SoK: Verifiable Federated Learning

Francesco Bruschi[2], Marco Esposito[2], Tommaso Gagliardoni[1], and Andrea Rizzini[1,2]

[1] *Horizen Labs*
[2] *Politecnico di Milano*

December 20, 2025

Abstract

Federated Learning (FL) is an advancement in Machine Learning motivated by the need to preserve the privacy of the data used to train models. While it effectively addresses this issue, the multi-participant paradigm on which it is based introduces several challenges. Among these are the risks that participating entities may behave dishonestly and fail to perform their tasks correctly. Moreover, due to the distributed nature of the architecture, attacks such as Sybil and collusion are possible. Recently, with advances in Verifiable Computation (VC) and Zero-Knowledge Proofs (ZKP), researchers have begun exploring how to apply these technologies to Federated Learning aiming to mitigate such problems. In this Systematization of Knowledge, we analyze the first, very recent works that attempt to integrate verifiability features into classical FL tasks, comparing their approaches and highlighting what is achievable with the current state of VC methods.

1 Introduction

Machine Learning (ML) is a field of Artificial Intelligence that has seen tremendous growth over the past two decades and is now one of the most actively researched areas in computer science. ML enables the development of models that, starting from a dataset, can learn to perform specific tasks

such as prediction, object classification, question answering and many others. In order for a model to achieve high accuracy, the training dataset must contain high-quality data; otherwise, the resulting model will be unreliable, following the well-known paradigm "garbage in, garbage out".

In many practical scenarios, however, the training data is not only critical for the model accuracy, but also sensitive in nature since it may contain private or confidential information. In such cases, data owners may be unwilling to share their data with a central authority or third party. Furthermore, multiple related entities may consider collaborating to train a shared model while keeping their own data private. This is often the case in scenarios such as healthcare, where departments from different hospitals want to leverage their data to obtain a model that benefits all participants.

To address these issues, a new machine learning approach called Federated Learning (FL) was introduced by Google in 2016 [59]. The basic framework follows an iterative procedure, where in each round two main steps are performed: (1) each client trains a model locally, thereby keeping its dataset private; and (2) the locally trained models are sent to a central server (the *aggregator*), which aggregates them into a global model; this updated global model is then redistributed to the clients, allowing another iteration to begin until the model converges.

However, the basic FL setup exposes some limitations, in particular the model updates shared by the clients may leak private information about their local dataset and the central server may learn sensitive information or act maliciously [26, 90]. To address such concerns, various cryptographic and statistical tools have been proposed, leading to the concept of Privacy-Preserving Federated Learning (PPFL). These tools include Differential Privacy (DP), which adds noise to the updates to obfuscate individual contributions [40, 76]; Homomorphic Encryption (HE), which enables computations on encrypted data [33, 52]; and Multi Party Computation (MPC), which allows parties to collaboratively compute functions over inputs while keeping those inputs private [16].

Despite these enhancements, a fundamental problem is the lack of verifiability in the FL process. The training and aggregation steps often rely on the "honest-but-curious" assumption, where all participants are expected to follow the protocol as intended. However, both the aggregator server and the clients may behave maliciously or erroneously. The rapidly evolving field of verifiable computation has recently found applications in Machine Learning, specifically in proving the computation of training and inference, leading to

the concept of Verifiable Federated Learning (VFL).

The landscape and terminology here are complicated by the fact that many of these concepts and threat scenarios are orthogonal to each other: verifiability can be achieved separately or in addition to privacy, there is a difference between honest-but-curious, honest-but-unreliable, and fully malicious adversaries, and all of these considerations apply not only to FL, but to distributed (either peer-to-peer, or gossip-based) ML models as well. In addition, all adversarial attacks that apply to traditional ML (data poisoning, model extraction, etc) also apply to FL. Lest not talk about what happens *after* the training of the model is complete, i.e. all the issues involved in private or provable inference, reliability, and model watermarking or fingerprinting. Navigating the complex ecosystem of different properties and security notions for distributed machine learning models can be a difficult task.

1.1 Our Contribution

In this Systematization of Knowledge (SoK) work, we organize in an organic way existing terminology, properties, threat models, and state of the art results in the field of Verifiable Federated Learning. Moreover, we also propose a novel framework to classify VFL schemes according to the properties of privacy, security, and decentralization they achieve. We choose to focus on the following scenarios:

- *Federated only*, i.e. we only look at scenarios where the training dataset is split into private subsets among the participants.
- *Central aggregator only*, i.e. we do not consider peer-to-peer or gossip-based schemes. This is in order to focus on efficient, practical schemes that can have immediate applications.
- *Training and aggregation only*, i.e. we do not address all those problems that arise at inference time, such as privacy of inference and model watermarking.

The reasons for our choices are simply dictated by the need for a focused approach on real-world applications of immediate need, in particular in fields such as healthcare and finance. Inference-time privacy and security are also very important, but orthogonal to our analysis, which is focused on the training and aggregation steps in FL. Within the boundaries of this analysis,

we will examine the various approaches and state of the art, and we will discuss various properties of privacy, resilience, verifiability, performance, and decentralization level.

Despite the seemingly narrow scope of this survey, we show that there is enough depth to grant an extensive study that covers many of the existing approaches, made even more advantageous by the fact that such study can easily be expanded, or complement in an orthogonal way other analyses focusing on other aspects of ML. In this SoK we explicitly target the issues that arise when end-users need to trust models that have not been entirely produced by a trusted third party for sensitive applications. We analyze how recent work has attempted to bridge this trust gap, and identify open challenges, assumptions, and trade-offs across different approaches. To the best of our knowledge, this is the first attempt to systematize proposals for Verifiable Federated Learning.

1.2 Related Work

While several works have surveyed verifiability in ML broadly [73, 57, 89, 80, 103], there is still a lack of systematization of verifiability as a first-class dimension specific to FL settings. For example, Xing et al. [103] attempt to systematize verifiability in distributed machine learning settings (i.e., single- vs. multi-worker, and training vs. inference), but they focus primarily on circuit optimizations and proof efficiency, treating FL simply as an application scenario. Instead, the closest work in scope is a recent survey that focuses on cross-silo settings and presents protocols primarily through a discussion of cryptographic proof cost drivers and optimizations [60]. In contrast to a scope restricted to cross-silo settings where participants are bound by reputation, we characterize verifiability in broader adversarial environments, including those vulnerable to Sybil attacks and collusion where accountability cannot be assumed. More broadly, the FL literature offers high availability of surveys. Their goals are either largely overlapping or focus on different niches, so comprehensive evaluation metrics across goals have remained fragmented until recently [24]. The supply of these surveys is distributed, though not limited, across several distinct perspectives, including privacy-preserving protocols [105, 29, 25, 66, 17], security threats and defenses [87, 7, 63, 23], fairness, bias and data heterogeneity [84, 93, 14], network topology and decentralization models [10, 82, 106], scalability and convergence efficiency [5, 108, 6]. Notably, Bontekoe et al. recently provided a comprehensive survey of

verifiable privacy-preserving computation over distributed data, cataloging a wide range of cryptographic mechanisms [17]. However, their analysis prioritizes low-level constructions and remains agnostic to the nuances of FL protocols.

In terms of FL evaluation, Chai et al. organize evaluation goals around three primary categories: utility, efficiency, and security and privacy [24]. These respectively capture the quality of the learned model, the system costs incurred to achieve that quality, and resilience against adversarial behavior and privacy leakage. Although this offers a coherent baseline for organizing evaluation, it primarily instantiates all three goal categories through observed outcomes (model performance, measured resource costs, and measured robustness and leakage under attacks). Such metrics don't provide machine-checkable evidence that the protocol was actually executed as specified, nor do they enable detecting deviations as they occur. The authors themselves flag this as critical for any real deployment that goes beyond the semi-honest model. Therefore, we argue that verifiability constitutes a distinct and orthogonal evaluation dimension for FL systems. Unlike security and privacy, which evaluate resistance to attacks or information leakage, verifiability focuses on accountability and auditability of the learning process itself. As such, it adds to existing evaluation goals by enabling stronger trust guarantees without relying solely on observed outcomes or assumed honest behavior.

Adjacent lines of research

We briefly highlight works that do not target FL protocols per se, but introduce system patterns that directly shape what verifiability can mean and what is implementable. In particular, these works inform how to reason about decentralization and trust assumptions in distributed learning.

Conventional FL is *data-parallel* and incentives attach to data value; the cryptographic task is to verify (and, if needed, hide) aggregation of the partially, locally trained models. An emerging related paradigm is that of *model-parallel* training where a neural network is partitioned across parties that contribute compute and bandwidth, with ownership accruing to partial models and reliable stage execution [67]. This opens up to the cryptographic challenge of certifying the correctness of activations and gradients under asynchrony and bandwidth constraints. For example, recent work proposes delay-aware optimizers that make asynchronous pipelines more stable

by compensating for the effect of stale updates [3]. The further challenge of transmitting large activations between stages has been addressed with communication efficient designs suited for billion-parameter models to be trained across ordinary internet links [85]. We observe that, at stage boundaries of model-parallel training, the required guarantees could be cast as a form of proof of aggregation (in the sense of our Algorithm 2), with the distinction that the objects being aggregated are compressed activations or optimizer states rather than plain client gradients. In this view, the same abstraction of aggregation consistency, already central in data-parallel FL, applies, but under the additional constraints of staleness and communication-efficient encoding. Beyond aggregation, stage-local updates could also use proof of training, and since activations ultimately derive from some data, model-parallel training would also require commitment to the dataset, as we summarize in Algorithm 1.

1.3 Structure of this Work

Section II collects preliminaries (FL, verifiable computing, zero-knowledge, and blockchain); Section III formalizes threat models and goals; Section IV introduces our logical framework for verifiable FL; Section V reviews six representative state-of-the-art works in VFL; Section VI systematizes these six works using our framework, and compares them along verifiability, privacy, decentralization, and overhead; Section VIII concludes with discussions and future directions.

2 Preliminaries

In this section, we present the essential building blocks needed to understand how Federated Learning works, together with the concept of Verifiability, as well as the additional building blocks required to combine the two. We assume the reader is already familiar with basic machine learning concepts.

2.1 Federated Learning

In a classic machine learning scenario, if multiple entities intend to build a new model, the most common solution would be to send their data to a central server, which performs the training once it has collected all the datasets.

However, in certain fields, such as healthcare and finance, clients may prefer not to reveal their datasets. For example, we may want to keep data about individuals' health conditions or financial status private. Even if we assume the server is trustworthy, data leaks can still occur. This is where Federated Learning (FL) comes into play. FL is a distributed machine learning technique introduced to tackle the problem of data privacy during the training phase [59]. To better understand how it works, let's first consider the high-level idea, without involving extra actors or tools. For now, let's assume a set of n clients $\{C_1, ..., C_n\}$, each with its own dataset $\{D_1, ..., D_n\}$ and an initial model $\{\theta_{1,init}, ..., \theta_{n,init}\}$, typically $\theta_{i,\text{init}} = \theta_{j,\text{init}} \quad \forall i, j \in \{1, \ldots, n\}$, i.e. the same across all clients. The goal is to collaboratively train a single, shared model in an iterative procedure without sharing their local datasets. Figure 1 shows this basic setup:

1. Each client C_i at round t trains a model locally on its own dataset D_i, obtaining $\theta_{i,t}$, $t \in [1, k), k \in \mathbb{N}$. The number of rounds k is not fixed in advance.

2. Each client C_i sends its local model $\theta_{i,t}$ to an aggregator, which we consider to be a black box for now.

3. The aggregation is performed, producing a global model $\theta_{\text{global},t}$.

4. The global model $\theta_{\text{global},t}$ is then distributed back to each client.

Steps 1 to 4 are then repeated until a stopping criterion is met.

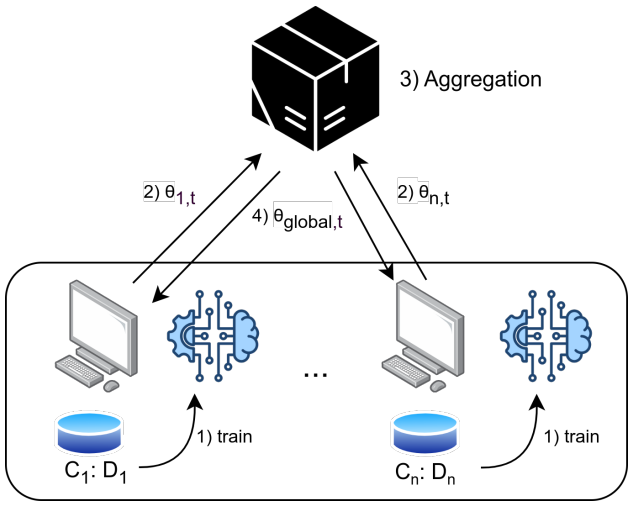

Figure 1: Basic FL setup considering aggregation as a black box

It is worth noting that, in Step 2, clients may send either the full updated model $\theta_{i,t}$ or only the difference $\Delta\theta_{i,t} = \theta_{i,t} - \theta_{global,t-1}$, i.e. the change from the previous model version. Sending deltas instead of full models can reduce communication costs and is often preferred.

2.1.1 Different approaches for aggregation

Aggregation is the process that, at iteration t, combines each $\theta_{i,t}$ from the various clients to obtain a global model $\theta_{\text{global},t}$ which is then redistributed to the clients.

Communication approaches. In Federated Learning we can distinguish two approaches, which lead to two corresponding aggregation strategies. In the first case, a central server collects all the updates $\theta_{i,t}$ and computes $\theta_{\text{global},t}$. In contrast, in a fully distributed approach, the clients communicate the updates $\theta_{i,t}$ with each other according to a predefined protocol, and $\theta_{\text{global},t}$ is computed locally. For the fully distributed approach, among the various communication strategies, the gossip communication approach [56] has been shown to be particularly effective where updates are shared gradually among clients using randomized peer selection. Unlike generic peer-to-peer communication, where each node may attempt to exchange updates with many or all others, gossip protocols limit each node to communicating

with only one or a small, randomly chosen subset of peers at each round. This reduces overall network traffic while still ensuring eventual global dissemination of updates through repeated interactions. In the Federated Learning literature, this was first introduced by Hegedűs et al. [47], and later adopted in more recent works such as [46] and [48].

Aggregation algorithms. Independently of the setup and communication strategy, we can distinguish between different aggregation algorithms [83]; the main ones:

- FedAvg [71]: the client parameters are weighted and averaged to produce the global model; the weight factor for each client is calculated as N_i/N, where N_i is the data volume of client i and N is the total data volume. This is the most widely adopted technique for aggregation.
- FedProx [65]: an extension of FedAvg that adds a proximal term to the local objective to constrain client drift (i.e., the divergence of local model updates due to data heterogeneity) and stabilize training under statistical and system heterogeneity.
- FedNova [98]: another enhancement of FedAvg designed to normalize client updates by their effective local training length to remove the inconsistency introduced by varying numbers of local steps; this improves robustness under non-IID data (i.e., when client datasets follow different, non-identically distributed distributions) and heterogeneous local epochs.
- Scaffold [54]: designed to correct client drift and reduce the variance of local updates in heterogeneous settings, accelerating convergence.

2.1.2 Privacy-Preserving FL

While the basic FL setup allows each client to keep its own dataset private, there exist attacks capable of inferring information about the dataset from the model updates (see Section 4.3). In this section, we briefly present various techniques that enable Privacy-Preserving Federated Learning (PPFL), where privacy is understood as the protection of shared model updates, not only from a potentially curious central server, but also from other clients.

Secure aggregation. Secure aggregation is a broad umbrella of techniques where the focus is on protecting shared model updates not only from a curious central server, but also from other clients in fully distributed setups in which aggregation is performed on the client side [56]. In other words, these techniques enable recovering the same global model as in the non-private setting, while preventing the disclosure of individual local updates. The main techniques are as follows:

- Masking: in this technique, each client uses a mask that is added to its update in order to protect the privacy of its contribution, i.e., $\theta_{i,t}^{\text{masked}} = \theta_{i,t} + s_{i,t}$, where $s_{i,t}$ is the masking factor. The protocol is constructed such that, during aggregation, summing all masked updates leads to the cancellation of the masking factors, yielding the unmasked aggregated model. A relevant construction is due to Bell et al. [8].

- Homomorphic Encryption: a particular type of encryption that allows specific operations to be performed directly on ciphertexts [72]. Once decrypted, the result is the same as if the operations had been performed on the plaintext. This concept can be applied to FL, where aggregation can be done directly on encrypted gradients [33, 52].

- Multi Party Computation (MPC): allows multiple parties to jointly compute a function over their inputs while keeping those inputs private from one another. This approach can be leveraged for a secure aggregation in FL. Practical solutions were first proposed by Bonawitz et al. [16], and have since been explored in more recent works such as [53] and [104].

- TEE-based: Trusted Execution Environments (TEEs) are hardware-backed isolated execution contexts (e.g., Intel SGX, ARM TrustZone) that protect the confidentiality and integrity of code and data during execution, even if the host OS or cloud operator is untrusted. In FL, a recent approach consists in running the aggregation logic inside a server-side enclave: clients send their updates to the enclave, the enclave aggregates them in protected memory, and only the aggregated model is released outside the TEE, preventing the server from inspecting individual contributions [64, 74].

Differential Privacy. Another widely used technique is Differential Privacy (DP), where random noise is added to the data, effectively masking the original values and making it harder for an attacker to infer underlying information [50, 100]. A common way to apply DP in FL is Local Differential Privacy (LDP), consisting in adding noise to the client updates before they are transmitted, so that aggregation is performed over perturbed updates, yielding a noisy aggregated model. Therefore, unlike secure aggregation, the added noise is not designed to cancel out, and it is generally not possible to recover a clean aggregate from the released result. This typically comes at the cost of degraded model accuracy. An alternative approach is to add noise to the aggregated/global model before it is redistributed by the aggregator, this approach is know as Central Differential Privacy (CDP).

Approach	**Privacy Level**	**Model Accuracy**	**Overhead**
Local Differential Privacy	High	Low	High
Central Differential Privacy	Medium	High	Medium

Table 1: Comparison between LDP and CDP

2.2 Verifiable Computing

Modern systems increasingly split computation from trust: mobile apps delegate heavy inference to cloud GPUs, scalable blockchains rely on off-chain provers, and FL servers aggregate millions of client updates they are not willing to re-run. Verifiable computing (VC) gives the caller a succinct cryptographic proof that the remote work was executed correctly, so that verifying the proof is much cheaper - often polylogarithmic - than redoing the job [38]. Formally, a protocol (P, V) is *complete* if an honest prover P convinces the verifier V with overwhelming probability when the result is correct, and *(computationally) sound* if no probabilistic polynomial-time adversary can make V accept an incorrect result except with negligible probability [42]. Most state-of-the-art VC systems build on over thirty years of interactive-proof theory distilled into two reusable gadgets: (i) the *Sumcheck* protocol, that lets V test, through a log-round dialogue, whether $\sum_{x\in\{0,1\}^n} f(x) = H$ for a low-degree polynomial f without enumerating 2^n points [69]; this is made possible by representing f via its multilinear extension, which interpolates Boolean functions over all of $\mathbb{F}$ and enables efficient algebraic checks; and (ii) the *GKR* protocol, that applies Sumcheck layer-by-layer to an arithmetic circuit that

expresses the target program, giving proofs whose size and verify-time are quasi-linear in the input and sublinear in the circuit's gate count [41]. For usability, the protocol's interaction is usually removed with the Fiat–Shamir transform, replacing the verifier's random queries with hashes of the transcript [34]. Combined with polynomial-commitment schemes (e.g., KZG for elliptic-curve groups or FRI for hash-based codes [55, 11]), this yields succinct, non-interactive arguments of knowledge (SNARKs) whose proofs are kilobytes long and verify in milliseconds. Many modern systems also support incrementally verifiable computation (IVC) [96]: a constant-size proof is maintained across an unbounded sequence of steps by recursively composing (or folding) proofs (e.g., Nova [61]).

2.2.1 General-Purpose Zero-Knowledge Proofs

General-purpose zero-knowledge proofs (ZKPs) aim to prove the correctness of arbitrary computations expressed as Boolean or arithmetic circuits rather than a domain-specific relation. Moreover, any argument can be upgraded to *zero knowledge*—revealing nothing beyond the statement's validity—by adding random blinding and by committing to intermediate polynomials. This property becomes indispensable when the computation touches sensitive data or when proofs migrate to public ledgers. The design space of ZKPs is largely described by two orthogonal choices: *(i) Setup.* Early systems require a one-off "ceremony" that produces a circuit-specific structured reference string (SRS) [77, 44]. Later designs relax this by letting any participant rerandomise the SRS, while universal setups—employed, e.g., by Marlin, Plonk, Halo 2—reuse a single size-bounded SRS for arbitrary programs [37, 19, 28]. At the extreme end of the spectrum, fully transparent schemes (STARKs) skip the ceremony altogether, further gaining post-quantum security but at the cost of longer proofs [20, 12, 86]. *(ii) Arithmetisation.* The way a program is translated into algebra greatly influences performance. Rank-1 Constraint Systems (R1CS) and Quadratic Arithmetic Programs encode every logic gate as one simple equation, making witnesses easy to generate [39, 13]. Plonk introduces column-wise table constraints that slash memory and enable cheap recursion [37]. In an Algebraic Intermediate Representation (AIR), the whole execution can be modeled as a long trace that obeys linear recurrences, allowing quasi-linear-time provers over FFT-friendly fields [12]. Finally, modern systems mix R1CS constraints with lookup arguments into read-only tables (for S-boxes, range checks, etc.), cutting prover time by an

order of magnitude on non-numeric parts of real programs [36, 107]. Thus, because each argument system offers different prover and verifier costs, proof sizes, and transparency guarantees, practitioners must choose their framework to balance throughput, auditability, and privacy all of which are crucial for verifiable federated-learning pipelines (see Section 4).

2.3 Blockchain

Blockchain is a technology that allows for the creation of distributed ledgers, mainly used to record transactions. The first blockchain system was introduced in 2008 by Nakamoto with Bitcoin [75], an open and distributed ledger capable of storing transactions in a secure, transparent and persistent way. The blockchain is structured as a temporal sequence of blocks containing transactions, where each block is linked to the previous one using its hash. The blockchain is immutable, meaning that once a new block is added, it cannot be removed. Adding a new block to the blockchain is a distributed process: there is no central authority deciding which block to add and which to reject. Instead, some participants in the network run a consensus protocol to agree on the next valid block. A blockchain can be public or private: the former are open, permissionless networks where anyone can participate and validate transactions, while the latter are permissioned systems that restrict access and control to a defined group of trusted entities. In both cases, the two most widely used consensus protocol families are Proof of Work and Proof of Stake.

Proof of Work (PoW). This is the approach adopted, e.g., by Bitcoin and performed by the so-called miners. Specifically, they try to find a nonce such that the hash of the latest block's content concatenated to this nonce is a value below a target threshold defined by the network's difficulty. The first miner to find a valid nonce broadcasts the new block to the network. If the block is valid, it is appended to the blockchain. This process ensures that adding a block requires significant computational effort, making it expensive to alter the chain and thus enhancing security.

Proof of Stake (PoS). This is the approach adopted, e.g., by Ethereum [22] and other well-known blockchains. The protocol is performed by validators, that are selected to propose new blocks based on the amount of tokens

they have staked as collateral. After a block is proposed, a set of validators participates in an attestation process, where they sign the block to confirm its validity. If the majority agrees, the block is added to the blockchain. This approach eliminates the need for energy-intensive computations, offering better efficiency compared to Proof of Work, at the cost of reliance on the validators' honest collective behavior.

Smart contracts. The concept of smart contracts was introduced by Nick Szabo [92] and it gained popularity with the support provided by the Ethereum blockchain. A smart contract is a computer program designed to execute automatically upon user calls, according to the terms or agreements of the contract itself. Ethereum introduced a Turing-complete language called Solidity to write smart contracts, and since then, many other blockchain platforms have adopted support for them. A smart contract, once deployed on a blockchain, is immutable, deterministic, and transparent in its execution, due to the intrinsic properties of blockchain systems. Thanks to these properties, it becomes a fundamental component for applications where the correctness and verifiability of a process are essential requirements. As for fetching data from off-chain, smart contracts can rely on Decentralized Oracle Networks (DONs), sets of independent nodes that retrieve or compute external data, attest to its integrity, and deliver a verifiable aggregate on-chain, enabling smart contracts to react securely to external inputs.

2.4 Proof of Training

As ML systems scale, reproducing training to audit claims about how a model was obtained is economically prohibitive. The general idea is to demonstrate the integrity of the ML training computations without redundant work.

2.4.1 Technical Overview

Training neural networks relies on repeated *tensor operations*—such as matrix multiplication and convolution—performed during *forward propagation* (data flow through the network) and *backward propagation* (gradient computation). At each layer, non-linear activation functions (e.g., ReLU, sigmoid) allow the network to learn complex patterns. Thus, with an input tensor $\mathbf{X} \in \mathbb{R}^{d_1 \times \cdots \times d_n}$ and weights $\mathbf{W} \in \mathbb{R}^{m_1 \times \cdots \times m_k}$, the transformation is

$\mathbf{Y} = f(\mathbf{W} \cdot \mathbf{X})$, where $\cdot$ indicates either convolution or matrix multiplication. Backward propagation computes gradients $\nabla_{\mathbf{W}} L(\theta)$ of the loss $L(\theta)$, used to update parameters as $\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L(\theta)$, with η as learning rate. At a high level, the training process can be abstracted in terms of succinct proving systems (see 2.2), where every arithmetic step is verifiably correct. Given a tensor $T : [d_1] \times \cdots \times [d_n] \to \mathbb{F}$, its multilinear extension $\widetilde{T} : \mathbb{F}^n \to \mathbb{F}$ is the unique polynomial of degree one in each variable that coincides with T on the Boolean hypercube. Encoding $\mathbf{X}$, $\mathbf{W}$, and all intermediate activations in this way, an entire layer can be represented as an identity between low-degree polynomials. For example, a fully-connected layer computes $Y_i = \sum_{j=1}^{d} W_{i,j} X_j$ for all i, and correctness can be verified by checking that $\sum_{j=1}^{d} \widetilde{W}(r, j), \widetilde{X}(j) - \widetilde{Y}(r) = 0$ at a random verifier challenge r (this is also done for proof of inference, e.g., [68]). This type of relation can be efficiently verified with a sum-check or GKR protocol. Activation functions, such as nonlinear ones, are more difficult yet possible to build into these frameworks [91]. As for convolutions or higher-order tensor contractions, the output can be written as $Y_{i_1,\ldots,i_p} = \sum_{j_1,\ldots,j_q} W_{i_1,j_1} X_{j_1,\ldots,j_q}$, leading to similar polynomial relations. Finally, by recursively aggregating proofs for each layer's forward pass, backward gradient computation, and parameter update using an incrementally verifiable computation (IVC) framework—such as Nova [62], which is based on folding schemes, or recursive SNARK–based systems like Halo2 [18, 30] and Plonky2 [81]—one can obtain a single, succinct proof attesting that all updates $\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L(\theta)$ were performed correctly throughout the training process.

2.4.2 State of the Art of zkPoT

The first systematic treatment shows that the randomness introduced by stochastic gradient descent (SGD) can—in principle—be used as a *witness* certifying that an honest trainer expended (at least) as much work as gradient descent itself, albeit with several impracticalities [51]. The last two years have seen a surge of constructions that make proofs of training both succinct and zero-knowledge:

1. **Kaizen** (CCS '24) by [1] combines an iteration-level proof of gradient descent built from an optimised GKR protocol with recursive composition. A batch of $N = 16$ images of size 32×32 can be verified in 103 ms, while the prover spends $\approx$ 22 min per iteration on VGG-11

(10 M parameters); the proof is 1.36 MB irrespective of the number of iterations.

2. **ZKAudit** (ICML '24) by [97] focuses on *trustless audits*: the model owner commits to the training data and weights and proves (in Halo2) that SGD was executed faithfully. End-to-end audits of MobileNet v2 (1.0, 224) take $\approx$ 47–328 s per SGD step depending on the depth multiplier; a small DLRM model is handled in 3–23 s per step.

3. **zkDL** (TIFS '24) by [91] introduces *zkReLU*—a specialised proof for ReLU and its gradient—and *FAC4DNN*, a flattened arithmetic-circuit representation that aggregates forward and backward passes across layers and training steps. With CUDA acceleration, an 8-layer perceptron (10M weights, batch size 64) is proved in <1 s per iteration; the verifier cost stays logarithmic in the model size and independent of the number of steps.

4. **Verifiable Machine Unlearning** (SaTML '25) shows how the Spartan proof system can certify that retraining *removes* a designated datum [32]. Although its primary goal is unlearning, the scheme implicitly provides a training proof for linear and shallow two-layer networks; scalability to modern DNNs remains open.

As shown, current zkPoT systems already handle $\sim 10^7$ parameters with sub-second verification, yet the prover cost is still dominated by large-field arithmetic and the scarcity of GPU-friendly recursion primitives. Compared to the less expensive proof of model inference, three obstacles persist: (i) a single SGD step embeds both forward and backward passes and is therefore $\approx 10^2\times$ more expensive—Kaizen still spends 6–22 min per step on mid-size CNNs. (ii) Non-arithmetic activations (e.g. ReLU) lie outside the algebraic front-ends used by SNARKs; zkDL overcomes this for ReLU via its dedicated gadget, yet other activations remain unsupported. (iii) Modern DNNs mix layers with wildly different tensor shapes; although FAC4DNN clusters operations of similarity to avoid zero-padding overhead, proof aggregation across truly heterogeneous layers is still arguably inefficient. A recent survey of [80] synthesizes these challenges and reviews lookup-based protocols that could further shrink the gap between theory and practice in future zkPoT systems.

3 Challenges in Federated Learning

We have seen how the Federated Learning setup addresses the problem of data privacy by decentralizing the training process. However, this setup introduces additional parties that may operate in partially trusted or even adversarial environments. As a result, FL systems must account for threats arising from both the clients and the central aggregator [27]. Here, we first state goals/assumptions, then define the threat model and summarize known attacks.

3.1 Security goals & assumptions

- *Privacy:* limit information leakage about any client's local data (see Section 2.1.2).
- *Robustness:* resist poisoning and Byzantine behavior so the global model remains accurate.
- *Availability:* tolerate dropouts and stragglers; avoid targeted update suppression.
- *Channels:* authenticated transport; specify whether secure aggregation is deployed (server sees only aggregates) and whether TEEs are assumed.
- *Adversary scope:* allowed fraction of Byzantine clients; whether server may be honest-but-curious or malicious; whether collusion is possible.

3.2 Threat Model

For clients, it is typically assumed either an honest-but-curious or a malicious behavior. In the former case, clients may attempt inference or reconstruction attacks; in the latter, they may perform active attacks such as data poisoning or free-riding

For the aggregator, it is typically assumed either an honest-but-curious or a malicious behavior as well.

3.3 Known Attacks

In the following table we summarize the main attacks

Role	Model	Description
Client	Honest	Executes the training procedure correctly, without attempting to infer information or alter the global model.
	Honest-but-curious	It executes the training protocol correctly but can attempt to infer information from the global model or from other clients' gradients. The latter is more feasible in gossip-based setups.
	Byzantine	It can perform active attacks by sending adversarial updates with the goal of deviating the global model, or by not participating in the protocol at all.
	Colluding	A set of clients coordinate (e.g., Sybil or coalition) to bias the global model or extract information.

Table 2: Behavioral models of the clients in Federated Learning

Role	Model	Description
Aggregator	Honest	It executes the protocol correctly, without attempting to infer information or alter the aggregation procedure.
	Honest-but-curious	It executes the protocol correctly but can attempt to infer information from the gradients received from the clients.
	Byzantine	It can alter, ignore, or create false updates (e.g. performs selective dropping or model replacement).

Table 3: Behavioral models of the aggregator in Federated Learning

3.4 Open issues

The threat taxonomy summarized above provides a map of adversarial strategies in federated learning and informs the design of defenses aimed at preventing, detecting, or mitigating such attacks. Different systematization of these attacks exist in the literature, for instance highlighting that low-budget and low-visibility attacks can still induce disproportionate damage in practical deployments [63]. Secure system design remains largely *ex ante*, as it focuses on anticipating deviations and evaluating their potential effects. Existing FL deployments are in fact evaluated primarily through outcomes (e.g., accuracy, convergence, resource costs, and robustness under chosen attacks) [24]. However, with respect to *how* individual model updates were produced and *which* updates ultimately determined a published global model, such evaluations provide limited evidence that can be audited *during execution* or *ex post*. This limitation is exacerbated by privacy-preserving deployments that adopt secure aggregation which make the server, and thus any external

Attack Type	Description
Poisoning attacks [94]	Data poisoning: the attacker can add data to the training set maliciously. Model poisoning: the attacker can send adversarial gradients to corrupt the global model update.
Inference attacks [88]	The attacker can infer useful information starting from the data transmitted by the client.
Reconstruction attacks [35]	Under certain conditions the attacker can estimate the attribute information of the original data.
Model extraction [95]	Typically, once training is complete, the global model is exposed through an API. Under certain conditions, it is possible to extract the model by observing its input-output behavior.
Selective dropping	The aggregator ignores certain client updates to manipulate the aggregated model.
Free riding	Clients reuse stale updates or skip local training while claiming credit, harming convergence and fairness.
Sybil / collusion	Adversary spawns many clients or coordinates several to dominate aggregation or evade defenses.

Table 4: Representative attacks against FL under the above threat models

auditor, unable to observe individual updates in the clear and potentially hinder regulatory compliance [101]. Furthermore, filling this "evidence gap" is fundamentally different from verifying the quality or effectiveness of the resulting model, which can often be done more efficiently [43]. In contrast, verifiability and auditability ask for statements about the correctness of the entire process transcript. Regardless of whether the setting is cross-silo or cross-edge, the decomposition between client-side work and centralized aggregation implicitly necessitates strong trust assumptions about the honest execution of both entities. How to achieve auditability without such assumptions remains an open issue. Next, we address this challenge by adopting a hierarchical approach to define an FL framework whose auditability spans the full process transcript.

4 A Framework for Verifiable FL

In this section, we show how verifiable computation (see Section 2.2) leads to federated learning by introducing a conceptual framework, intended to structure the exposition and, for the moment, abstracting away from implementability concerns and any performance or overhead considerations.

One useful way to organize verifiability objectives is into three application

classes:

1. **Data inclusion or exclusion.** Prove that a model was trained with or without specific data items (e.g., authorized records, revoked consent).

2. **Provenance and authorship.** Prove that a released model or checkpoint is derived from a given dataset and training specification, enabling attribution and resolving disputes over intellectual property.

3. **Collaborative correctness over partitions.** Prove that multiple independent parties correctly trained submodels on disjoint private partitions and that these contributions (i.e. local model updates) were combined according to the declared protocol.

Notably, (1) and (2) are relevant to ML in general, while (3) is the FL-specific setting where distributed trust and partial observability make provenance and correctness inherently multi-party. This hierarchy also clarifies the main open issues for verifiable FL:

Data validation. How should a client commit to data so that later audits are meaningful under realistic data lifecycles? How can commitments bind not only to items but also to policies (sampling, preprocessing, filtering, consent/authorization constraints) without forcing full disclosure? How can one enforce "right to be forgotten" requirements, i.e., produce post-hoc evidence that subsequent models are independent of specified data?

Verifiable training. Current practice lacks enforceable binding between an update and the declared *architecture, loss, optimizer, hyperparameters*. Even when the learning algorithm is fixed, providing evidence of no duplicated updates and single-use participation remains nontrivial when some clients join and leave training rounds unpredictably.

Verifiable (secure) aggregation. An audit requires public evidence that the server applied the stated aggregation rule to the declared update set, rather than selectively presenting different aggregates to different recipients. These issues can also be seen as a state-management problem where one needs a verifiable log of (i) which updates were submitted, (ii) which were deemed admissible, (iii) which were consumed in which round, and (iv) which global model checkpoints were finalized. This is instantiated via a blockchain

by several recent approaches where smart contracts can act as a coordination and finalization layer [82]. Moreover, the verification logic itself can be enforced either directly on-chain, or via an off-chain prover that produces succinct proofs whose validity is checked on-chain.

4.1 Client's Verifiable Work

First of all, each client should commit to the dataset they intend to use during the training process in a data structure like a Merkle-Tree stored on a blockchain state. Gradient Descent (GD) is widely recognized as the de facto technique used during the training phase. Depending on the specific variant of GD employed, the corresponding data-inclusion constraints must be generated accordingly as the first part of the proof of training.

- Batch Gradients Descent (BGD): the entire training dataset is used to update the model's weights and biases at each iteration. In this case the client would commit to the full dataset and, before starting the training round, demonstrate knowledge of the preimage. This ensures that the training is being performed on the committed data.

- Mini Batch Gradient Descent (mBGD): a small batch of training examples is used to update the model's weights and biases at each iteration. In this case, the client would commit to the concatenation of all batches, that is, to the full dataset reordered according to the batching strategy. During each training round, the circuit concatenates the current batch to the accumulation of the previous ones. At the end of the training process, if the final accumulated dataset matches the preimage of the original commitment, we are guaranteed that the committed dataset was used throughout.

- Stochastic Gradient Descent (SGD): only one training example is used to update the model's weights and biases at each iteration. This approach can follow the same structure as Mini-Batch Gradient Descent, but with a greater number of rounds due to the use of individual examples.

Another important aspect we recommend for achieving fully verifiable Federated Learning is the verifiability of the training process itself (see 2.4). This ensures that the client actually performs the training procedure starting

from the committed dataset. The client generates a proof of correct execution, which is sent along with the model updates to the server for aggregation. If the proof is invalid, or missing entirely, the corresponding updates are considered invalid and discarded.

Algorithm 1 Verifiable local training at iteration t (executed independently by each peer i)

Input: Dataset D_i, initial model $\theta_{global,t-1}$, training function $\mathcal{T}$, commitment scheme $\mathcal{C}$, proving function $\mathcal{P}$

Output: Model update $\Delta\theta_{i,t}$, proof $\pi_{i,t}$

1: Create batches: $B \leftarrow \text{partition}(D_i, strategy)$ ▷ Depending on strategy: $|B| = 1$ (BGD), $|B| = |D|$ (SGD), $1 < |B| < |D|$ (mBGD)
2: Commit to dataset: $d \leftarrow \mathcal{C}(B)$ ▷ Depending on strategy: $|d| = 1$ (BGD), $|d| = |D|$ (SGD), $1 < |d| < |D|$ (mBGD)
3: Train model: $\theta_{i,t} \leftarrow \mathcal{T}(\theta_{global,t-1}, B)$
4: Compute update: $\Delta\theta_{i,t} \leftarrow \theta_{i.t} - \theta_{global,t-1}$
5: Generate proof for train: $\pi_{i,t} \leftarrow \mathcal{P}(B, \Delta\theta_{i,t}, d)$
6: Send $(\Delta\theta_{i,t}, \pi_{i,t})$ to the Aggregator

In the proof generation step of Algorithm 1 (line 6), the batched-dataset is provided as a private input, while $\Delta\theta_{i,t}$ and d are given as public inputs. The proof $\pi_{i,t}$ demonstrates that, starting from B_i and performing the training procedure, $\Delta\theta_{i,t}$ is obtained, and that the batches used in each round lead to the corresponding commitments d. By committing to the data and constructing the proving function P in a tailored way, we can obtain guarantees for (1), (2), and (3) discussed at the beginning of this section. In the following we provide an intuition for a possible approach for (1), requiring the introduction of a simple constraint in the proving function P of Algorithm 1 proving that a model has been trained without a specific element(s) $\bar{x}$:

- BGD: $d = C(B) \wedge \nexists b_i \in B : b_i = \bar{x}$, i.e. proving knowledge of the batch leading to the corresponding commitment and that each sample of the batch, i.e. each sample of the dataset, is not equal to $\bar{x}$
- SGD: $\forall b_i \in B : d_i = C(b_i) \wedge b_i \neq \bar{x}$, i.e. proving knowledge of every sample of the dataset leading to the corresponding commitments and that those samples are not equal to $\bar{x}$

- mBGD: $\forall b_i \in B : d_i = C(b_i) \wedge \nexists x \in b_i : x = \bar{x},$, i.e. proving knowledge of every (mini) batch leading to the corresponding commitments and that each sample of each (mini) batch is not equal to $\bar{x}$

4.2 Aggregator's Verifiable Work

The final important step in the Federated Learning procedure is aggregation, where a specific aggregation algorithm is executed to produce the updated global model, which is then redistributed to all clients. It is important to make this step verifiable as well, to ensure that the correct updated model is actually redistributed. Two main approaches can be considered in this context:

- Leveraging a smart contract for aggregation: in this approach, the smart contract, deployed on a public ledger, verifies all training proofs and then performs the aggregation. Thanks to its immutable nature and transparent execution, the entire process can be publicly audited. The procedure is illustrated in Algorithm 2.

- Perform the aggregation off-chain, while providing a proof of correct execution that can be verified on-chain. In this case, a commitment/nullifier mechanism is required to ensure that the aggregation algorithm uses only the updates correctly generated by the clients, committed after the training procedure, and that each update is used exactly once. In this approach, the training proofs received by the aggregator must also be verified before the effective aggregation take place; one possible choice is to rely on a verifier deployed on-chain as a smart contract, providing auditability for the process.

Algorithm 2 On-Chain Aggregation via Smart Contract at iteration t

Input: Set of updates $\{\Delta\theta_{i,t}\}_{i=1}^{n}$, set of proofs $\{\pi_{i,t}\}_{i=1}^{n}$, commitment set $\{c_i\}_{i=1}^{n}$

Output: Global model update $\theta_{\text{global,t}}$

Initialize valid update list: $\mathcal{L} \leftarrow \emptyset$
for each client $i \in \{1, \ldots, n\}$ **do**
    if VerifyProof$(\pi_{i,t}, \Delta\theta_{i,t}, c_i)$ = `true` **then**
        Append $\Delta\theta_{i,t}$ to $\mathcal{L}$
    else
        `continue`
    end if
end for
Compute aggregated update: $\theta_{\text{agg,t}} \leftarrow \text{Aggregate}(\mathcal{L})$
Update the global model: $\theta_{global,t} \leftarrow \theta_{global,t-1} + \theta_{agg,t}$
Emit $\theta_{\text{global,t}}$

Data commitment, verifiable training (constraining training to the committed dataset and enforcing data-validation constraints), and verifiable aggregation, when well coordinated, make the entire federated learning process more robust and significantly reduce the feasibility of cheating.

5 State-of-the-Art in VFL

In this work, we examine in depth six representative works in the area of VFL. These contributions were selected for the relevance of the frameworks they introduce, which leverage verifiability to address some of the challenges of FL discussed in Section IV. Three papers appeared in peer-reviewed journals, two in conference proceedings, while the work by Xing et al. [102] is, at present, available only as a preprint.

Title	Year	Venue	Paper type
zkFL: Zero-Knowledge Proof-Based Gradient Aggregation for Federated Learning [99]	2023	IEEE TBDATA	Journal
zkFDL: An efficient and privacy-preserving decentralized federated learning with zero knowledge proof [2]	2024	IEEE ICAIC	Conference
zkPPFL: Zero-Knowledge Proof-based Practical Federated Learning on Blockchain [102]	2023	arxiv	Preprint
VPFL: Enabling verifiability and privacy in federated learning with zero-knowledge proofs [70]	2024	ACM	Journal
Federify: A Verifiable Federated Learning Scheme Based on zkSNARKs and Blockchain [58]	2024	IEEE Access	Journal
VerifBFL: Leveraging zk-SNARKs for A Verifiable Blockchained Federated Learning [9]	2025	IEEE NOMS	Conference

Table 5: Relevant papers on Verifiable Federated Learning

From this point onward, we assume that the use of committed data is proved as part of the training phase. In what follows, we analyze two works that provide verifiability only in the aggregation phase, and then moving to four others that offer broader verifiability guarantees.

Notation

- C_i: client i
- D_i : dataset of client i
- $\theta_{i,t}$: weights of client i at round t
- $\theta_{global,t}$: aggregated weights at round t
- $\{pk_i, sk_i\}$: keypair of a user i
- sig_i : a signature by user i
- c_i : a ciphertext by user i
- d_i : a commitment by user i (to disambiguate from the ciphertexts)
- π_t : a ZK generated at round t (alternatively, to distinguish ZKPs used for different purposes in the same work, you can find the notation α, β, and γ).
- $H_{i,t}$: an hash performed by user i at round t

- $(x_1, \ldots, x_n)$ represent an ordered vector of n elements
- $\{x_1, \ldots, x_n\}$ represent a set of n elements
- $u = (u_1, \ldots, u_n)$ represents the statement vector
- $w = (w_1, \ldots, w_n)$ represents the witness vector

Note: The meaning of subscript indices may vary across the papers; it will always be specified.

5.1 Verifiable Aggregation

Aggregation in FL refers to the step in which model updates from participants are collected, combined using a specific strategy, and then redistributed to the clients. In the following we present two works that introduce verifiability solely at the aggregation stage.

5.1.1 zkFL: Zero-Knowledge Proof-Based Gradient Aggregation for Federated Learning

The first work we present is by Wang et al.[99]: they propose a blockchain-based Federated Learning framework in which the aggregator proves the correctness of the aggregations using ZKPs, which are verified on-chain.

Protocol Overview

Considering n clients, for each round t:

1. Clients train the model locally obtaining the updates $\{\theta_{i,t}\}_{i=1}^{n}$.
2. Clients encrypt their updates using Pedersen commitments [79] obtaining $\{c_i\}_{i=1}^{n}$, where $c_{i,t} = Enc(\theta_{i,t}) = g^{\theta_{i,t}} \cdot h^{s_{i,t}}$, g and h are public parameters and $s_{i,t}$ is randomly chosen by client i.
3. Clients signs their encrypted updates obtaining $\{sig_{i,t}\}_{i=1}^{n}$.
4. Each client i sends $(\theta_{i,t}, s_{i,t}, c_{i,t}, sig_{i,t})$ to the Aggregator.
5. Aggregator aggregates $\{\theta_{i,t}\}_{i=1}^{n}$ obtaining $\theta_{global,t} = \sum_{i=1}^{n} \theta_{i,t}$.
6. Aggregator aggregates $\{c_{i,t}\}_{i=1}^{n}$ obtaining $c_{\theta_{global,t}} = \prod_{i=1}^{n} c_{i,t}$ and signs it with its private key obtaining sig.
7. Aggregator generates a ZKP π_t proving that the aggregation was performed correctly, encryptions by the clients were performed correctly and signatures were verified correctly.
8. Aggregator sends $(\theta_{global,t}, c_{\theta_{global,t}})$ to the n clients.
9. Aggregator sends π_t on-chain and, if verified, $H(c_{\theta_{global,t}})$ is published.
10. Before starting the next round, each client i checks if $H(c_{\theta_{global,t}})$ is appended on-chain. If the check is valid, the clients start the local training based on $\theta_{global,t}$.

The system was deployed on the Ethereum public blockchain, with Halo2[18] employed as the zero-knowledge proving scheme. The aggregation strategy used, as reported by authors is FedAVG [71]. Verifiability is achieved through a proof generated by the aggregator, demonstrating the correctness of the aggregation and ensuring that only user-submitted weights were used for the computation; formally:

$$u = (\{c_{i,t}\}_{i=1}^{n}, \{sig_{i,t}\}_{i=1}^{n}, c_{\theta_{global,t}})$$

$$w = (\{\theta_{i,t}\}_{i=1}^{n}, \{s_{i,t}\}_{i=1}^{n}, \theta_{global,t})$$

A ZKP is generated with the following constraints:

1. $\forall i, c_{i,t} = g^{\theta_{i,t}} \cdot h^{s_{i,t}}$, recalling that g and h are public

2. $\theta_{global,t} = FedAVG(\{\theta_{i,t}\}_{i=1}^{n})$

3. $Verify(pk_i, sig_{i,t}) = 1$

In zkFL, security against global model tampering is guaranteed since the aggregator generates a ZKP for the aggregation process. Selective dropping is also mitigated, as the encryption of each gradient is passed as a public input for the aggregation proof verification. Hence, each client can check that its gradient was actually included in the aggregation process

5.1.2 zkFDL: An efficient and privacy-preserving decentralized federated learning with zero knowledge proof

This work by Ahmadi and Nourmohammadi [2] proposes another FL framework leveraging blockchain, in which the server performs a zero-knowledge proof to demonstrate to the clients the correctness of the aggregation. Additionally, it is proven that all inputs provided by the clients have been used. The following provides an explanation of the protocol, which involves one central server and M clients registered with a public address on the blockchain.

The system was deployed on the Ethereum blockchain, using the Groth16 [44] zero-knowledge proving scheme to ensure the correctness of the aggregation process. The circuit takes as public input the hashes of each update, their summation, and the aggregated result, while it takes the raw updates as private input. The choice to use the hashes of the updates as public inputs is aimed at reducing the cost of proof verification and at hiding the actual updates, since verification takes place on a smart contract where all interactions are publicly visible. Formally:

$$u = (\{H_{i,t}\}_{i=1}^{n}, H_{sum,t}, H_{\theta_{global,t}})$$
$$w = (\{\theta_{i,t}\}_{i=1}^{n}, \theta_{global,t})$$

A ZKP is generated with the following constraints:

1. $\forall i, H_{i,t} = MiMC7(\theta_{i,t}), H_{sum,t} = \sum_{i=1}^{n} H_{i,t}$

2. $\theta_{global,t} = \sum_{i=1}^{n} \theta_{i,t}$

3. $H_{\theta_{global,t}} = MiMC7(\theta_{global,t})$

Protocol Overview

For each round t:

1. Clients compute the updates $\{\theta_{i,t}\}_{i=1}^{n}$ starting from $\theta_{global,t-1}$ and send them to the server.
2. The server hashes the updated weights using MiMC7, a variant of the well known zk-friendly hash function MiMC [4]:
   $H_{i,t} = MiMC7(\theta_{i,t}), \forall i \in [1, n]$ and calculates $H_{sum,t} = \sum_{i=1}^{n} H_{i,t}$.
3. The server calculates $N_{sum} = \sum_{i=1}^{n} N_i$ and performs the aggregation $\theta_{global,t} = \sum_{i=1}^{n} \frac{N_i}{N_{sum}} \theta_{i,t}$, where N_i is the contribution by the client i.
4. The server hashes the aggregated result: $H_{\theta_{global,t}} = MiMC7(\theta_{global,t})$.
5. The server generates a proof π_t for the aggregation process and sends it to the clients.
6. The server deploys $Contract_h$ and $Contract_p$, two smart contracts for the verification phase.
7. Each client i submits its respective $H_{i,t}$ to $Contract_h$, from which the stated $H_{sum,t}$ is expected to be obtained.
8. If the check on $Contract_h$ passes, the ZKP π_t is verified on $Contract_p$.

Now, two contracts are deployed: $Contract_h$ and $Contract_p$. In $Contract_h$, each client submits its respective $H_{i,t}$, and after the last client submission, $H_{sum,t}$, as stated by the aggregator, is expected to be obtained. If this is not the case, it is not possible to proceed with the next iteration. With $Contract_p$, if the previous check has passed, the ZKP generated by the aggregator is verified; if it is correct, the process proceeds to the next iteration, provided that no stopping criterion has been met. Since the hashes of the gradients are passed as public input, this allow each client to verify that its gradient was included, avoiding selective dropping.

5.2 Verifiable Training and Aggregation

Some works, despite the limitations we'll discuss in Section 5.3, have attempted to make the training phase verifiable as well.

5.2.1 zkPPFL: Zero-Knowledge Proof-based Practical Federated Learning on Blockchain

Xing et al.[102], to the best of our knowledge, is the first work to approach the problem of making both training and aggregation verifiable. In this protocol are involved: the *Publisher* (who also acts as the Aggregator), and m clients, referred to by the authors as *trainers*. In the following, we aim to bring clarity to the various phases by simplifying and streamlining the notation used in the original paper.

Protocol Overview

The training algorithm F is split into q identical and consecutive subalgorithms P (q is the number of epochs). P is converted into a circuit R, which is sent to n trainers. Then, for each round t:

1. Trainer i trains the model on D_i obtaining $\theta_{i,t}$.
2. Trainer i generates proofs $\{\pi_{\mathrm{i,j}}\}_{\mathrm{j}=1}^{q}$, one for each training epoch, where the statements for each one are $\{u_{\mathrm{i,j}}^{'}\}_{\mathrm{j}=1}^{q}$, modifications of the original ones $\{u_{\mathrm{i,j}}\}_{\mathrm{j}=1}^{q}$ to conceal intermediate gradients.
3. Trainer i generates two ZKPs for each training epoch, $\{\alpha_{\mathrm{i,j}}\}_{\mathrm{j}=1}^{q}$ and $\{\beta_{\mathrm{i,j}}\}_{\mathrm{j}=1}^{q}$, proving that the modification of the statement is valid, and that the output of the previous epoch is the input to the next one.
4. Trainer i sends $(\{\pi_{\mathrm{i,j}}\}_{\mathrm{j}=1}^{q}, \{u_{\mathrm{i,j}}^{'}\}_{\mathrm{j}=1}^{q}, \{\alpha_{\mathrm{i,j}}\}_{\mathrm{j}=1}^{q}, \{\beta_{\mathrm{i,j}}\}_{\mathrm{j}=1}^{q})$ to the Publisher, which verifies all these proofs.
5. The publisher runs a key generation algorithm, obtaining $\{pk_h, sk_h\}$ and sends pk_h to all Trainers.
6. Trainer i computes $c_i = \mathrm{Enc}_{pk_h}(u_{\mathrm{i,q}} + a - b)$, encrypting the output of the final epoch with noise added.
7. Trainer i generates a ZKP γ_i proving that c_i encrypts the same gradients value that has been modified to obtain $u_{\mathrm{i,q}}^{'}$.
8. Each trainer i sends (c_i, γ_i) to a smart contract, which checks the correctness of the proofs and performs the aggregation, obtaining $c_{sum} = \sum_{i=1}^{n} c_i$.
9. The publisher can then obtain the global model as $c_{sum}^{-} = \frac{\mathrm{Dec}_{\mathrm{sk}_h}(c_{sum})}{n}$.

In this process the verifiability of the aggregation is achieved via a smart contract. In contrast, the verifiability of the training is ensured by executing a Groth16 prover at each training epoch, being linked using a $\Sigma - protocol$ [31] to guarantee the correctness of the overall process. Formally:

1. The client i runs the training process and generates a ZKP for each epoch, resulting in a total of q ZKPs: $\{\pi_{i,j}\}_{j=1}^{q}$. The statement being used is $\{u_{\mathrm{i,j}}^{'}\}_{\mathrm{j}=1}^{q}$, a modification of the original one $\{u_{\mathrm{i,j}}\}_{\mathrm{j}=1}^{q}$ to conceal intermediate inputs and outputs. More specifically, focusing on a given

epoch $\bar{j}$, the public statement can be expressed as:

$$u_{i,\bar{j}} = \{a_k\}_{k=1}^{l}$$

For each element a_k, a random value t_k is sampled from a uniform distribution and added to it, producing the modified element a'_k. The updated public statement is then:

$$u'_{i,\bar{j}} = \{a'_k\}_{k=1}^{l}.$$

The dataset is instead kept private being included in the witness vector.

2. Client i generates two additional ZKPs for each training epoch: $\{\alpha_{i,j}\}_{j=1}^{q}$ and $\{\beta_{i,j}\}_{j=1}^{q}$. The first proves that the modification of the statement is valid; the second proves that the output of epoch j is used as the input for epoch $j+1$.

3. Client i sends $(\{\pi_{i,j}\}_{j=1}^{q}, \{u'_{i,j}\}_{j=1}^{q}, \{\alpha_{i,j}\}_{j=1}^{q}, \{\beta_{i,j}\}_{j=1}^{q})$ to the aggregator, who verifies all the proofs.

4. Client i computes $c_i = \mathrm{Enc}_{\mathrm{pk}_h}(u_{\mathrm{i,q}} + a - b)$, where pk_h is the public key of the aggregator and $a-b$ is some noise added to the output of the last training epoch. This guarantees that even if the aggregator attempts to decrypt before the encrypted global model has been obtained, it cannot recover the gradients in plaintext.

5. Client i also generates γ_i proving that c_i encrypts the same gradients value that has been modified to obtain $u'_{i,q}$ (i.e. the output of the last training epoch).

6. Each trainer i sends (c_i, γ_i) to a smart contract, which verifies the proof.

An important aspect of this process is that it is divided into two phases. The first phase solely proves the correctness of the training while protecting the inputs and outputs of each epoch. Instead of immediately providing the gradients to the aggregator, it sends only the necessary data to verify the correctness of the computation. In the second phase, the outputs of the last epoch are perturbed and encrypted, then sent to a smart contract that subsequently performs the aggregation. Once the aggregation is complete, the aggregator can decrypt using its private key to obtain the new global model.

This framework makes it possible to prevent free-riding, as each client must generate a proof of having correctly performed the training process. Model poisoning can likewise be mitigated, since the resulting gradients must be proven to follow a well-defined process, preventing arbitrary values from being sent to the server.
No public ledger is explicitly mentioned in the paper. We therefore infer that the system is intended for deployment on a permissioned blockchain, since aggregation is a computationally intensive task that would be prohibitively expensive to perform on a smart contract in a public blockchain environment, due to both gas limits and high fees. These issues are typically avoided in permissioned settings.

5.2.2 VPFL: Enabling verifiability and privacy in federated learning with zero-knowledge proofs

This work by Ma et al. [70] targets third-party verifiability of FL while preserving data confidentiality by combining homomorphic commitments, Merkle trees, and non-interactive zero-knowledge range proofs (Bulletproofs). In contrast to PoT as described in Section 2.4, VPFL does *not* prove that local updates were computed by running the declared optimizer over the committed data; instead, it proves (i) inclusion of committed data, (ii) boundedness of each client's submitted model parameters, and (iii) additive consistency of server aggregation. In the presented framework, the participating entities include n clients, m data owners, a central server acting as the aggregator, the bulletin board (which can be instantiated using a blockchain), the auditor, and the model user. While in many FL settings the data owner and the client coincide, VPFL intentionally separates them to enable storage-auditability: data owners upload raw data $d_{i,t}$ to clients, receive Merkle-commitment inclusion proofs, sign the corresponding commitments, and publish digests to a public bulletin board that auditors can check. However, the paper does not argue when (or why) this role should not coincide with the client in standard FL. As the notation in the paper is not always clear, we attempt below to reconstruct the protocol more systematically.

Protocol Overview

Before starting, data owner $i^{'}$ sends raw data D_i to the corresponding client i. Intuition of the algorithm for round t:

1. The server sends the global weights $\theta_{global,t-1}$ to each client i.
2. Each client i generates an inclusion proof $\pi_{i,t}$ for data D_i.
3. Client i trains on $\theta_{global,t-1}$ to obtain $\theta_{i,t}$, and generates a range proof $\pi^{'}_{i,t}$ attesting that $\theta_{i,t} < \theta_{\max}$, where $\theta_{\max}$ is a threshold set by the server.
4. Client i sends $(\theta_{i,t}, \pi^{'}_{i,t})$ to the server, along with a digest $d_{i,t} = \text{COM.Pedersen}(\theta_{i,t})$; $d_{i,t}$ is also published to the bulletin board.
5. The server aggregates all the $\{\theta_{i,t}\}_{i=1}^{n}$ that pass $Verify(\pi^{'}_{i,t}, .)$, obtaining $\theta_{global,t}$.
6. The server generates also a range proof $\pi_{global,t}$ attesting that $\theta_{global,t} < \theta_{global\_max}$.
7. The server then calculates $d_{global,t} = \sum_{i=1}^{n} d_{i,t}$ and sends $(d_{global,t}, \pi_{global,t})$ to the bulletin board.

The digests sent to the bulletin board are intended to allow third-party verification of aggregation consistency: the auditor recomputes $\bar{d} = \sum_{i=1}^{n} d_{i,t}$ from the client commitments and checks whether $\bar{d} = d_{global,t}$, while the model user verifies $\pi_{global,t}$ to ensure that the global parameters lie within the range. In addition, data owners obtain Merkle-inclusion proofs and signatures attesting that their raw data were committed before training began, which provides storage auditability but does not prove that the committed data were actually used in producing the updates. Crucially, VPFL does not implement a proof of training; formally:

1. To ensure the correctness of the data being used, each client generates a ZKP of inclusion $\pi_{i,t}$ when queried by the data owner, where $u = d_{D_i}$ (the commitment of the dataset) and $w = D_i$, demonstrating knowledge of the preimage (i.e., the dataset) corresponding to its respective commitment in a Merkle tree.
2. To prove at each iteration that the resulting gradients are below a certain threshold, each client generates a range proof $\pi^{'}_{i,t}$ (with Bulletr-

proof [21]) showing that $\theta_{i,t} < \theta_{\max}$ and $d_{i,t} = COM.Pedersen(\theta_{i,t})$, where $u = (\theta_{\max}, d_{i,t})$ and $w = \theta_{i,t}$.

3. After the server performs the aggregation, it generates $\pi_{global,t}$ proving that $\theta_{global,t} < \theta_{global,max}$ where $u = (\theta_{global\_max}, d_{global,t})$ and $w = \theta_{global,t}$.

These steps prevent the model from being poisoned with excessively large gradients that exceed the limit.
Thus, VPFL combines proofs of committed data, bounded updates, and aggregation consistency, but does not establish correctness of the training procedure itself.

5.2.3 Federify: A Verifiable Federated Learning Scheme Based on zkSNARKs and Blockchain

The notable aspect of this work by Keshavarzkalhori et al. [58] is the use of El Gamal threshold encryption [78] to preserve the privacy of model updates. In the framework presented, the participating entities, according to the nomenclature used by the authors, include Model Owners MO (index i) and Data Owners DO (index j). As we will see, the former collaborate in the decryption process, while the latter are the clients responsible for training the model.

Protocol Overview

Each MO_i registers in a Smart Contract its local (partial) public key pk_i; $PK = \sum_{i=1}^{m} pk_i$ is the global public key used for encryption.
For each round t:

1. Each DO_j trains locally to obtain the updates $\theta_{j,t}$ and encrypts them using ElGamal's threshold encryption scheme, resulting in $\zeta_{j,t} = ElG.Enc_{PK}(\theta_{j,t})$.
2. Each DO_j creates a proof $\pi_{j,t}$ proving that the updates and the encryption are computed correctly.
3. Each DO_j sends $(\pi_{j,t}, \zeta_{j,t}, PK)$ to the Smart Contract.
4. The Smart Contract validates the received proofs $\pi_{j,t}$ and aggregates the encrypted updates $\zeta_{j,t}$ by homomorphically summing them, obtaining $\zeta_{global,t}$.
5. Each MO_i retrieves the current model $\zeta_{global,t}$ from the Smart Contract and partially decrypts it: $p_{i,t} = \text{Elg.Dec}(sk_i, \zeta_{global,t})$.
6. Each MO_i creates a SNARK $\pi'_{i,t}$ proving the correct decryption.
7. Each MO_i sends $(\pi'_{i,t}, p_{i,t})$ to the Smart Contract.
8. The Smart Contract validates the received proofs $\pi'_{i,t}$ and sums all the partial decryptions $p_{i,t}$ until the model is fully decrypted.

In this process aggregation takes place on a smart contract deployed on the Ethereum public blockchain, where correctness is ensured by its intrinsic properties. The verifiability of training is guaranteed by a ZKP that proves the entire training phase, including the correct encryption of the updates. The specific ZKP used is Groth16 [44]; formally:

$$u = (\zeta_{j,t}, P), w = (\theta_{j,t}, D_j)$$

Constraints:

1. By performing training on dataset D_j, the gradients $\theta_{j,t}$ are obtained
2. Encrypting $\theta_{j,t}$ with the global public key P yields $\zeta_{j,t}$

Thanks to the use of ElGamal threshold decryption, is possible to mitigate inference and reconstruction attacks, as the gradients remain encrypted.

Likewise, model poisoning attacks can also be mitigated, since the training process is fully verifiable.

5.2.4 VerifBFL: Leveraging zk-SNARKs for A Verifiable Blockchained Federated Learning

The last work we analyze is by Bellachia et al. [9]. It is the most recent among the six works we examined and the first to leverage a recursive ZKP for both the proof of training (effectively a proof of accuracy) and the proof of aggregation. In the described framework, the participating entities are the Task Publishers (TP), the Trainers (TR), and the Aggregators (AG). The framework relies on IPFS, a distributed storage system where each stored object is identified by a Content Identifier (CID)[15].
Setup: the TP publishes the learning task on IPFS and sends the corresponding CID_{task} along with a target accuracy to the blockchain. The blockchain emits $TaskSubmitted(CID_{task})$. Each TR_i and AG subscribes to a specific CID_{task}; the blockchain emits $Subscribed(CID_{task}, address)$. Each TR_i downloads the initial model from IPFS.

Protocol Overview

For each round t:

1. Each TR_i performs local training to obtain $\theta_{i,t}$, which is then injected with ε-differential privacy noise.
2. Each TR_i generates a ZKP $\pi_{i,t}$ proving a certain accuracy.
3. Each TR_i uploads the local model to IPFS and sends the corresponding CID along with $\pi_{i,t}$ to the blockchain.
4. Proofs $\pi_{i,t}$ are verified via a Decentralized Oracle Network (DON), and the AG downloads the local models $\theta_{i,t}$ from IPFS.
5. The AG performs aggregation and generates a proof π_t attesting to the integrity of the resulting global model.
6. The AG uploads the global model to IPFS and sends the corresponding CID along with π_t to the blockchain.
7. The proof π_t is verified on-chain.

This work, similarly to VPFL, proofs only certain properties of the training process, specifically, whether a given level of accuracy has been achieved. The goal is to prove that the value $Acc = \frac{z_n}{n}$ has been correctly computed, where z_n is the number of correct predictions and n is the total number of predictions. At each step, a forward pass is performed, the output is compared with the expected label, and if they match, the counter is incremented: $z_n = z_n + 1$. This process is repeated iteratively over the test set. In contrast, the proof of aggregation focuses on verifying the correct computation of the global model at each round t using Nova [61]; ensuring that $\theta_{\text{global},t} = \sum_{i=1}^{m} \frac{N_i}{N_{\text{sum}}} \theta_{i,t}$, that is, the correct calculation of the FedAvg algorithm.

6 Comparative Analysis

In this section, we compare the previously solutions from various perspectives to assess which features and guarantees they provide.

6.1 Verifiability assessment

In the following table, we summarize, for each work, the extent of verifiability along the axes defined in 4. In the table, PoCD stands for Proof of Committed Data, PoT for Proof of Training and PoA for Proof of Aggregation.

Table 6: Verifiability properties

Work	**PoCD**	**PoT**	**PoA**	**Conceptual Framework alignment**
zkFL [99]	-	-	●	Low
zkFDL [2]	-	-	●	Low
zkPPFL[102]	-	●	●	Medium
VPFL [70]	●	◐	-	Medium
Federify [58]	-	●	●	Medium
VerifBFL [9]	-	◐	●	Low

● provides property; ◐ partially provides property; − does not provide property.

In summary, the analysis highlights that existing approaches only partially cover the verifiability dimensions identified in the logical construction. While some solutions provide proofs of aggregation, guarantees on committed data and training remain largely underexplored or addressed only in limited forms (e.g., range proofs or accuracy checks). As a result, the overall alignment with the logical construction is still weak to medium across all works. This suggests that, despite notable progress, there is no comprehensive solution yet, and current proposals should be regarded as complementary rather than exhaustive in addressing the verifiability challenges of Federated Learning.

6.2 Privacy Guarantees

In this subsection, we evaluate each work from a privacy perspective. In the context of Federated Learning, privacy refers to the protection of the outgoing updates at each iteration (2.1.2). During our review, it emerged that three out of the six works do not implement any privacy-preserving technique; these are zkFL, zkFDL, and VPFL. We now turn to the remaining three, focusing on the strategies they adopt to ensure privacy.

Work	**Methodology for privacy**	**Privacy level**
zkFL [99]	-	○
zkFDL [2]	-	○
zkPPFL[102]	Noise injection	●
VPFL [70]	-	○
Federify [58]	ElGamal threshold encryption	●
VerifBFL [9]	Differential Privacy	●

Table 7: Overview of the gradients privacy level of the analyzed works. Symbols indicate the degree of privacy guarantees: ● denotes high, ◐ medium, and ○ low.

zkPPFL focuses on preventing information leakage during the training process by modifying the public statements of each epoch. This ensures that intermediate gradients remain hidden, while final gradients are further perturbed and encrypted before being released to the smart contract. Although

effective in concealing intermediate computations, this approach introduces additional complexity and still relies on perturbation and encryption of the final outputs to protect privacy.

Federify adopts a different strategy, relying on ElGamal threshold encryption to secure gradients. Here, no single party can decrypt the data on its own: only the collective partial decryptions of all model owners yield the global model. The use of zero-knowledge proofs guarantees the correctness of each decryption share. This design prevents aggregator from accessing raw gradients and achieves a strong form of privacy without perturbing the data.

VerifBFL approaches privacy from the perspective of inference attacks, employing differential privacy to protect client updates before they are shared. By injecting calibrated noise, the system ensures that sensitive information cannot be reconstructed, even when model updates are publicly accessible via IPFS. However, this comes at the cost of a trade-off between privacy guarantees and model accuracy.

While none offers a complete solution on its own, their comparison highlights the spectrum of design choices available for enhancing privacy in verifiable federated learning. Privacy mechanisms and verifiability are, in fact, largely complementary. With secure aggregation (HE/MPC/threshold), the server cannot inspect per-client updates; thus, clients must attach proofs of admissibility (e.g., norm bounds or PoT) and aggregators (or committees) must prove correct homomorphic summation and partial decryptions, as in Federify. With differential privacy, certifying injected noise is subtler than certifying small norms: in practice systems either commit to seeds/ranges or simply state DP is applied—VerifBFL follows the latter route. In contrast, zkFL and zkFDL purposely focus on aggregation proofs with public digests and no PPFL—hence both low privacy and low alignment in our framework—illustrating how optimizing for cheap PoA alone tends to sideline privacy and end-to-end auditability.

6.3 Security Against Attacks

In the following, we discuss the security of each work against the different types of attacks described in Section 3. We will also examine how, by adding

verifiability, it is possible to mitigate certain attacks.

Work	Threat model	Security against
zkFL [99]	Byzantine aggregator Honest clients	Global model tampering Selective dropping
zkFDL [2]	Byzantine aggregator Honest clients	Global model tampering Selective dropping
zkPPFL[102]	Byzantine aggregator Lazy-but-curious clients [1]	Model poisoning * Free riding Global model tampering Selective dropping
VPFL [70]	Byzantine aggregator Byzantine clients	Poisoning attacks* Selective dropping
Federify [58]	Byzantine and curious aggregator Byzantine clients	Inference and reconstruction attacks Model poisoning * Free riding Global model tampering
VerifBFL [9]	Byzantine and curious aggregator Byzantine and curious clients	Inference and reconstruction attacks Free riding Global model tampering

Table 8: Overview of the security guarantees for each work

In zkFL, security against global model tampering is guaranteed since the aggregator generates a ZKP for the aggregation process. Selective dropping is also mitigated, as the encryption of each gradient is passed as a public input for the aggregation proof verification. Hence, each client can check that its gradient was actually included in the aggregation process. Similarly, in zkFDL, the hashes of the gradients are passed as public input, allowing each client to verify that its gradient was included, avoiding selective dropping. Xing et al. in zkPPFL also make it possible to prevent free-riding, as each client must generate a proof of having correctly performed the training process. Model poisoning can likewise be mitigated, since the resulting gradients must be proven to follow a well-defined process, preventing arbitrary values from being sent to the server. VPFL can partially prevent poisoning attacks by generating an inclusion proof of the dataset being used by the clients and a range proof that guarantees the gradients fall below a certain threshold. This prevents the model from being poisoned with excessively large gradients that exceed the limit. In Fedeify, thanks to the use of ElGamal threshold decryption, is possible to mitigate inference and reconstruction attacks, as the gradients remain encrypted. Likewise, model poisoning attacks can also be mitigated, since the training process is fully verifiable. Similarly, VerifBFL

[1]The authors use this term to indicate an honest-but-curious client that can also perform free-riding attacks.

mitigate the problem of inference and reconstruction attacks through the use of differential privacy. This analysis highlights that the proper application of verifiability mechanisms can effectively contribute to mitigating several types of attacks in a federated learning setting.

A final consideration concerns collusion among clients (Sybil or coalition attacks). None of the surveyed works explicitly address this threat. Only VPFL offers partial mitigation, as range proofs constrain the magnitude of submitted updates, reducing the impact of colluding clients but not fully preventing aggregation bias if many identities are controlled.

6.4 Decentralization Level

Federated Learning cannot be strictly considered a decentralized framework for model training, as its basic setup relies on a central server for aggregation. Instead, it is more accurately classified as a distributed system that relies on a coordinator (i.e., the central server). To remove this entity, a new paradigm has recently been proposed, leveraging a gossip-based communication approach [56], thereby moving closer to a truly decentralized protocol. Among the works surveyed in this SoK, none adopt a gossip-based approach; this is a reasonable design choice since most of the works rely on the blockchain as the communication layer and a gossip-style communication typically introduces significant communication overhead resulting in high costs for fees.

Work	**Methodology for aggregation**	**Decentralization**
zkFL [99]	Off-chain + ZKP	◐
zkFDL [2]	Off-chain + ZKP	◐
zkPPFL[102]	Smart Contract	●
VPFL [70]	Off-chain + ZKP	◐
Federify [58]	Smart Contract	●
VerifBFL [9]	Off-chain + ZKP	◐

Table 9: Overview of the decentralization levels of the analyzed works. Symbols denote the degree of decentralization: ● high, ◐ medium, and ○ low.

An alternative path toward decentralization in federated learning is to leverage the blockchain itself not only as a communication layer but also as

the medium for aggregation.
As shown in Table 9, 2 out of 6 works leverage smart contracts for verifiability, which inherently increases the level of decentralization by relying on a decentralized infrastructure. The remaining works rely on a centralized aggregator to perform computations off-chain, providing correctness guarantees through ZKPs.

6.5 Tested ML Models and Benchmarks

This final subsection will compare the works by indicating whether they are evaluated on multiple models or only on a limited set. We will also compare them in terms of performance and computational cost.

Work	**ML models tested**	**Datasets**	**Tasks performed**
zkFL [99]	ResNet18, 34, 50 [45] DenseNet121, 169, 201 [49]	CIFAR-10, [2] Penn Treebank [3]	Image recognition Language understanding
zkFDL [2]	5 custom networks (1 to 5 MLPs)	Daily and Sports Activities [4]	Sport activity prediction
zkPPFL[102]	Custom CNN [5]	MNIST [6], custom handwritten digists	Iris classification House price prediction
VPFL [70]	2 custom CNNs	CIFAR-10, MNIST, Fashion MNIST [7], SVHM [8], EMNIST-balanced [9], Fruits-360 [10]	Image recognition
Federify [58]	Naive Bayes Classifier	Accelerometers sensors data [11]	Activities prediction
VerifBFL [9]	Custom CNN	MNIST	Image recognition

Table 10: Tested ML models.

By analyzing the experiments and evaluations of each work, it emerges that 5 out of 6 studies assessed performance using custom or relatively simple models. In contrast, zkFL made a notable effort to evaluate multiple well-known, deep architectures: residual networks with 18, 34, and 50 layers, and dense networks with 121, 169, and 201 layers.

Work	**Verifiable Training**	**Impact in training**	**Verifiable Aggregation**	**Impact in aggregation**	**Other overhead**
zkFL [99]	✗	None	✓	High	High
zkFDL [2]	✗	None	✓	-	-
zkPPFL[102]	✓	High	✓	-	Negligible
VPFL [70]	✓	Negligible	✓	Negligible	Negligible
Federify [58]	✓	-	✓	-	-
VerifBFL [9]	✓	Low	✓	Low	Negligible

Table 11: Overview of introduced overhead. A - indicates that no values are specified

Table 11 reports, for each work, the overhead introduced by the verifiable training component (if implemented), as well as the overhead from the verifiable aggregation component and any additional operations. While we attempted to base our comparison on setups that are as similar as possible, a precise benchmarking is not feasible due to substantial differences in hardware configurations, number of clients, and training epochs across the reviewed works. For instance, experiments in zkFL were conducted on a GPU-equipped server (NVIDIA Tesla T4, 16 GB RAM), whereas zkPPFl relied on a CPU-based setup (Intel Xeon Gold 5128, 4 GB RAM), highlighting the heterogeneity of the experimental environments.

[2] https://www.cs.toronto.edu/~kriz/cifar.html
[3] https://www.kaggle.com/datasets/aliakay8/penn-treebank-dataset
[4] https://archive.ics.uci.edu/dataset/256/daily+and+sports+activities
[5] https://github.com/xingzhibo2019/ZKP-FL/tree/main/cnn
[6] https://www.kaggle.com/datasets/hojjatk/mnist-dataset
[7] https://tensorflow.google.cn/datasets/catalog/fashion_mnist
[8] http://ufldl.stanford.edu/housenumbers/
[9] https://www.nist.gov/itl/products-and-services/emnist-dataset
[10] https://www.kaggle.com/datasets/moltean/fruits
[11] https://mdx.figshare.com/articles/dataset/Dataset_used_for_federated_learning/14107121

zkFL (Wang et al.) We consider times for the ResNet50 model, 16 clients, and 6 epochs. The time required for a single epoch of train is 2.5 minutes, resulting in 15 minutes for 6 epochs for each client. The most time-consuming component is the verifiable aggregation, taking 105 minutes, of which ~ 55 minutes are spent on ZKP generation. We also account for the overhead introduced by encrypting and signing the updates, which adds another 45 minutes, leading from our calculations to a total round time of ~ 165 minutes.

zkFDL (Ahmadi and Nourmohammadi) In this work, the authors do not report any measurements for training or aggregation times. They only provide accuracy results for varying numbers of epochs and some considerations on ZKP generation, noting that increasing the number of layers raises the total number of constraints in the circuit, thereby increasing the proof generation time and the memory required. We also expect a small overhead due to the large number of hash operations performed, although this is not specified.

zkPPFL (Xing et al.) We consider times for the iris classification task and 5 epochs. In this work, a proof is generated for each epoch in a FL round. The overhead introduced by the ZKP generation for a single epoch is 3.75 minutes, so for 5 epochs the total time is ~ 19 minutes. The time required for the training itself is not specified, nor is the time for the verifiable aggregation. We consider the time needed to perturb the public statements as negligible, since it is just the addition of noise, although this is not specified either.

VPFL (Ma et al.) For this work, we consider that the models are trained for just one epoch per FL round, since the term *epoch* is never mentioned in the paper. From the graph in their Figure 8, it can be observed that the overhead introduced by the verifiable training feature is negligible for a single client (0.001 minutes per client), thanks to the efficiency of the Bulletproofs proving scheme, where only a range proof needs to be generated. However, the overhead becomes quite relevant as the number of clients increases, since a proof must be generated for each client. A range proof is also used for the result of the aggregation, resulting in an overhead to the aggregation phase of ~ 0.006 minutes. Additional overhead comes from the data proof, which can also be considered negligible, as it is simply a Merkle proof.

Federify (Keshavarzkalhori et al.) In this work, no performance times are specified; instead, the authors focus on storage requirements for ZKP generation. A significant part of the overhead to be considered likely comes from the encryption and partial decryption processes, as well as from the cost of proving these operations in ZKPs.

VerifBFL (Bellachia et al.) As in VPFL, we assume that the models in this work are trained for just one epoch per FL round. The overhead introduced during training is approximately 1.36 minutes, which is relatively small considering that what is being proved is the achievement of a certain accuracy rather than the integrity of the training procedure. The reported time required for aggregation proof generation, ~ 0.03 minutes, can likely be attributed to the efficiency of the recursive proving scheme used (i.e., Halo2), and possibly to the small size of the model. The overhead introduced by the use of the differential privacy feature can instead be considered negligible.

7 Conclusions

In this Systematization of Knowledge, we consolidated and structured the current state of the art in verifiable federated learning. Our comparative analysis of the six selected works indicates that the integration of verifiability and privacy-preserving mechanisms is pivotal in preventing dishonest behaviors and mitigating specific categories of attacks. We further observe that guaranteeing the desirable framework we have defined remains challenging and that the comparison between systems exposes a lack of common ground. Experiments vary in hardware (CPU-only vs. GPU-accelerated), in model choice (from toy networks to ResNet-50), and in the placement of the verifier (on-chain, off-chain, or hybrid). This heterogeneity makes cross-paper benchmarking difficult, and it motivates the need for a standard reporting format that covers not only proof time, but also communication bandwidth, monetary fees, and amortized per-client overheads. Despite these differences, our observations yield three main conclusions: (i) the complexity of the statement being proved dominates the overall cost: simple checks (e.g., norm bounds or range proofs) are inexpensive, whereas certifying a full gradient-descent step still incurs costs in the order of minutes to hours, regardless of the underlying proof system; (ii) auxiliary cryptographic mechanisms such as digital signatures, commitments, or authenticated channels are not asymptot-

ically expensive but nevertheless introduce substantial wall-clock overheads: in large-scale experiments, they account for delays comparable to, or even exceeding, those of the ZK prover itself; (iii) recursive composition and modular proof design demonstrably reduce per-round cost, but present benefits mainly for shallow architectures or toy models; extending these techniques to modern deep networks remains an open challenge. These limitations highlight the need for future research on strategies to reduce both computational and communication overheads, and minimize the expense of on-chain anchoring. For the latter, several measures are already deployable in the Ethereum ecosystem. First, verification can be settled on rollups that exploit EIP-4844 blobs, which provide cheaper short-lived data slots than calldata and thus lower the cost of publishing commitments. Second, large aggregates need not be posted directly: specialized data-availability layers with sampling, or restaked services such as EigenDA, allow commitments to be anchored while the bulk of the data remains off-chain but still verifiable. Third, recursive or batched proofs amortize many training rounds into a single on-chain verification, cutting both calldata and gas. Reducing onchain fees could make fully decentralized setups more plausible, potentially leveraging gossip learning. At present, the high communication overhead inherent to gossip learning would almost certainly result in substantial fees.

Finally, there is an economic dimension. Proofs will not only need to be fast; they must also be incentivized. Without careful mechanism design, rational clients may free-ride or collude. Conversely, properly designed markets for verified contributions could reward marginal improvements that are cryptographically attested, thereby aligning incentives with integrity. In summary, verifiable FL is already practical when the claims are narrow or the models small, but achieving end-to-end guarantees at realistic scales requires breakthroughs in at least three directions: collaborative aggregation proofs, proof-carrying updates that push integrity checks to the edge, and deployment on low-fee environments.

References

[1] Kasra Abbaszadeh, Christodoulos Pappas, and Jonathan Katz. "Zero-Knowledge Proofs of Training for Deep Neural Networks". In: Dec. 2024, pp. 4316–4330. DOI: 10.1145/3658644.3670316.

[2] Mojtaba Ahmadi and Reza Nourmohammadi. "zkFDL: An efficient and privacy-preserving decentralized federated learning with zero knowledge proof". In: *2024 IEEE 3rd International Conference on AI in Cybersecurity (ICAIC)*. 2024, pp. 1–10. DOI: 10.1109/ICAIC60265.2024.10433831.

[3] Thalaiyasingam Ajanthan et al. "Nesterov Method for Asynchronous Pipeline Parallel Optimization". In: *arXiv preprint arXiv:2505.01099* (2025).

[4] Martin Albrecht et al. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity". In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 191–219. ISBN: 978-3-662-53887-6.

[5] Omair Rashed Abdulwareth Almanifi, Jeevan Kanesan, et al. "Communication and computation efficiency in Federated Learning: A survey". In: *Internet of Things* 23 (2023), p. 100835. DOI: 10.1016/j.iot.2023.100835.

[6] Meriem Arbaoui et al. "Federated Learning Survey: A Multi-Level Taxonomy of Aggregation Techniques, Experimental Insights, and Future Frontiers". In: *ACM Transactions on Intelligent Systems and Technology* 15.6 (2024), 113:1–113:69. DOI: 10.1145/3678182.

[7] Li Bai et al. "Membership Inference Attacks and Defenses in Federated Learning: A Survey". In: *ACM Computing Surveys* 57.4 (2024), pp. 1–35. DOI: 10.1145/3704633.

[8] James Henry Bell et al. "Secure Single-Server Aggregation with (Poly)Logarithmic Overhead". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 1253–1269. ISBN: 9781450370899. DOI: 10.1145/3372297.3417885. URL: https://doi.org/10.1145/3372297.3417885.

[9] Ahmed Ayoub Bellachia et al. *VerifBFL: Leveraging zk-SNARKs for A Verifiable Blockchained Federated Learning*. 2025. arXiv: 2501.04319 [cs.CR]. URL: https://arxiv.org/abs/2501.04319.

[10] Paolo Bellavista, Luca Foschini, and Alessio Mora. "Decentralised Learning in Federated Deployment Environments". In: *ACM Computing Surveys* 54.1 (2021), pp. 1–38. DOI: 10.1145/3429252.

[11] Eli Ben-Sasson et al. "Fast Reed–Solomon Interactive Oracle Proofs of Proximity". In: *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Vol. 107. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, 14:1–14:17. DOI: 10.4230/LIPIcs.ICALP.2018.14. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2018.14.

[12] Eli Ben-Sasson et al. *Scalable, transparent, and post-quantum secure computational integrity*. Cryptology ePrint Archive, Paper 2018/046. 2018. URL: https://eprint.iacr.org/2018/046.

[13] Eli Ben-Sasson et al. "SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge". In: *Advances in Cryptology – CRYPTO 2013, Part II*. Vol. 8043. Lecture Notes in Computer Science. Extended version: IACR ePrint Report 2013/507. Springer, 2013, pp. 90–108. DOI: 10.1007/978-3-642-40084-1_6. URL: https://eprint.iacr.org/2013/507.

[14] Nawel Benarba and Sara Bouchenak. "Bias in Federated Learning: A Comprehensive Survey". In: *ACM Computing Surveys* 57.11 (2025), pp. 1–xx. DOI: 10.1145/3735125.

[15] Juan Benet. *IPFS - Content Addressed, Versioned, P2P File System*. 2014. arXiv: 1407.3561 [cs.NI]. URL: https://arxiv.org/abs/1407.3561.

[16] Keith Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1175–1191. ISBN: 9781450349468. DOI: 10.1145/3133956.3133982. URL: https://doi.org/10.1145/3133956.3133982.

[17] Tariq Bontekoe, Dimka Karastoyanova, and Fatih Turkmen. "Verifiability for Privacy-Preserving Computing on Distributed Data — a Survey". In: *International Journal of Information Security* 24.3 (May 2025), p. 141. ISSN: 1615-5262. DOI: 10.1007/s10207-025-01047-7. URL: https://doi.org/10.1007/s10207-025-01047-7.

[18] Sean Bowe, Jack Grigg, and Daira Hopwood. "Halo: Recursive Proof Composition without a Trusted Setup". In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 1021. URL: https://api.semanticscholar.org/CorpusID:202670380.

[19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Paper 2019/1021. 2019. URL: https://eprint.iacr.org/2019/1021.

[20] Benedikt Bünz et al. *Bulletproofs: Short Proofs for Confidential Transactions and More*. Cryptology ePrint Archive, Paper 2017/1066. 2017. URL: https://eprint.iacr.org/2017/1066.

[21] Benedikt Bünz et al. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[22] Vitalik Buterin. "A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM". In: 2015. URL: https://api.semanticscholar.org/CorpusID:19568665.

[23] Vincenzo Carletti et al. "SoK: Gradient Inversion Attacks in Federated Learning". In: *Proceedings of the 34th USENIX Security Symposium (USENIX Security 2025)*. Open Access. Seattle, WA, USA: USENIX Association, 2025, pp. 6439–6459.

[24] Di Chai et al. "A Survey for Federated Learning Evaluations: Goals and Measures". In: *IEEE Transactions on Knowledge and Data Engineering* 36.10 (2024), pp. 5007–5024. DOI: 10.1109/TKDE.2024.3382002.

[25] Jingxue Chen et al. "When Federated Learning Meets Privacy-Preserving Computation". In: *ACM Computing Surveys* 56.12 (2024), pp. 1–36. DOI: 10.1145/3679013.

[26] Yao Chen et al. " Federated Learning Attacks and Defenses: A Survey ". In: *2022 IEEE International Conference on Big Data (Big Data)*. Los Alamitos, CA, USA: IEEE Computer Society, Dec. 2022, pp. 4256–4265. DOI: 10.1109/BigData55660.2022.10020431. URL: https://doi.ieeecomputersociety.org/10.1109/BigData55660.2022.10020431.

[27] Yao Chen et al. *Federated Learning Attacks and Defenses: A Survey*. 2022. arXiv: 2211.14952 [cs.CR]. URL: https://arxiv.org/abs/2211.14952.

[28] Alessandro Chiesa et al. *Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS*. Cryptology ePrint Archive, Paper 2019/1047. 2019. URL: https://eprint.iacr.org/2019/1047.

[29] Christos Chronis et al. "A Survey on the Use of Federated Learning in Privacy-Preserving Recommender Systems". In: *IEEE Open Journal of the Computer Society* 5 (2024), pp. 227–247.

[30] Electric Coin Company. *The halo2 Book*. https://zcash.github.io/halo2/. 2023.

[31] Ivan Damgård. *On σ-protocols*. Lecture Notes, University of Aarhus, Department for Computer Science. p. 84. 2002.

[32] Thorsten Eisenhofer et al. " Verifiable and Provably Secure Machine Unlearning ". In: *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. Los Alamitos, CA, USA: IEEE Computer Society, Apr. 2025, pp. 479–496. DOI: 10.1109/SaTML64287.2025.00033. URL: https://doi.ieeecomputersociety.org/10.1109/SaTML64287.2025.00033.

[33] Hao Fang and Qian Qian. "Privacy-preserving machine learning with homomorphic encryption and federated learning". In: *Future Internet* 13.4 (2021), p. 94. DOI: 10.3390/fi13040094. URL: https://doi.org/10.3390/fi13040094.

[34] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, pp. 186–194.

[35] Matthew Fredrikson et al. "Privacy in pharmacogenetics: an end-to-end case study of personalized warfarin dosing". In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. SEC'14. San Diego, CA: USENIX Association, 2014, pp. 17–32. ISBN: 9781931971157.

[36] Ariel Gabizon and Zachary J. Williamson. *plookup: A Simplified Polynomial Protocol for Lookup Tables*. Cryptology ePrint Archive, Paper 2020/315. 2020. URL: https://eprint.iacr.org/2020/315.

[37] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Paper 2019/953. 2019. URL: https://eprint.iacr.org/2019/953.

[38] Rosario Gennaro, Craig Gentry, and Bryan Parno. "Non-interactive verifiable computing: Outsourcing computation to untrusted workers". In: *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*. Springer. 2010, pp. 465–482.

[39] Rosario Gennaro et al. "Quadratic Span Programs and Succinct NIZKs without PCPs". In: *Advances in Cryptology – EUROCRYPT 2013*. Vol. 7881. Lecture Notes in Computer Science. Springer, 2013, pp. 626–645. DOI: 10.1007/978-3-642-38348-9_37. URL: https://www.iacr.org/archive/eurocrypt2013/78810623/78810623.pdf.

[40] Robin C. Geyer, Tassilo Klein, and Moin Nabi. *Differentially Private Federated Learning: A Client Level Perspective*. 2018. arXiv: 1712.07557 [cs.CR]. URL: https://arxiv.org/abs/1712.07557.

[41] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. "Delegating computation: interactive proofs for muggles". In: *Journal of the ACM (JACM)* 62.4 (2015), pp. 1–64.

[42] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof-Systems". In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing (STOC '85)*. New York, NY, USA: Association for Computing Machinery, 1985, pp. 291–304.

[43] Shafi Goldwasser et al. "Interactive proofs for verifying machine learning". In: *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2021, pp. 41–1.

[44] Jens Groth. "On the size of pairing-based non-interactive arguments". In: *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*. Springer. 2016, pp. 305–326.

[45] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[46] István Hegedűs, Gábor Danner, and Márk Jelasity. "Decentralized learning works: An empirical comparison of gossip learning and federated learning". In: *Journal of Parallel and Distributed Computing* 148 (Feb. 2021), pp. 109–124. DOI: 10.1016/j.jpdc.2020.10.006.

[47] István Hegedűs, Gábor Danner, and Márk Jelasity. "Gossip Learning as a Decentralized Alternative to Federated Learning". In: *Lecture Notes in Computer Science*. Ed. by José Pereira and Laura Ricci. Vol. LNCS-11534. Distributed Applications and Interoperable Systems. Kongens Lyngby, Denmark: Springer International Publishing, June 2019, pp. 74–90. DOI: 10.1007/978-3-030-22496-7_5. URL: https://inria.hal.science/hal-02319574.

[48] Chenghao Hu, Jingyan Jiang, and Zhi Wang. *Decentralized Federated Learning: A Segmented Gossip Approach*. 2019. arXiv: 1908.07782 [cs.LG]. URL: https://arxiv.org/abs/1908.07782.

[49] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV]. URL: https://arxiv.org/abs/1608.06993.

[50] Zhanglong Ji, Zachary Lipton, and Charles Elkan. "Differential Privacy and Machine Learning: a Survey and Review". In: (Dec. 2014).

[51] Hengrui Jia et al. "Proof-of-learning: Definitions and practice". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 1039–1056.

[52] Weizhao Jin et al. *FedML-HE: An Efficient Homomorphic-Encryption-Based Privacy-Preserving Federated Learning System*. 2024. arXiv: 2303.10837 [cs.LG]. URL: https://arxiv.org/abs/2303.10837.

[53] Aditya Pribadi Kalapaaking et al. "SMPC-Based Federated Learning for 6G-Enabled Internet of Medical Things". In: *IEEE Network* 36.4 (2022), pp. 182–189. DOI: 10.1109/MNET.007.2100717.

[54] Sai Praneeth Karimireddy et al. “SCAFFOLD: Stochastic Controlled Averaging for Federated Learning”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 5132–5143. URL: https://proceedings.mlr.press/v119/karimireddy20a.html.

[55] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. “Constant-Size Commitments to Polynomials and Their Applications”. In: *Advances in Cryptology – ASIACRYPT 2010*. Vol. 6477. Lecture Notes in Computer Science. Springer, 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11. URL: https://www.iacr.org/archive/asiacrypt2010/6477178/6477178.pdf.

[56] D. Kempe, A. Dobra, and J. Gehrke. “Gossip-based computation of aggregate information”. In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* 2003, pp. 482–491. DOI: 10.1109/SFCS.2003.1238221.

[57] Vid Keršič, Sašo Karakatič, and Muhamed Turkanović. “On-chain Zero-Knowledge Machine Learning: An Overview and Comparison”. In: *Journal of King Saud University – Computer and Information Sciences* 36.9 (2024), p. 102207. DOI: 10.1016/j.jksuci.2024.102207. URL: https://www.sciencedirect.com/science/article/pii/S1319157824002969.

[58] Ghazaleh Keshavarzkalhori et al. “Federify: A Verifiable Federated Learning Scheme Based on zkSNARKs and Blockchain”. In: *IEEE Access* PP (Jan. 2023), pp. 1–1. DOI: 10.1109/ACCESS.2023.3347039.

[59] Jakub Konečný et al. “Federated Learning: Strategies for Improving Communication Efficiency”. In: (Oct. 2016). DOI: 10.48550/arXiv.1610.05492.

[60] Aleksei Korneev and Jan Ramon. “A Survey on Verifiable Cross-Silo Federated Learning”. In: *Transactions on Machine Learning Research* (June 2025). DOI: 10.1000/TMLR.2025.XXXX. eprint: 2410.09124. URL: https://openreview.net/forum?id=uMir8UIHST.

[61] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. “Nova: Recursive Zero-Knowledge Arguments from Folding Schemes”. In: *Advances in Cryptology – CRYPTO 2022: 42nd Annual International*

Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV. Santa Barbara, CA, USA: Springer-Verlag, 2022, pp. 359–388. ISBN: 978-3-031-15984-8. DOI: 10.1007/978-3-031-15985-5_13. URL: https://doi.org/10.1007/978-3-031-15985-5_13.

[62] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. "Nova: Recursive zero-knowledge arguments from folding schemes". In: *Annual International Cryptology Conference*. Springer. 2022, pp. 359–388.

[63] Naveen Kumar Kummari, Krishna Mohan Chalavadi, and Linga Reddy Cenkeramaddi. "The Impact of Adversarial Attacks on Federated Learning: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.5 (May 2024), pp. 2672–2691. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2023.3322785.

[64] Romain de Laage et al. "Practical Secure Aggregation by Combining Cryptography and Trusted Execution Environments". In: *Proceedings of the 19th ACM International Conference on Distributed and Event-Based Systems*. DEBS '25. Association for Computing Machinery, 2025, pp. 152–163. ISBN: 9798400713323. DOI: 10.1145/3701717.3730543. URL: https://doi.org/10.1145/3701717.3730543.

[65] Tian Li et al. *Federated Optimization in Heterogeneous Networks*. 2020. arXiv: 1812.06127 [cs.LG]. URL: https://arxiv.org/abs/1812.06127.

[66] Yue Liu et al. "A Survey on Federated Learning with Secure Multi-Party Computation". In: *Frontiers of Computer Science* (2024). DOI: 10.1007/s11704-023-3282-7.

[67] Alexander Long. *Decentralized Training Looms*. Pluralis Research. July 2024. URL: https://blog.pluralis.ai/p/decentralized-ai-looms.

[68] Tao Lu et al. "An Efficient and Extensible Zero-knowledge Proof Framework for Neural Networks". In: *ACM Conference on Computer and Communications Security (CCS)*. 2024.

[69] Carsten Lund et al. "Algebraic methods for interactive proof systems". In: *Journal of the ACM (JACM)* 39.4 (1992), pp. 859–868.

[70] Juan Ma et al. “VPFL: Enabling verifiability and privacy in federated learning with zero-knowledge proofs”. In: *Knowledge-Based Systems* 299 (June 2024), p. 112115. DOI: 10.1016/j.knosys.2024.112115.

[71] H. B. McMahan et al. “Communication-Efficient Learning of Deep Networks from Decentralized Data”. In: *International Conference on Artificial Intelligence and Statistics*. 2016. URL: https://api.semanticscholar.org/CorpusID:14955348.

[72] J Josepha Menandas and Mary Subaja Christo. “Analysis of various homomorphic encryption algorithms based on primitive functions and applications”. In: *OPSEARCH* (2025), pp. 1–27.

[73] Fan Mo, Zahra Tarkhani, and Hamed Haddadi. “Machine Learning with Confidential Computing: A Systematization of Knowledge”. In: *ACM Computing Surveys* 56.11 (2024), 281:1–281:40. DOI: 10.1145/3670007. URL: https://doi.org/10.1145/3670007.

[74] Fan Mo et al. “PPFL: privacy-preserving federated learning with trusted execution environments”. In: *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys ’21. Virtual Event, Wisconsin: Association for Computing Machinery, 2021, pp. 94–108. ISBN: 9781450384438. DOI: 10.1145/3458864.3466628. URL: https://doi.org/10.1145/3458864.3466628.

[75] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: *Cryptography Mailing list at https://metzdowd.com* (Mar. 2009).

[76] Ahmed El Ouadrhiri and Ahmed Abdelhadi. “Differential Privacy for Deep and Federated Learning: A Survey”. In: *IEEE Access* 10 (2022), pp. 22359–22380. DOI: 10.1109/ACCESS.2022.3151670.

[77] Bryan Parno et al. “Pinocchio: Nearly Practical Verifiable Computation”. In: *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 238–252. DOI: 10.1109/SP.2013.47.

[78] Torben Pryds Pedersen. “A Threshold Cryptosystem without a Trusted Party”. In: *Advances in Cryptology — EUROCRYPT ’91*. Ed. by Donald W. Davies. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 522–526. ISBN: 978-3-540-46416-7.

[79] Torben Pryds Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology — CRYPTO '91*. Ed. by Joan Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 129–140. ISBN: 978-3-540-46766-3.

[80] Zhizhi Peng et al. *A Survey of Zero-Knowledge Proof Based Verifiable Machine Learning*. 2025. arXiv: 2502.18535 [cs.CR]. URL: https://arxiv.org/abs/2502.18535.

[81] Polygon Zero Team. *Plonky2: Fast Recursive Arguments with PLONK and FRI*. Tech. rep. https://docs.rs/crate/plonky2/latest/source/plonky2.pdf. Polygon Zero, Sept. 2022.

[82] Attia Qammar et al. "Securing federated learning with blockchain: a systematic literature review". In: *Artificial Intelligence Review* 56.5 (2023), pp. 3951–3985. DOI: 10.1007/s10462-022-10271-9.

[83] Pian Qi et al. "Model aggregation techniques in federated learning: A comprehensive survey". In: *Future Generation Computer Systems* 150 (2024), pp. 272–293. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2023.09.008. URL: https://www.sciencedirect.com/science/article/pii/S0167739X23003333.

[84] Taki Hasan Rafi et al. "Fairness and privacy preserving in federated learning: A survey". In: *Information Fusion* 105 (2024), p. 102198. DOI: 10.1016/j.inffus.2023.102198.

[85] Sameera Ramasinghe et al. "Protocol Models: Scaling Decentralized Training with Communication-Efficient Model Parallelism". In: *arXiv preprint arXiv:2506.01260* (2025).

[86] Srinath Setty. *Spartan: Efficient and general-purpose zkSNARKs without trusted setup*. Cryptology ePrint Archive, Paper 2019/550. 2019. URL: https://eprint.iacr.org/2019/550.

[87] Geetanjli Sharma et al. "SoK: Systematizing Attack Studies in Federated Learning – From Sparseness to Completeness". In: *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIACCS 2023)*. Association for Computing Machinery, 2023, pp. 579–592. DOI: 10.1145/3579856.3590328.

[88] Reza Shokri et al. " Membership Inference Attacks Against Machine Learning Models ". In: *2017 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2017, pp. 3–18. DOI: 10.1109/SP.2017.41. URL: https://doi.ieeecomputersociety.org/10.1109/SP.2017.41.

[89] Ilia Shumailov et al. "Trusted Machine Learning Models Unlock Private Inference for Problems Currently Infeasible with Cryptography". In: *arXiv preprint* arXiv:2501.08970 (2025). URL: https://arxiv.org/abs/2501.08970.

[90] Hira Sikandar et al. "A Detailed Survey on Federated Learning Attacks and Defenses". In: *Electronics* 12 (Jan. 2023), p. 260. DOI: 10.3390/electronics12020260.

[91] Haochen Sun et al. *zkDL: Efficient Zero-Knowledge Proofs of Deep Learning Training*. 2023. arXiv: 2307.16273 [cs.LG]. URL: https://arxiv.org/abs/2307.16273.

[92] Nick Szabo. "Formalizing and Securing Relationships on Public Networks". In: *First Monday* 2 (1997). URL: https://api.semanticscholar.org/CorpusID:33773111.

[93] Alysa Ziying Tan et al. "Towards Personalized Federated Learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 34.12 (Dec. 2023), pp. 9587–9603. ISSN: 2162-237X. DOI: 10.1109/TNNLS.2022.3160699.

[94] Vale Tolpegin et al. "Data Poisoning Attacks Against Federated Learning Systems". In: *European Symposium on Research in Computer Security*. 2020. URL: https://api.semanticscholar.org/CorpusID:220546077.

[95] Florian Tramèr et al. "Stealing machine learning models via prediction APIs". In: *Proceedings of the 25th USENIX Conference on Security Symposium*. SEC'16. Austin, TX, USA: USENIX Association, 2016, pp. 601–618. ISBN: 9781931971324.

[96] Paul Valiant. "Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency". In: *Theory of Cryptography*. Ed. by Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–18. ISBN: 978-3-540-78524-8.

[97] Suppakit Waiwitlikhit et al. *Trustless Audits without Revealing Data or Models*. 2024. arXiv: 2404.04500 [cs.CR]. URL: https://arxiv.org/abs/2404.04500.

[98] Jianyu Wang et al. *Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization*. 2020. arXiv: 2007.07481 [cs.LG]. URL: https://arxiv.org/abs/2007.07481.

[99] Zhipeng Wang et al. "zkFLzkFL: Zero-Knowledge Proof-Based Gradient Aggregation for Federated Learning". In: *IEEE Transactions on Big Data* 11.2 (2025), pp. 447–460. DOI: 10.1109/TBDATA.2024.3403370.

[100] Kang Wei et al. *Federated Learning with Differential Privacy: Algorithms and Performance Analysis*. 2019. arXiv: 1911.00222 [cs.LG]. URL: https://arxiv.org/abs/1911.00222.

[101] Herbert Woisetschläger et al. "Federated Learning and AI Regulation in the European Union: Who is Responsible? – An Interdisciplinary Analysis". In: *Proceedings of the GenLaw'24 Workshop at the 41st International Conference on Machine Learning (ICML)*. Vol. 235. Proceedings of Machine Learning Research. Presented at GenLaw 2024. Vienna, Austria, 2024. arXiv: 2407.08105 [cs.AI].

[102] Zhibo Xing et al. *Zero-Knowledge Proof-based Practical Federated Learning on Blockchain*. 2023. arXiv: 2304.05590 [cs.CR]. URL: https://arxiv.org/abs/2304.05590.

[103] Zhibo Xing et al. "Zero-Knowledge Proof-Based Verifiable Decentralized Machine Learning in Communication Networks: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* (2025). Early Access. ISSN: 1553-877X. DOI: 10.1109/COMST.2025.3561657. URL: https://ieeexplore.ieee.org/document/3561657.

[104] Yi Xu et al. "Non-interactive verifiable privacy-preserving federated learning". In: *Future Gener. Comput. Syst.* 128 (2022), pp. 365–380. URL: https://doi.org/10.1016/j.future.2021.10.017.

[105] Xuefei Yin, Yanming Zhu, and Jiankun Hu. "A Comprehensive Survey of Privacy-preserving Federated Learning: A Taxonomy, Review, and Future Directions". In: *ACM Computing Surveys* 54.6 (2022), 131:1–131:36. DOI: 10.1145/3460427.

[106] Liangqi Yuan et al. “Decentralized Federated Learning: A Survey and Perspective”. In: *IEEE Internet of Things Journal* 11.21 (2024), pp. 34617–34638. DOI: 10.1109/JIOT.2024.3407584.

[107] Arantxa Zapico et al. *Baloo: Nearly Optimal Lookup Arguments*. Cryptology ePrint Archive, Paper 2022/1565. 2022. URL: https://eprint.iacr.org/2022/1565.

[108] Weishan Zhang et al. “A Survey on Federated Learning: Challenges, Methods, and Applications”. In: *International Journal of Machine Learning and Cybernetics* 14.2 (2023), pp. 513–535. DOI: 10.1007/s13042-022-01647-y.