup typically requires the client to perform heavy computations or engage in costly interactive protocols for the non-linear operations. In contrast, when using two non-colluding servers, the client is only required to provide its input once—without being involved in the computationally intensive steps of the protocol—while the servers carry out the bulk of the processing.

Furthermore, in a three-party setting (client plus two non-colluding servers), the inputs to the garbled circuits are already secret-shared among the servers, so the standard need for Oblivious Transfer (OT)—which is essential in a two-party GC protocol to hide the evaluator's input—is largely avoided. In such protocols, the garbled circuit evaluation is performed jointly by the servers using preprocessed randomness and correlated values, thus avoiding the additional overhead associated with OT. In contrast, if only a single server is available, the client must act as one party and the server as the other in a two-party protocol using Yao's Garbled Circuits [13] and OT [14] to ensure that neither party learns the other's private input. Although this two-party model can maintain privacy under honest-but-curious assumptions, it is generally more computationally and communicationally expensive compared to the three-party secret-sharing approach.

Advantages: MPC-based solutions often have significantly lower computational cost than FHE-only, at the expense of more communication. They can be scaled to larger neural networks more easily. For instance, the Delphi system uses a preprocessing phase to handle most heavy cryptography before the input is known, achieving only a few seconds of online time for ResNet-32 inference (with some accuracy trade-offs) [4]. MPC protocols (especially with preprocessing) can leverage fast symmetric cryptography (AES, etc.) which is much faster than public-key operations. Also, MPC naturally supports protecting both input and model (since both can be secret-shared or encoded as inputs to the protocol). Many frameworks (like Facebook's CrypTen [15], Microsoft's CrypTFlow [16]) exist to facilitate implementing neural nets with MPC.

Limitations: The main cost of MPC is communication. Many protocols require multiple rounds of interaction. Every multiplication gate in a circuit might involve sending some bits or shares. In high-latency networks, this slows down inference. The data transferred can be large (Delphi cited 560MB for ResNet-32 with Gazelle-like method in one example). Also, if the client has low upload bandwidth (e.g., mobile device), sending large garbled circuits or many OT messages is problematic. Another limitation is that MPC typically assumes at least two non-colluding parties when using secret sharing for efficiency. If we only have the client and one server, the client ends up participating in the computation, which means the client might see intermediate results unless carefully masked. Protocols like Gazelle indeed require the client to do some work (decrypting and re-encrypting between layers). This shifts some compute burden to the client and introduces more rounds (each layer in Gazelle involves a round trip). Newer techniques try to minimize rounds (e.g., performing multiple ReLUs in one GC batch).

Several **hybrid protocols** combine HE and MPC to capitalize on their strengths. A prime example is **Gazelle** (2018), which achieves low latency by using homomorphic encryption for linear layers (fast dot-product on encrypted data) and garbled circuits for ReLU (which is a cheap boolean operation) [10]. Gazelle's approach yields $20\times$ faster online runtime than a prior pure-MPC method (MiniONN [17]) and $1000\times$ faster than pure HE (CryptoNets) [2]. Many subsequent works (Chameleon [18], Delphi [4], etc.) follow this template: **use additive HE or secret sharing for linear computations, use garbled or boolean circuits for non-linear parts**. This partition exploits the fact that linear layers dominate compute but are friendly to HE/SS, while nonlinear activations are few but not efficiently done in HE [7]. The client can assist by handling the garbled circuit for ReLUs, which slightly weakens model privacy (the client gets some info on intermediate values, mitigated by random masks as Gazelle does [10]).

Overall, MPC approaches are currently the most practical for complex deep learning models. For example, a 2022 work used 3-party MPC to run LeNet on MNIST in under 0.1 seconds online time [1]. Two-party approaches for large models still face challenges, but continuous improvements are being made to reduce communication via quantization, model structure changes, or combining with FHE for one-shot computation of whole layers.

3.3. Trusted Execution Environments (Hardware Enclaves)

Trusted Execution Environments (TEEs) like Intel SGX or ARM TrustZone provide hardware-protected regions of memory where computations can be performed in isolation. A TEE allows the server to load the model and run inference on plaintext data inside an enclave, with assurances that the data and model are not accessible to the rest of the system. With remote attestation, a client can verify that the correct enclave code (e.g., the inference algorithm) is running on genuine hardware before sending their encrypted input, which the enclave will decrypt and compute on. Using TEEs for PPML inference can be very efficient, since inside the enclave the computation is on plaintext and as fast as normal inference[1]. For example, Ohrimenko et al. (2016) demonstrated decision tree and linear model inference in SGX with small overhead. The Nervana's HE-Transformer (nGraph-HE) library even combined SGX with partial HE for neural nets [11]. More recently, Slalom (2019) proposed splitting heavy linear layers to run on an untrusted GPU while sensitive non-linear parts run in SGX, achieving significant speed-ups.

Advantages: The primary advantage is near-native performance and low latency, since no cryptographic operations are performed on the critical path of inference (data is decrypted inside the enclave and processed normally). It requires minimal changes to model code. Unlike pure cryptographic approaches, TEEs do not inflate computation or memory by large factors. They can also protect the model weights and the input at rest and during computation (everything inside the enclave is encrypted in memory). TEEs reduce the need for complex protocols—conceptually, it's like having a "secure black box" on the server.

Limitations: The trust model for TEEs is different: one must trust the hardware manufacturer (Intel/AMD etc.) that the enclave is secure and has no backdoors. In practice, TEEs have suffered from side-channel attacks (e.g., cache timing, speculative execution vulnerabilities like Spectre/Meltdown) that can leak sensitive data from enclaves. Additionally, SGX enclaves have limited memory (secure memory is often limited to a few hundred MB), which can be problematic for very large models or batch processing. The overhead of switching into the enclave and making syscalls can also impact performance for certain workloads. Another issue is that a determined client cannot verify the entire behavior of the server's enclave beyond what attestation covers; if the enclave program is supposed to not leak data, one must ensure it is written correctly (no inadvertent side-channels).

From a deployment perspective, TEEs require specific hardware support. Not all cloud providers or edge devices have SGX or similar enabled. There are also management challenges (enclave provisioning, attestation setup). Furthermore, if model privacy from the client is desired, the client should not receive the raw model output in plaintext either, otherwise they could use the output to infer model parameters in some cases. Typically, in TEE solutions the client trusts the enclave enough that model privacy is not a concern, or the enclave can post-process outputs (e.g., apply DP noise or only release final predictions).

A notable hybrid approach is **CHEX-MIX (2021)** by Natarajan et al., which combines HE with TEEs [19]. In their design, the model is loaded into an enclave on the cloud, but the enclave itself operates on homomorphically encrypted inputs from the client. The TEE ensures the model's integrity and helps with computations, but because the data stays encrypted, the client does not even need to trust the enclave with plaintext. This removes the need for client-side attestation of enclave code correctness. Their evaluation showed that this hybrid can reduce communication by 3× compared to pure multi-key HE offloading, while still leveraging hardware speed.

In summary, TEEs can be extremely useful for PPML inference, especially when certain trust assumptions are acceptable. For instance, enterprise use-cases where the hardware is owned by the data owner might prefer TEEs (no need for heavy cryptography). On the flip side, for cloud services where users do not fully trust the provider, a purely cryptographic solution might be preferred despite its cost.

3.4. Differential Privacy and Output Perturbation

Differential Privacy (DP) is a framework that provides guarantees against privacy leakage by adding randomness to computations. In the context of inference, DP techniques do not hide the input during computation, but rather ensure the output (or the model) does not reveal too much about any single training example. Classic DP training (like training with noise or the PATE framework [20]) aims to produce models that are "safe" to query [9]. However, DP can be applied at inference time in a couple of ways as a complementary approach:

- **Noising the output:** The server can add calibrated noise to the prediction or confidence scores before returning them to the client, so that the client cannot precisely infer certain details about the model or training data. This is rarely desirable for classification (it could cause wrong predictions), but in some probabilistic query settings it might be acceptable.
- **Limiting query access:** Using DP concepts, one can set a budget on how many queries a client can make, or add noise if too many queries are made, to protect against model inversion attacks [20].
- **Local DP for inputs:** In scenarios where the client does not fully trust the server and does not have cryptographic means, they could locally perturb their input (e.g., add noise) before sending it to the server model. This is essentially anonymization or randomization of input. For example, a user might add a small amount of noise to a feature vector so the exact data is obscured. The challenge is to maintain accuracy – local DP often incurs a substantial utility loss, especially for high-dimensional data. Recent work on text data has explored local DP by embedding text and adding noise to the embedding [21].

Advantages: DP techniques are generally lightweight and have mathematically rigorous guarantees. They do not require special hardware or complex cryptographic protocols. When training a model with DP (e.g., DP-SGD [22]), one can directly deploy it and users can query it normally, with the assurance that their queries cannot leak specific training data beyond ε-differential privacy. This addresses the second aspect of privacy (training data privacy).

Limitations: Differential privacy alone does not hide the actual input during inference from the server. So if a user simply queries a model on the cloud normally, DP doesn't stop the server from seeing that raw input. Thus, DP by itself is insufficient for input privacy (which is our main concern in PPML inference). It needs to be combined with cryptographic or trust-based methods to hide the input. Another limitation is that adding noise to outputs can degrade the utility of each individual inference. For classification tasks, a noisy output might mean the client gets an incorrect prediction with some probability, which is usually unacceptable in critical applications. Therefore, output perturbation is more applicable when returning aggregate statistics or when many identical queries are made such that averaging can recover the true answer.

In practice, DP is often used to complement cryptographic PPML: for example, one might train the model with DP so that even if the client or an adversary obtains many outputs (via the cryptographically secure protocol or via repeated queries to an enclave), they cannot exploit those outputs to infer training set details [9]. A holistic PPML system thus might use FHE/MPC/TEE for input privacy and DP for training data privacy. We include DP in this survey to emphasize that privacy is not only about encryption but also about what the model might implicitly reveal. Some open challenges involve unifying these: e.g., can we have an encrypted inference that is also differentially private with respect to the training data? Recent research suggests it's possible by injecting noise either in the model (during training) or in a post-processing step in the secure computation.

3.5. Federated and Split Inference Approaches

Federated learning is a technique for training, but the concept extends to inference: instead of sending the data to the model, send the model (or parts of it) to the data. In a naive approach, the server could send the entire model to the client, and the client runs inference locally. This is often termed **on-device inference**. It protects data privacy trivially (data never leaves), but completely gives away the model. In some cases (like open source models or where model IP is not a concern) this is acceptable. In others, this is undesirable. Federated inference per se is not commonly referenced, but on-device inference is effectively that. Many companies prioritize on-device AI to avoid sending user data to the cloud [23].

A compromise is **split inference** (also known as split learning). Here, the model is divided into two parts: the first few layers run on the client, and the remaining layers run on the server. The client sends the intermediate activation (sometimes called the cut layer representation) to the server, which continues the forward pass and returns the final output. This way, raw data is not sent to the server, only an intermediate feature vector. This vector hopefully is less revealing than the raw input, although research shows it can still leak information if the model is not carefully designed [21]. Some works inject noise into the intermediate activations to provide privacy (applying local DP at the cut layer). The recent SplitNN frameworks and the *Split-and-Denoise (SnD)* method [21], [24] follow this pattern. The client does a small part of the computation (affordable even on mobile), then noised activations are sent to server. The server may apply a denoising model or error-correction code to mitigate the impact of the added noise and then complete the inference. This approach can significantly reduce the information content of what the server sees (since noise is added and only partial features are shared).

Advantages: Split inference reduces communication size relative to sending raw data (in cases where feature representations are lower-dimensional than input, e.g., images). It also balances computational load. More importantly, from a privacy angle, it can hide some sensitive raw patterns (like exact pixel values) and with added noise it can provide a formal privacy guarantee in the local differential privacy sense [21]. Unlike full cryptographic protocols, the overhead here is modest (some noise addition and possibly a small local model on the client side). It doesn't require special hardware beyond the ability to run a truncated model on the client side.

Limitations: The intermediate activations can still leak a lot. The server could potentially train a model inversion attack to reconstruct the input from the activation. Studies have shown that without noise, it's often possible to partially recover original images from layer activations of a vision model. Adding noise helps but then accuracy drops, requiring careful balance. Another limitation is that the client must run part of the model, which might be an issue for very lightweight client devices if the model part is not extremely small. Also, split inference is not a universal privacy solution: it addresses input privacy to an extent, but the model is still partially revealed (the client knows the first part of the model's architecture and weights). Some proposals mitigate model leakage by only sharing the activations (not the weights) – the server never sends the first-layer weights to the client, it only receives activations. This keeps the model architecture known but weights hidden.

In scenarios like LLMs (large language models), model sizes are huge, so we cannot ship them to clients. But researchers have explored sending a small adapter or doing prompt encryption. One example is to use BERT with an encoding such that the server sees only an encoded prompt which it cannot decode, then returns an encoded answer [25].

In summary, alternative approaches like federated on-device inference and split learning provide lighter-weight privacy but with weaker guarantees compared to cryptography. They might be suitable when some leakage is tolerable or when performance is paramount. They can also be combined with cryptographic methods: e.g., one could secret-share or encrypt the intermediate activations instead of sending them raw, combining split learning with MPC.

To summarize the various privacy-preserving inference approaches discussed in this paper, Table 1 presents a comparative analysis based on key criteria such as privacy guarantees, efficiency, communication overhead, scalability, and security assumptions.

4. Use Cases and Recommendations

We illustrate the practical relevance of PPML inference through specific cases in education policy, medical diagnostics, and finance, along with guidance on choosing suitable techniques.

Education Policy Analysis: Estonia faced high dropout rates (43%) among IT students in 2012, prompting an investigation into the impact of employment on graduation [26]. The Estonian Association of Information and Communication Technology wanted to mine education and tax records to see if there was a correlation. However, privacy legislation prevented direct data sharing between the Ministry of Education and the Tax Board. Traditional anonymization methods like k-anonymity risked significant analytical quality loss due to unique student profiles. Secure Multi-Party Computation (MPC), implemented via Cybernetica's Sharemind framework [27], provided a solution. The data analysis was performed as a three-party computation involving servers representing the Estonian Information System's Authority, the Ministry of Finance, and Cybernetica. This enabled joint analysis without exposing raw data. The study, reported in [28], found no correlation between working during studies and failure to graduate on time but revealed that higher education correlated with increased income. MPC effectively handled distributed data ownership and maintained

Table 1: Comparison of Privacy-Preserving ML Inference Approaches

Approach	Privacy Guarantee	Efficiency	Communication Cost	Scalability	Security Assumptions	Limitations
Homomorphic Encryption (HE)	Strong (Mathematical)	Low (High Computation Cost)	Low (Minimal Interaction)	Limited (Slow for Deep Models)	No Trusted Party	High Latency, Large Ciphertext Size
Secure Multi-Party Computation (MPC)	Strong (Mathematical)	Medium (Depends on Protocol)	High (Multiple Rounds)	Moderate	Honest Majority Required	High Communication Overhead
Trusted Execution Environments (TEEs)	Hardware-Based	High (Near-Native Speed)	Low (Minimal Communication)	High (Supports Large Models)	Trusted Hardware Vendor	Side-Channels, Trust Vendor
Differential Privacy (DP)	Statistical (Training Data Only)	High	Low (Minimal Overhead)	High	Trusted Aggregator Needed	Adds Noise, Utility Loss
Federated/Split Learning	Partial (Limited Input Privacy)	High (On-Device Computation)	Medium (Intermediate Activations)	High	Secure Aggregator Needed	Intermediate Activations May Leak

confidentiality for this statistical analysis task, which shares principles with ML model application.

Medical Diagnostics: Hospitals and healthcare providers increasingly leverage cloud-based AI for medical image diagnosis (e.g., analyzing X-rays or MRIs) or predicting patient outcomes. Such data is highly sensitive and protected by regulations like HIPAA in the US and the GDPR in Europe [3]. For robust privacy of patient data sent to a third-party model provider, Homomorphic Encryption or MPC are strong candidates. HE can be beneficial when minimizing communication rounds is critical, while MPC (especially hybrid protocols) might offer better performance for complex models. If a hospital trusts its internal hardware infrastructure and the hardware vendor, TEEs can provide high performance with strong protection within the enclave. For real-time, device-based prescreening applications (e.g., on a portable scanner), Split Inference can be useful. This reduces data exposure by processing initial layers locally on the device and sending only intermediate, potentially less sensitive, feature representations to a server for final analysis. Adding Local Differential Privacy to these intermediate activations can further enhance privacy, though with a potential trade-off in diagnostic accuracy that must be carefully managed.

Finance: Financial institutions use ML for credit risk assessment, fraud detection, and algorithmic trading. These applications involve sensitive customer financial data and often proprietary models developed by the institutions. For cloud-based credit risk assessment using third-party models, strong privacy for both client data and the model IP can be achieved using HE or MPC. MPC is particularly suitable for scenarios requiring collaboration between multiple financial institutions (e.g., for fraud detection across banks) without direct data sharing. In applications like high-frequency trading or real-time fraud detection where minimal latency is paramount, TEEs can offer the necessary performance. However, the security risks associated with TEEs (e.g., side-channels) must be critically assessed and mitigated. Integrating Differential Privacy, for instance, by adding noise to query outputs or limiting query rates, can add a layer of protection against inference attacks aiming to reconstruct sensitive training data or model parameters from repeated interactions, thereby enhancing overall system security.

In conclusion, selecting appropriate PPML inference techniques depends critically on the specific application's privacy requirements (data sensitivity, model IP), performance needs (latency, throughput), communication constraints, and the trust model (trust in third parties, hardware vendors, or colluding entities). Highly regulated sectors like healthcare and finance typically benefit most from strong cryptographic guarantees offered by HE and MPC, or the hardware-enforced isolation of TEEs. Lighter-weight approaches like split inference and DP can effectively complement these methods or serve as primary solutions in scenarios with less stringent privacy requirements or where performance is the overriding concern.

5. Implementations and Performance

Each approach above has seen continuous improvements, and several frameworks exist to implement PPML inference in practice. We highlight some notable systems and results.

5.1. Cryptographic Frameworks and Libraries

There are mature libraries for homomorphic encryption, such as Microsoft SEAL (used in CryptoNets and many follow-ups) [29], OpenFHE [30], HElib [31], and TFHE [32]. SEAL, for instance, implements BFV and CKKS schemes and has been used to evaluate CNNs on encrypted data (like SEALion for ImageNet classifications with approximations). These libraries provide basic operations; on top of them, researchers have built toolkits like nGraph-HE (Intel) [33] which can take a neural network description and execute parts homomorphically. Zama's Concrete [34] library (with the CGGI FHE scheme) is another modern tool that aims to ease implementing private inference with FHE.

For MPC, frameworks include:

- **ABY / ABY3** [35]: A C++ framework that supports mixed protocols (Arithmetic, Boolean, Yao sharing) for 2-party and 3-party computation. Many PPML research prototypes use ABY to implement neural net layers with a mix of sharings.
- **CrypTen**: A research-oriented library by Facebook for secure ML, built on PyTorch, which currently supports primarily MPC with additive sharing [15]. CrypTen allows developers to write PyTorch-like code and execute it under the hood with MPC.
- **TF Encrypted** [36]: A library that integrates secure computation into TensorFlow. It was one of the earlier industry-led libraries to make PPML accessible, supporting both MPC and hybrid approaches.
- **EMP-toolkit [37] and Obliv-C [38]**: Low-level libraries to implement Yao's GC and OT efficiently, which have been used for custom protocols in papers like MiniONN and XONN.
- **SecureDFL [39]**: A recent line of research on using secret sharing for federated learning can also be adapted for inference (though these typically focus on training).

These frameworks have demonstrated various state-of-the-art results. For example, using CrypTen—a secure multi-party computation library developed by Facebook Research—a simple CNN on CIFAR-10 has been shown to run with a latency of a few seconds in a 2-party semi-honest setting (with some accuracy loss due to fixed-point quantization) [15]. In addition, frameworks such as Delphi have demonstrated that even larger models (e.g., comparable to VGG-16) can be executed in a 3-party setting with inference times under one second on powerful hardware [4].

The **Delphi** system [4] combined ideas from Gazelle and DeepSecure and introduced a neural architecture search to simplify the network (reduce nonlinear operations) for faster private inference. It achieved 22× lower online latency than prior art for certain ImageNet-scale networks by moving most cryptographic cost to offline preprocessing. This indicates a trend: tailoring the model (through quantization, layer replacement, etc.) can significantly improve performance. Techniques like replacing ReLU with low-degree polynomial (SecureML [12] did quadratic) or using special activations that are MPC-friendly (e.g., truncation instead of ReLU [11], [12]) have been explored.

5.2. Performance

Performance in Privacy-Preserving Machine Learning is typically evaluated based on latency (time per inference), throughput, and communication volume between the parties. The scale and architecture of the ML model significantly impact these metrics.

For example, for classic computer vision models:

- A pure Homomorphic Encryption (HE) approach would have negligible communication (just encrypted input and output) but incurs substantial latency, often orders of magnitude higher than plaintext execution, due to the computational complexity of operations on ciphertexts [2], [9].
- Hybrid or Secure Multi-Party Computation (MPC) frameworks offer better performance. Gazelle [10] demonstrated secure inference for a small CNN (CIFAR10) in under 0.5 seconds with a few MB of communication. However, scaling these methods to larger models like ResNet-50 poses significant challenges. Delphi [4] reported a baseline for ResNet-32 secure inference taking approximately 82 seconds and 560MB of communication without their optimizations, highlighting the performance cost of larger models.
- The optimized Delphi reduced ResNet-32 inference to a few seconds with tens of MB of communication, largely by moving computationally expensive steps to an offline preprocessing phase [4].

Scaling these PPML techniques to Large Language Models (LLMs) with billions of parameters introduces even greater performance hurdles. LLM inference involves massive matrix multiplications and complex non-linear functions across many layers. Recent research specifically addresses secure inference for these large transformer-based models.

- The BumbleBee framework [40] proposes an optimized two-party secure inference approach for large transformers. It focuses on significantly reducing communication costs for matrix multiplication (claiming 80-90% reduction over prior HE-based OLT methods based on microbenchmarks) and optimizing non-linear activation functions. For a BERT-base model (128 input tokens), BumbleBee achieved an end-to-end inference time of around 2.55 minutes with 6.4GB of communication in a LAN setting, showing notable improvements in both metrics compared to earlier 2PC frameworks like Iron and BOLT for BERT models [40]. For LLaMA-7B, BumbleBee reported an inference time of approximately 13.87 minutes for 8 tokens with 5.64GB communication [40].
- Another recent work [41] focuses on secure inference for fine-tuned LLMs leveraging the architecture of Parameter-Efficient Fine-Tuning (PEFT) techniques like LoRA. Their approach splits the model into a public base model (processed client-side in plaintext) and private LoRA matrices (processed server-side using Fully Homomorphic Encryption). This division significantly reduces the computation performed in the HE domain. For a ChatGLM2-6B model fine-tuned with LoRA, they report an inference efficiency of 1.61 seconds *per token* for sequences longer than 1000 tokens. This result demonstrates a significant performance leap compared to some prior LLM secure inference methods cited in their paper [41].
- Microbenchmarks within frameworks like BumbleBee and findings from other recent works [1], [11] consistently highlight that matrix multiplication and the evaluation of non-linear activation functions (like GeLU and Softmax) remain the most computationally expensive operations in secure transformer inference, underscoring the importance of optimizing these core components.

Despite progress, challenges remain. Results from an AWS engineering blog [42] demonstrate that even using CKKS-based FHE for a simpler model like logistic regression on the IRIS dataset incurs significant computational overhead (processing 140 samples took 60 seconds end-to-end). This linear scaling with input size and model complexity highlights why pure FHE can still be impractical for very large and complex models in interactive settings, driving the need for hybrid methods or optimized MPC.

Nevertheless, research continues to push boundaries. Recent work demonstrates that GPU acceleration can reduce FHE inference latency by an order of magnitude for certain models [43]. On the MPC side, efforts like PriViT [44] aim to adapt techniques for complex models such as Vision Transformers by minimizing non-linear operations. The advancements demonstrated in BumbleBee and the FHE-LoRA approach for LLMs show promising paths towards more practical and efficient privacy-preserving inference for even the largest and most complex AI models.

5.3. Open Challenges and Ongoing Work

Despite much progress, several challenges remain for PPML inference:

Scalability to Large Models: While small and medium models have been demonstrated, state-of-the-art large models (e.g., transformer-based language models with billions of parameters) are currently infeasible to run with full cryptographic privacy. The communication and compute would be enormous. One direction to handle this is model compression or distillation to get smaller models that approximate the large ones, then run those privately [45]. Another is using privacy-friendly models (Delphi's approach for CNNs, PriViT for transformers [44]).

Reducing Interaction Rounds: Protocols with many rounds (like one round per layer) suffer in high latency settings. Techniques to pack more computation into a single round (at the cost of more client compute or using somewhat homomorphic encryption in intermediate steps) are being explored. For example, one could evaluate multiple layers in one go by having the client provide some auxiliary info or by using leveled HE across a couple of layers before needing to communicate. Achieving near one-round (two-message) protocols for entire deep networks remains an open problem. Some fully HE approaches are one-round but then the challenge is performance.

Robustness and Side-Channels: If using TEEs, one must consider side-channel attacks. Recent research attempts to combine masking or oblivious algorithms inside enclaves to reduce leakage, but at performance cost. For cryptographic protocols, side-channels can also appear if implementations aren't careful (e.g., timing differences in operations could leak secret information if one party is malicious). Achieving malicious security (where parties can deviate arbitrarily) typically doubles the overhead or worse (because you need to add zero-knowledge proofs or MACs on every operation). Nearly all current systems target semi-honest adversaries for efficiency [7]. Bridging that gap is important for real-world deployment.

Privacy-Utility Trade-offs: There is a need for systematic ways to trade a little privacy for significant gains in performance when appropriate. For instance, allowing the client to learn some minimal additional information that isn't too sensitive might simplify a protocol. Conversely, adding a little noise might drastically speed things up by allowing use of quantized approximations. Understanding which minor relaxations of the threat model could yield major efficiency gains is an area of active exploration. An example is the "rational adversary" model used in CHEX-MIX [19], which assumes the model provider is rational (cares about output integrity) so they remove the need for client-side attestation by using HE, thereby simplifying trust issues.

Integration and Usability: From a machine learning engineer perspective, PPML inference is still not plug-and-play. Each new model might require custom tuning to run efficiently. Developing high-level compilers that can take an arbitrary TensorFlow/PyTorch model and compile it into an efficient secure protocol (choosing automatically in which parts to use HE vs MPC vs etc.) is a challenging software engineering problem. Projects like *CrypTFlow* [16] and *EzPC* [46] are early steps in this direction, but more work is needed for wide adoption.

Multi-client and Batch Inference: In many real cases, a server will serve many clients. Fully HE solutions allow easy batching of different inputs (they are independent). MPC solutions can sometimes amortize cost if the server can batch process multiple queries together or reuse some precomputation. However, if multiple clients want to jointly get some function of all their inputs (not the usual scenario for inference, more for aggregation), then multi-party protocols are needed. In inference-as-a-service, usually each query is separate. One question is: can a server amortize work across queries to hide cost? Some techniques like using the same garbled circuit for multiple clients (if model fixed) exist, but then security between clients must be ensured (a malicious client might use that to invert another's output, etc.).

Finally, an open challenge is **standardization of evaluation**. Different works often use different models, datasets, hardware; some report online time excluding preprocessing, others include it. There is a lack of consensus on benchmarks (beyond maybe MNIST and CIFAR). The healthcare survey [3] noted that most PPML inference studies use their own validation setups and not a common benchmark, making it hard to compare. A community effort to establish reference tasks (e.g., a private version of ImageNet inference challenge) would be valuable to measure progress.

6. Big Picture and Future Directions

Privacy-preserving ML inference is an interdisciplinary endeavor at the intersection of machine learning and cryptography. Figure 3 conceptually places the approaches on a spectrum of privacy vs. efficiency. On one end, we have pure on-device inference (max efficiency, minimum server trust needed); on the other end, we have FHE (max privacy, huge cost). In between lie hybrid MPC and enclave solutions offering different balances. The big picture is that no single approach is universally best – it depends on the threat scenario:

- If the primary concern is data confidentiality from an honest but curious service provider, and some latency can be tolerated, then HE or two-party MPC is appropriate.
- If low latency is required and the user is willing to trust hardware, TEE-based inference may be preferred.
- If model IP is not a concern but data is, one could even send the model to the client (which some companies do for premium users in federated settings).
- If both sides are equally concerned (e.g., two hospitals want to run one's model on the other's data), then multi-party or dual-enclave setups might be used.

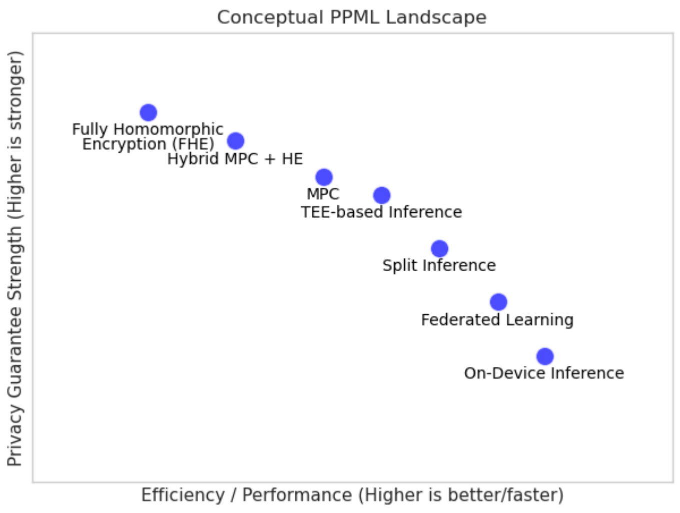

Figure 3: PPML methods balance privacy and efficiency. On-device inference maximizes speed but lacks model protection, while FHE ensures privacy at high cost. Hybrid approaches (MPC, TEE, split and federated learning) offer varying trade-offs based on security needs.

6.1. Industry Approaches: Vendor-Controlled Secure Infrastructure

Beyond academic research, major technology companies are developing and deploying large-scale systems that aim to provide strong privacy for ML services, often blending hardware security, operational policies, and transparency mechanisms. A notable example is Apple's Private Cloud Compute (PCC) [47], [48].

PCC is designed to process complex AI queries (e.g., for advanced Siri features) that are too demanding for on-device execution, without Apple retaining user data. Instead of relying on cryptographic blindness during computation (like HE or MPC for the main processing), PCC's privacy model centers on strong infrastructure security and operational guarantees:

- **Stateless Servers:** Data sent for processing is not stored persistently on PCC servers after the query is handled.
- **Cryptographic Unlinkability:** Requests processed on PCC servers are designed not to be tied to a user's Apple ID or other personally identifiable information visible to the server infrastructure during processing.
- **Minimal Software Stack:** PCC servers run a purpose-built, minimal operating system and software stack to reduce the attack surface.
- **Access Controls:** Technical measures are implemented to prevent unauthorized access, including by Apple employees, to user data during processing.

To build trust in these claims, Apple employs a verifiability mechanism. They make the software images that run on PCC servers publicly available for independent security researchers to download and audit. The goal of this audit is to allow experts to verify that the code implements the stated privacy promises (e.g., statelessness, no logging of sensitive data). This approach contrasts with per-computation cryptographic proofs (like ZKPs) and instead relies on transparency and expert validation of the system's software design and operational integrity.

Conceptually, such vendor-controlled secure infrastructure models like PCC offer high efficiency (computation occurs on plaintext data within the secured environment, comparable to standard cloud inference) while aiming for strong privacy guarantees. The privacy level is stronger than typical cloud processing but relies on trust in the vendor's operational security, the robustness of their hardware/software isolation, and the thoroughness of the public audit process. This positions such solutions near TEEs in terms of efficiency, with a privacy model that heavily depends on vendor commitments and verifiable transparency. This trend highlights how industry is tackling the PPML challenge for demanding, large-scale ML tasks by creating custom ecosystems.

6.2. Alternative Approaches:

Beyond what we discussed, there are a few niche methods:

- *Obfuscation*: Attempts to obfuscate the model itself such that it can be executed without revealing inner workings. General program obfuscation is impractical, but for specific circuits there might be methods (this is largely theoretical at this point) [49].
- *Zero-Knowledge Proofs for inference*: A client could prove to a server that they know an input that produces a given output, without revealing the input. This way the server doesn't even run the model; the client does and proves correctness. Recent advances in succinct ZK proofs (like zkSNARKs) could potentially make this viable for small models. However, applying it to deep networks is challenging due to the need to prove knowledge of a large computation [50].
- *Specialized neural architectures*: Using binary neural networks (weights and activations are 0/1) so that inference can be done with simple XOR operations under MPC (as in XONN [51]) or efficient boolean circuits [52]. Alternatively, use can be made of quantized ReLUs that are effectively "max(0,x)" which can be done by bit tricks in MPC more easily. These changes can improve speed but may reduce accuracy, so careful design is needed.

The key **open challenges** moving forward include:

1. Efficiency at Scale: How to support models with millions of parameters and high-dimensional data within acceptable time. This may involve algorithmic breakthroughs (better compression of circuits, asymptotically faster HE operations) or hardware (accelerators as proposed by Cheetah) [9].
2. Robust Security Guarantees: Achieving protection against malicious servers/clients so that the system can be robust in adversarial settings. Also, quantifying information leakage (if any) when using hybrid schemes or if protocols abort.
3. Composable Privacy: If one wants both training data privacy and inference input privacy, combining DP-trained models with cryptographic inference is promising. But analyzing the combined leakage is non-trivial. Ensuring that no vector of outputs over many queries can breach privacy requires consideration of both cryptographic leakage (which is zero in ideal case) and statistical leakage (bounded by DP ε).
4. Regulatory Acceptance: For PPML methods to be adopted in regulated industries (finance, healthcare), there needs to be trust in the technology. This may involve standardizing security claims, certification of libraries (perhaps via formal verifi-

cation of protocols), and education of stakeholders about how PPML protects data [53].

5. Interoperability: In many scenarios, the party holding the data and the party holding the model might use different platforms or frameworks. Interoperable protocols (maybe using a common intermediate representation for models) would ease deployment.

There is optimism in the community that with continuing advances, PPML inference will become practical for an increasing set of use cases. The fact that major cloud providers (Amazon, Microsoft, Google) are investing in homomorphic encryption research [42] and releasing related tools indicates a push towards making this technology usable. Meanwhile, academic collaborations like OpenMined are fostering open-source efforts for PPML. A collaborative benchmark suite and clearer consensus on requirements (as noted in healthcare domain survey [3]) will further accelerate progress by allowing apples-to-apples comparisons and highlighting bottlenecks.

7. Conclusion

Privacy-preserving machine learning inference has evolved from a theoretical possibility to working prototypes and frameworks that can handle non-trivial models and datasets. Through a combination of cryptographic protocols (HE, MPC), hardware-based solutions (TEEs), and privacy-aware algorithms (DP, split learning), we now have a toolkit of approaches that can be tailored to specific privacy needs and performance constraints. The state-of-the-art shows that for small to medium models, one can achieve seconds or sub-second latency with full data confidentiality [10], and even approach real-time inference with specialized hardware [7]. However, challenges remain in scaling to extremely complex models and achieving robust security against all adversaries.

Key open problems include improving efficiency (perhaps via model co-design or hardware acceleration) and standardizing methods for broader adoption. We also need more research into measuring and mitigating any subtle information leakage (e.g., through side-channels or from output distributions). Encouragingly, trends such as integrating PPML into cloud services, and combining techniques (like CHEX-MIX's HE + TEE, or MPC + DP training), alongside industry developing large-scale privacy-focused infrastructures like Apple's PCC, suggest that practical deployments are on the horizon. For example, Apple has deployed private lookup services using HE for certain features [23], and Amazon's prototype with FHE shows even logistic regression can be served with reasonable throughput on encrypted data [42].

In conclusion, PPML inference is a rapidly maturing field that sits at a crucial intersection of machine learning and privacy. As models continue to permeate sensitive applications (medical diagnosis, personal assistants, financial forecasting), the demand for techniques reviewed in this paper will only grow. By harnessing the state-of-the-art methods outlined and addressing remaining challenges, we move closer to a future where users can benefit from powerful ML services without sacrificing their privacy, and companies can deploy models without fear of leaking proprietary data. Achieving this promises to unlock data sharing and collaboration opportunities that are currently hindered by privacy concerns [3], truly enabling machine learning to reach its full potential in a privacy-conscious world.

References

[1] J. Mo, K. Garimella, N. Neda, A. Ebel, and B. Reagen, "Towards Fast and Scalable Private Inference," in *Proceedings of the 20th ACM International Conference on Computing Frontiers*, Bologna Italy: ACM, May 9, 2023, pp. 322–328, ISBN: 9798400701405. DOI: 10.1145/3587135.3592169. [Online]. Available: https://dl.acm.org/doi/10.1145/3587135.3592169 (visited on 03/06/2025).

[2] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, NY, USA: JMLR.org, Jun. 19, 2016, pp. 201–210.

[3] A. Guerra-Manzanares, L. J. L. Lopez, M. Maniatakos, and F. E. Shamout, "Privacy-preserving machine learning for healthcare: Open challenges and future perspectives," in vol. 13932, 2023, pp. 25–40. DOI: 10.1007/978-3-031-39539-0_3. arXiv: 2303.15563 [cs]. [Online]. Available: http://arxiv.org/abs/2303.15563 (visited on 03/05/2025).

[4] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A Cryptographic Inference Service for Neural Networks," presented at the 29th USENIX Security Symposium (USENIX Security 20), 2020, pp. 2505–2522, ISBN: 978-1-939133-17-5. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/mishra (visited on 03/06/2025).

[5] C. Zhang and S. Li. "State-of-the-Art Approaches to Enhancing Privacy Preservation of Machine Learn-

ing Datasets: A Survey.” arXiv: 2404.16847 [cs]. (Jan. 28, 2025), [Online]. Available: http://arxiv.org/abs/2404.16847 (visited on 03/06/2025), pre-published.

[6] K. Chellapilla, S. Puri, and P. Simard, “High Performance Convolutional Neural Networks for Document Processing,” presented at the Tenth International Workshop on Frontiers in Handwriting Recognition, Suvisoft, Oct. 23, 2006. [Online]. Available: https://inria.hal.science/inria-00112631 (visited on 03/10/2025).

[7] B. Reagen, W. Choi, Y. Ko, *et al.* “Cheetah: Optimizing and Accelerating Homomorphic Encryption for Private Inference.” arXiv: 2006.00505 [cs]. (Oct. 8, 2020), [Online]. Available: http://arxiv.org/abs/2006.00505 (visited on 03/06/2025), pre-published.

[8] R. Gennaro, C. Gentry, and B. Parno. “Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers.” (2009), [Online]. Available: https://eprint.iacr.org/2009/547 (visited on 04/01/2025), pre-published.

[9] R. Xu, N. Baracaldo, and J. Joshi. “Privacy-Preserving Machine Learning: Methods, Challenges and Directions.” arXiv: 2108.04417 [cs]. (Sep. 22, 2021), [Online]. Available: http://arxiv.org/abs/2108.04417 (visited on 03/05/2025), pre-published.

[10] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. “Gazelle: A Low Latency Framework for Secure Neural Network Inference.” arXiv: 1801.05507 [cs]. (Jan. 16, 2018), [Online]. Available: http://arxiv.org/abs/1801.05507 (visited on 03/06/2025), pre-published.

[11] L. Zhou, Z. Wang, H. Cui, Q. Song, and Y. Yu. “Bicoptor: Two-round Secure Three-party Non-linear Computation without Preprocessing for Privacy-preserving Machine Learning.” arXiv: 2210.01988 [cs]. (Apr. 19, 2024), [Online]. Available: http://arxiv.org/abs/2210.01988 (visited on 03/06/2025), pre-published.

[12] P. Mohassel and Y. Zhang. “SecureML: A System for Scalable Privacy-Preserving Machine Learning.” (2017), [Online]. Available: https://eprint.iacr.org/2017/396 (visited on 03/12/2025), pre-published.

[13] A. C. Yao, “Protocols for secure computations,” in *23rd annual symposium on foundations of computer science (sfcs 1982)*, IEEE, 1982, pp. 160–164.

[14] M. O. Rabin, “How to exchange secrets with oblivious transfer,” *Cryptology ePrint Archive*, 2005.

[15] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten. “CrypTen: Secure Multi-Party Computation Meets Machine Learning.” arXiv: 2109.00984 [cs]. (Sep. 15, 2022), [Online]. Available: http://arxiv.org/abs/2109.00984 (visited on 03/06/2025), pre-published.

[16] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma. “CrypTFlow: Secure TensorFlow Inference.” (2019), [Online]. Available: https://eprint.iacr.org/2019/1049 (visited on 03/06/2025), pre-published.

[17] J. Liu, M. Juuti, Y. Lu, and N. Asokan, “Oblivious neural network predictions via minionn transformations,” presented at the Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 619–631.

[18] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar. “Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications.” arXiv: 1801.03239 [cs]. (Jan. 10, 2018), [Online]. Available: http://arxiv.org/abs/1801.03239 (visited on 03/06/2025), pre-published.

[19] D. Natarajan, A. Loveless, W. Dai, and R. Dreslinski. “CHEX-MIX: Combining Homomorphic Encryption with Trusted Execution Environments for Two-party Oblivious Inference in the Cloud.” (2021), [Online]. Available: https://eprint.iacr.org/2021/1603 (visited on 03/06/2025), pre-published.

[20] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow, and K. Talwar. “Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data.” arXiv: 1610.05755 [stat]. (Mar. 3, 2017), [Online]. Available: http://arxiv.org/abs/1610.05755 (visited on 03/05/2025), pre-published.

[21] P. Mai, R. Yan, Z. Huang, Y. Yang, and Y. Pang. “Split-and-Denoise: Protect large language model inference with local differential privacy.” arXiv: 2310.09130 [cs]. (Aug. 27, 2024), [Online]. Available: http://arxiv.org/abs/2310.09130 (visited on 03/06/2025), pre-published.

[22] X. Li, F. Tramèr, P. Liang, and T. Hashimoto. “Large Language Models Can Be Strong Differentially Private Learners.” arXiv: 2110.05679 [cs]. (Nov. 10, 2022), [Online]. Available: http://arxiv.org/abs/2110.05679 (visited on 03/05/2025), pre-published.

[23] “Combining Machine Learning and Homomorphic Encryption in the Apple Ecosystem,” Apple Machine Learning Research. (Aug. 24, 2024), [Online]. Available: https://machinelearning.apple.com/

research / homomorphic - encryption (visited on 03/06/2025).

[24] X. Yang, J. Sun, Y. Yao, J. Xie, and C. Wang. "Differentially Private Label Protection in Split Learning." arXiv: 2203.02073 [cs]. (Mar. 4, 2022), [Online]. Available: http://arxiv.org/abs/2203.02073 (visited on 03/11/2025), pre-published.

[25] X. Liu and Z. Liu. "LLMs Can Understand Encrypted Prompt: Towards Privacy-Computing Friendly Transformers." arXiv: 2305.18396 [cs]. (Dec. 15, 2023), [Online]. Available: http://arxiv.org/abs/2305.18396 (visited on 03/11/2025), pre-published.

[26] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Foundations and Trends® in Privacy and Security*, vol. 2, no. 2–3, pp. 70–246, 2018, ISSN: 2474-1558.

[27] D. Bogdanov, S. Laur, and J. Willemson. "Sharemind: A framework for fast privacy-preserving computations." (2008), [Online]. Available: https://eprint.iacr.org/2008/289 (visited on 03/12/2025), pre-published.

[28] "Track Big Data For Governments and Education | Sharemind." (Oct. 26, 2015), [Online]. Available: https://sharemind.cyber.ee/big-data-analytics-protection/ (visited on 03/12/2025).

[29] *Microsoft/SEAL*, Microsoft, Mar. 5, 2025. [Online]. Available: https://github.com/microsoft/SEAL (visited on 03/06/2025).

[30] A. Al Badawi, J. Bates, F. Bergamaschi, *et al.*, "Openfhe: Open-source fully homomorphic encryption library," in *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, ser. WAHC'22, Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 53–63. DOI: 10.1145/3560827.3563379. [Online]. Available: https://doi.org/10.1145/3560827.3563379.

[31] *Homenc/HElib*, homenc, Mar. 11, 2025. [Online]. Available: https://github.com/homenc/HElib (visited on 03/12/2025).

[32] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, *TFHE: Fast fully homomorphic encryption library*, https://tfhe.github.io/tfhe/, 2016.

[33] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, ser. CF '19, New York, NY, USA: Association for Computing Machinery, Apr. 30, 2019, pp. 3–13, ISBN: 978-1-4503-6685-4. DOI: 10.1145/3310273.3323047. [Online]. Available: https://doi.org/10.1145/3310273.3323047 (visited on 03/12/2025).

[34] Zama, *Concrete: TFHE Compiler that converts python programs into FHE equivalent*, https://github.com/zama-ai/concrete, 2022.

[35] P. Rindal, *The ABY3 Framework for Machine Learning and Database Operations.* https://github.com/ladnir/aby3.

[36] *Tf-encrypted/tf-encrypted*, TF Encrypted, Mar. 2, 2025. [Online]. Available: https://github.com/tf-encrypted/tf-encrypted (visited on 03/12/2025).

[37] X. Wang, A. J. Malozemoff, and J. Katz, *EMP-toolkit: Efficient MultiParty computation toolkit*, https://github.com/emp-toolkit, 2016.

[38] "Obliv-C." (), [Online]. Available: https://oblivc.org/ (visited on 03/12/2025).

[39] B. Jeon, S. M. Ferdous, M. R. Rahman, and A. Walid. "Privacy-preserving Decentralized Aggregation for Federated Learning." arXiv: 2012.07183 [cs]. (Dec. 28, 2020), [Online]. Available: http://arxiv.org/abs/2012.07183 (visited on 03/12/2025), pre-published.

[40] W.-j. Lu, Z. Huang, Z. Gu, *et al.*, *BumbleBee: Secure Two-party Inference Framework for Large Transformers*, 2023. (visited on 06/17/2025).

[41] Z. Ruoyan, Z. Zhongxiang, and B. Wankang, *Practical Secure Inference Algorithm for Fine-tuned Large Language Model Based on Fully Homomorphic Encryption*, Jan. 2025. DOI: 10.48550/arXiv.2501.01672. arXiv: 2501.01672 [cs]. (visited on 06/17/2025).

[42] "Enable fully homomorphic encryption with Amazon SageMaker endpoints for secure, real-time inferencing | AWS Machine Learning Blog." (Mar. 23, 2023), [Online]. Available: https://aws.amazon.com/blogs/machine-learning/enable-fully-homomorphic-encryption-with-amazon-sagemaker-endpoints-for-secure-real-time-inferencing/ (visited on 03/06/2025).

[43] C. Gouert and N. G. Tsoutsos. "Data Privacy Made Easy: Enhancing Applications with Homomorphic Encryption." (2024), [Online]. Available: https://eprint.iacr.org/2024/118 (visited on 03/06/2025), pre-published.

[44] N. Dhyani, J. Mo, M. Cho, *et al.* "PriViT: Vision Transformers for Fast Private Inference." arXiv: 2310.04604 [cs]. (Oct. 6, 2023), [Online]. Available: http://arxiv.org/abs/2310.04604 (visited on 03/06/2025), pre-published.

[45] A. Polino, R. Pascanu, and D. Alistarh. "Model compression via distillation and quantization." arXiv: 1802.05668 [cs]. (Feb. 15, 2018), [Online]. Available: http://arxiv.org/abs/1802.05668 (visited on 03/06/2025), pre-published.

[46] N. Chandran, D. Gupta, A. Rastogi, R. Sharma, and S. Tripathi, "Ezpc: Programmable and efficient secure two-party computation for machine learning," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, 2019, pp. 496–511.

[47] "Blog - Private Cloud Compute: A new frontier for AI privacy in the cloud - Apple Security Research," Blog - Private Cloud Compute: A new frontier for AI privacy in the cloud - Apple Security Research. (Oct. 6, 2024), [Online]. Available: https://security.apple.com/blog/private-cloud-compute/ (visited on 04/08/2025).

[48] Apple, director, *WWDC 2024 — June 10 | Apple*, Jun. 10, 2024. [Online]. Available: https://www.youtube.com/watch?v=RXeOiIDNNek (visited on 04/08/2025).

[49] M. Zhou, X. Gao, J. Wu, *et al.*, "ModelObfuscator: Obfuscating Model Information to Protect Deployed ML-Based Systems," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2023, New York, NY, USA: Association for Computing Machinery, Jul. 13, 2023, pp. 1005–1017, ISBN: 9798400702211. DOI: 10.1145/3597926.3598113. [Online]. Available: https://doi.org/10.1145/3597926.3598113 (visited on 03/12/2025).

[50] H. Sun, J. Li, and H. Zhang. "zkLLM: Zero Knowledge Proofs for Large Language Models." arXiv: 2404.16109 [cs]. (Apr. 24, 2024), [Online]. Available: http://arxiv.org/abs/2404.16109 (visited on 03/12/2025), pre-published.

[51] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "{XONN}: {XNOR-based} Oblivious Deep Neural Network Inference," presented at the 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 1501–1518, ISBN: 978-1-939133-04-5. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/riazi (visited on 03/07/2025).

[52] M. Ball, B. Carmer, T. Malkin, M. Rosulek, and N. Schimanski. "Garbled Neural Networks are Practical." (2019), [Online]. Available: https://eprint.iacr.org/2019/338 (visited on 03/07/2025), pre-published.

[53] *Trustworthy AI*, in *Wikipedia*, Jan. 18, 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Trustworthy_AI&oldid=1270145996 (visited on 03/12/2025).