SoK: Verifiable Integrity Claims for Privacy-Preserving Federated Learning

Andrea Rizzini
Politecnico di Milano, Horizen Labs
andrea.rizzini@polimi.it

Marco Esposito
Politecnico di Milano
marco.esposito@polimi.it

Tommaso Gagliardoni
Horizen Labs
tomgag@horizenlabs.io

Francesco Bruschi
Politecnico di Milano
francesco.bruschi@polimi.it

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. This misbehavior, in turn, also threatens privacy, because an undetected deviation in training or aggregation can silently undermine the confidentiality guarantees that FL was designed to provide. This motivates mechanisms that provide checkable evidence that released checkpoints are consistent with a declared learning specification and an auditable execution trace. In this SoK, we model federated learning as an append-only transcript of submissions, admissions, aggregation, and finalization events, and formalize verifiability as a collection of integrity claims issued by clients and the aggregator, and checked by different verifier classes. We derive a taxonomy of recurring client-side and aggregator-side claims and use it to analyze representative verifiable FL (VFL) systems spanning Zero-Knowledge Proofs (ZKP) and Trusted Execution Environment (TEE) technologies. Our analysis suggests that, while verifiable aggregation is comparatively mature, data verifiability appears feasible but still sparsely adopted in practice, and verifiable training remain costly and rarely scale to modern models.

Keywords

Federated learning, privacy-preserving machine learning, secure aggregation, verifiable computation, auditability

1 Introduction

Machine Learning (ML) has seen tremendous growth over the past two decades and is now one of the most actively researched areas in computer science [9]. Its success has largely depended on the quality of training data, reflecting the well-known principle of "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 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 Federated Learning (FL) approach was introduced by Google in 2016 [57]. Over time, several limitations of this approach began to emerge. In particular, the model updates shared by clients may leak information about their local datasets, and an untrusted server may infer sensitive data [33]. On the other hand, it was also soon recognized that clients can behave Byzantine and submit poisoned model updates that disrupt convergence or implant targeted backdoors [35]. These limitations point to the familiar concerns of protecting client data and handling malicious participants. This is evident in the large body of work surveying FL threats and defenses whereby one anticipates deviations and evaluates their effects under selected attacks. Such research will arguably become increasingly critical, as low-budget and low-visibility attacks can still inflict disproportionate damage in practical deployments [60].

In practice, FL deployments are therefore assessed primarily through outcomes, like accuracy and convergence, resource costs, and robustness or leakage under chosen threat models [26]. Yet, metrics based on outcome provide little use for dispute resolution or auditing, when the learning workflow occurs among mutually distrustful organizations. This issue is exacerbated by privacy-preserving deployments, which can complicate regulatory oversight. For instance, under the EU AI Act and GDPR, organizations deploying FL must demonstrate not only that data stays local, but that the training process itself complied with declared policies, a requirement that pure privacy mechanisms cannot satisfy alone [94, 102]. Moreover, recent theoretical results show that post-hoc detection is not a reliable substitute for integrity guarantees. It was shown that an untrusted learner can plant computationally undetectable backdoors, such that the resulting model can be indistinguishable from an honestly trained one to any efficient detector [43]. This suggests that accountability mechanisms for FL should provide portable, checkable evidence that binds the released checkpoints to a specified transcript of local training and aggregation. On this matter, the evolving field of verifiable computation has recently found applications in ML, and by extension to FL settings as shown by the several works that we and others surveyed [104]. More broadly, this notion of verifiability is distinct from validating model quality, which can often be assessed efficiently by standard evaluation [45]. Verifiability instead concerns *integrity claims about the learning*

Proceedings on Privacy Enhancing Technologies 2026(4), 248–271

https://doi.org/10.56553/popets-2026-0119

process itself, namely whether the final results are consistent with that prescribed computation. Importantly, mechanisms for such integrity claims need not be exclusively cryptographic, but evidence may be instantiated also via hardware-backed attestations or recording of an auditable training history, as commonly proposed in systems such as blockchain-assisted FL [84].

Contributions. We organize in an organic way existing terminology, properties, threat models, and state of the art results in the field of verifiable FL (VFL). Moreover, we propose a novel classification framework to systematize VFL schemes according to their security and privacy properties. 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). As for verifiability mechanisms, we restrict our scope to systems instantiated via *Zero-Knowledge Proofs* and *Trusted Execution Environments, as these are the most established primitives for verifiable computation in the literature [72, 81] and the only two that produce verifiable evidence artifacts in the formal sense of Section 3.*

These scoping decisions are motivated by the need for a focused and comparable analysis of verifiability in high-stakes, privacy-sensitive deployments (e.g., healthcare, finance), where the core accountability questions concern the correctness and compliance of training and aggregation. 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.

2 Preliminaries

Federated Learning. The high-level idea of FL consists in a set of 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 as follows:

(1) Each client C_i at round t trains a model locally on its own dataset D_i, obtaining $\theta_{i,t}$, $t \in [1, T]$. The number of rounds T is usually not fixed in advance.
(2) Each client C_i sends its local model $\theta_{i,t}$ to an aggregator.
(3) The aggregation is performed, producing a global model $\theta_{\text{global},t}$.
(4) The global model is then distributed back to each client.

Steps 1 to 4 (Figure 1) are repeated until a stopping criterion is met.

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. Aggregation, Step 3, is the process that, at iteration t,

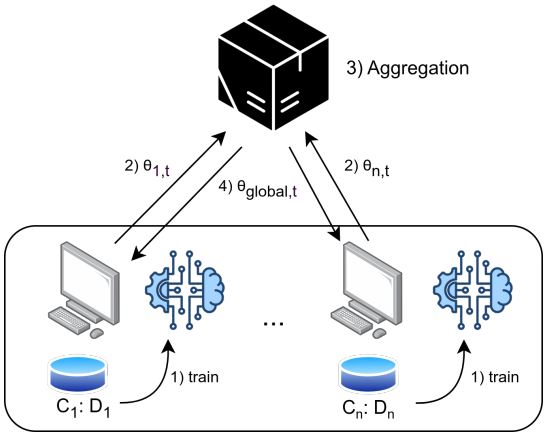

Figure 1: Basic FL setup considering aggregation as a black box

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.

We can distinguish between different algorithms for this step[83], such as FedAvg [69], FedProx [63], FedNova [97], Scaffold [53]. While FL preserves data locality, model updates may still leak sensitive information. To mitigate this risk, secure aggregation protocols aim to protect individual contributions while preserving correctness of $\theta_{\text{global},t}$ [54]. Existing approaches include: *Masking*, where each client perturbs its update as $\theta_{i,t}^{\text{masked}} = \theta_{i,t} + s_{i,t}$ and masks cancel out during aggregation [12]; *Homomorphic Encryption*, which enables aggregation directly over encrypted updates [34, 50, 70]; *Multi-Party Computation*, allowing clients to jointly compute the aggregate without revealing individual updates [18, 52, 106] — a systematic comparison of these and related cryptographic approaches in the FL setting can be found in [67]; and *TEE-based* approaches (see 4), where aggregation runs inside a trusted enclave and only $\theta_{\text{global},t}$ is released [32, 71]. Another widely used privacy-preserving technique is Differential Privacy (DP), where noise is added to data or updates to limit information leakage [49, 100]. In FL, noise is typically injected into client updates or into the global model, so aggregation is performed over perturbed values. Unlike secure aggregation, this noise does not cancel out, making exact recovery of the clean aggregate impossible and generally trading privacy for reduced model accuracy.

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 (e.g., polylogarithmic) than redoing the job [41]. 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 [44].

General-purpose zero-knowledge proofs prove the correctness of arbitrary computations and can be made *zero knowledge* via standard blinding techniques. Their design space is mainly shaped by setup and arithmetisation. The former concerns how public parameters are generated: from circuit-specific trusted setups [46, 77], to

updatable ones [21, 28, 38], up to fully transparent systems at the cost of larger proofs [15, 22, 87]. Arithmetisation determines how computation is encoded algebraically: from classic R1CS and QAPs [16, 42], to Plonk-style table constraints [38], to trace-based AIR formulations enabling quasi-linear-time proving [15], and modern hybrids combining constraints with lookup tables for efficiency [37, 109]. Overall, different designs trade off prover cost, proof size, and transparency.

Blockchain and Smart Contracts. Blockchain is a distributed ledger organized as an append-only chain of blocks, first popularized by Bitcoin [74]. New blocks of transactions are added via a consensus protocol (commonly Proof of Work or Proof of Stake) rather than by a central authority. Different systems can be categorized as permissionless (public) or permissioned (private). Smart Contracts are self-executing programs, originally envisioned by Szabo [92] and popularized by platforms such as Ethereum. Once deployed, they execute transparently and are typically immutable, making them useful for verifiable processes, where interaction with off-chain data is commonly mediated via oracle networks.

Trusted Execution Environments. A TEE, also called a secure enclave, is a hardware-backed execution context that aims to preserve the confidentiality and integrity of a protected computation even when the surrounding system software is potentially malicious. TEEs are now widely deployed across device classes and are often used as a pragmatic alternative to cryptographic approaches [72, 73, 113]. TEEs provide an isolated execution environment (e.g., enclaves or protected VMs) with a small trusted computing base rooted in the CPU and firmware, although recent work shows that architectural and microarchitectural attacks often weaken these guarantees [73, 113]. To externalize these guarantees, remote attestation allows a verifier to validate that a code image is running inside a genuine TEE via an attestation token (i.e., quote). Let $m \leftarrow H(\mathsf{code} \,\|\, \mathsf{config} \,\|\, \mathsf{init})$ be the measurement of the enclave's state. The quote is calculated as $q = \mathrm{Sign}(sk_{\mathsf{TEE}}, m \,\|\, \mathsf{nonce} \,\|\, \mathsf{ud})$. Here, ud binds optional user data (e.g., a public key), and nonce is challenge supplied by the verifier to prevent replay attacks. The verifier checks that $\mathrm{Verify}(pk_{\mathsf{TEE}}, q) = 1$ and that m matches the expected reference value. We denote q_i as the quote for enclave i and $\sigma_{i,t}$ as the specific attestation instance at round t.

3 A Framework for Verifiable FL

We distinguish three main application classes that characterize how verifiability manifests in FL:

(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, whereas (3) is specific to FL, where provenance and correctness are fundamentally multi-party due to distributed trust and partial observability. From these application classes we can gather different set of challenges:

- **Verifiable data usage**: *How can clients commit to data and policies so that audits remain meaningful over time, including compliance with "right to be forgotten" requirements and data minimization obligations under privacy regulations?*
- **Verifiable training algorithm**: *How can clients prove that updates follow the declared training procedure (i.e. architecture, loss, optimizer, and hyperparameters) without revealing private data or relying on trusted devices?*
- **Verifiable (secure) aggregation**: *How can the server prove correct aggregation over the declared updates, and how can auditors detect equivocation without breaking confidentiality?*

A common thread is that these challenges motivate keeping an auditable execution trace of submissions, admissions, and aggregations across rounds. However, auditability alone is insufficient to enable verification. What is still missing are concrete guarantees that the recorded actions truly comply with the declared training and aggregation rules. Hence, we must make explicit which actors produce evidence, for which claims, and under what privacy constraints.

3.1 System Model

The following system model illustrates (i) which entities participate in an FL execution, (ii) what information is recorded as an execution transcript, and (iii) what it means for a party to provide evidence that some recorded action satisfies a declared rule. We define the roles below, where entities may play multiple roles:

- *Data owner (DO).* The party that controls a dataset and associated usage policy (e.g., consent, sampling, preprocessing constraints). A data owner may delegate computation to a separate trainer, or coincide with the client.
- *Client (CL).* A party that performs local training and produces a candidate contribution for a training round, such as a model update or gradient.
- *Aggregator (AG).* The party that combines client contributions to form a global update and produces a new checkpoint. This may be a centralized server, a committee, a secure enclave service, or smart contract logic.
- *Transcript provider (TP).* A functionality that records an append-only trace of relevant events and makes them retrievable. This may be instantiated as a public blockchain, a permissioned ledger, or any authenticated log.
- *Verifier (V).* Any party that checks evidence. Verifiers can include the aggregator (online admission), participating clients (cross-checks), external auditors (compliance), or fully public verifiability.

We write $\mathcal{V}$ for the verifier class (e.g., $\mathcal{V} = \{\mathsf{AG}\}, \{\mathsf{AG}, \mathsf{CL}\}, \{\mathsf{auditor}\}$, or $\{\mathsf{public}\}$). Distinguishing verifier classes is essential because some protocols provide evidence that is checkable only under designated root of trust (e.g., platform attestation keys), rather than

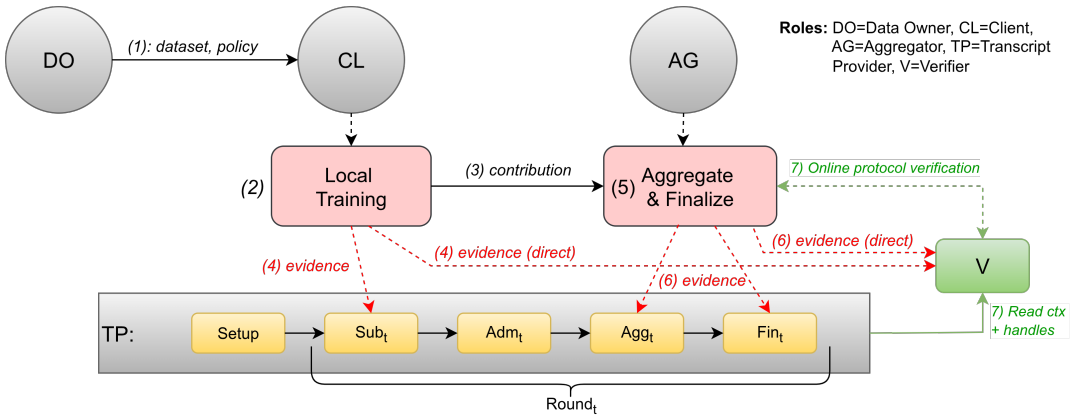

Figure 2: System model for verifiable federated learning. Data owners (DO) provide datasets and usage policies to clients (CL), who perform local training and submit contributions to the aggregator (AG). Evidence artifacts (dashed red arrows) may be recorded in an append-only transcript maintained by the transcript provider (TP), or delivered directly to verifiers (designated verification). TP logs the lifecycle of each round (i.e., submissions (Sub_t), admissions (Adm_t), aggregation (Agg_t), and finalization (Fin_t)) and may store full evidence or only handles to off-chain artifacts. Verifiers (V) read context and handles from TP to bind checks to ctx_t, and can check evidence either during protocol execution (dashed green arrow) or *ex post* over the recorded transcript (solid green arrow).

by arbitrary third parties. An FL execution proceeds over rounds $t \in \{1, \ldots, T\}$ and is governed by a declared specification Spec, which may include: (i) the model architecture identifier and initialization, (ii) the local training rule (loss, optimizer, hyperparameters, number of epochs), (iii) the admissibility policy for contributions, (iv) the aggregation rule, and (v) optional privacy mechanisms (e.g., secure aggregation, DP noise addition). We treat Spec as a reference object that evidence statements can refer to. When privacy mechanisms hide raw contributions (e.g., via secure aggregation), the transcript may record only commitments to off-chain data, which will affect verifiability statements accordingly.

Transcript model. We model an execution by an append-only transcript Tr consisting of a setup record followed by per-round records:

$$\mathsf{Tr} := \big(\mathsf{Setup},\ \mathsf{Round}_1,\ \ldots,\ \mathsf{Round}_T\big).$$

The setup record includes any public parameters and the initial checkpoint:

$$\mathsf{Setup} := (\mathsf{pp},\ \mathsf{Spec},\ \mathsf{ckpt}_0).$$

Here, pp may include both static public parameters and, when required, digests of the setup procedure that generated them. Typical examples in our surveyed papers include Distributed Key Generation (DKG) transcripts for threshold or HE-based aggregation [20], CRS identifiers and verification keys for proof systems [2, 13], and attestation bootstrap records that fix accepted enclave measurements and trust anchors [51, 61, 110]. The element ckpt_0 denotes either the initial global model state or a commitment to it. Each round record summarizes the lifecycle of client contributions $\mathsf{Round}_t := \big(\mathsf{tag}_t,\ \mathsf{Sub}_t,\ \mathsf{Adm}_t,\ \mathsf{Agg}_t,\ \mathsf{Fin}_t\big)$, where:

- tag_t is a round identifier that fixes the context for replay prevention.
- Sub_t is a set of submissions. A submission from client i is represented as $\mathsf{sub}_{i,t} := (i,\ h_{i,t},\ m_{i,t})$, where $h_{i,t}$ is a handle to the contribution (e.g., a plaintext update, a commitment, or a ciphertext under secure aggregation) and $m_{i,t}$ is optional metadata required by the declared protocol (e.g., weights, client identifiers, signatures).
- Adm_t records the admitted set and any admission metadata, e.g., $\mathsf{Adm}_t := (S_t,\ \mathsf{Spec.policy})$, where $S_t \subseteq \{1, \ldots, n\}$ denotes the indices admitted for aggregation.
- Agg_t records the aggregation output and any auxiliary data needed to interpret it (e.g., aggregation weights): $\mathsf{Agg}_t := (h_t^{\mathsf{agg}},\ \omega_t)$.
- Fin_t records the finalized checkpoint ckpt_t or commitment and its linkage to the transcript, e.g., via inclusion proofs or log references.

Next, we model a verifiability mechanism as producing an evidence artifact, denoted by e, that supports a claim about some transcript event. Formally, a claim is a predicate $\Phi(x) = 1$, where x is a statement derived from publicly available inputs, such as $(\mathsf{Spec}, \mathsf{ckpt}_{t-1}, i, h_{i,t}, \mathsf{tag}_t)$ for a client step, or $(\mathsf{Spec}, S_t, \{h_{i,t}\}_{i \in S_t}, \mathsf{ckpt}_t)$ for an aggregation step. A procedure then outputs e using private information (i.e., a witness) unavailable to verifiers; while verification is an efficient, deterministic check: $\mathsf{Verify}(\mathsf{pp}, x, e) \in \{0, 1\}$.

We require evidence to be bound to the relevant transcript context ctx_t, which consolidates all public information needed to verify round-t actions $\mathsf{ctx}_t := (\mathsf{pp}, \mathsf{Spec}, \mathsf{ckpt}_{t-1}, \mathsf{tag}_t)$, where ckpt_{t-1} and tag_t link the round to the transcript prefix. Concretely, the public statement x must include the round identifier and the referenced transcript commitments, so that replaying e across rounds or across different transcript instances is detectable. This abstraction intentionally subsumes both cryptographic evidence and hardware evidence. Finally, we distinguish between *online verification*, where Verify is executed as part of protocol control flow, and *ex post verification*, where Verify is run after the fact over the recorded transcript.

A graphical depiction of the proposed framework is presented in Figure 2.

3.2 Black-box Interfaces and Claim Taxonomy

Building on the roles and transcript objects, we summarize the minimal black-box interfaces used throughout this survey in Appendix A.1. We then instantiate the abstract predicate view introduced above by defining two families of claims: (i) *client-side claims*, which justify admitting a submission into Adm_t; and (ii) *aggregator-side claims*, which justify the aggregation output recorded in Agg_t and the finalized checkpoint ckpt_t in Fin_t. In both cases, a claim is a predicate over public transcript-bound inputs (derived from ctx_t and recorded handles), and is supported by an evidence artifact verified via the corresponding $\mathcal{F}_{\mathsf{V}}$ interface.

For each client submission $\mathsf{sub}_{i,t} = (i, h_{i,t}, m_{i,t})$, the associated public statement is

$$x_{i,t}^{\mathsf{CL}} := (\mathsf{ctx}_t, i, h_i^D, h_{i,t}, m_{i,t}),$$

where ctx_t denotes the public round context derived from the transcript, and h_i^D is a public handle to client i's dataset under the declared policy (e.g., a vector commitment, registry pointer, or TEE attestation reference). For aggregation, the public statement is

$$x_t^{\mathsf{AG}} := (\mathsf{ctx}_t, S_t, \{h_{i,t}\}_{i\in S_t}, h_t^{\mathsf{agg}}, \mathsf{ckpt}_t),$$

where h_t^{agg} denotes a public handle to the aggregation result. Evidence $e_{i,t}^{\mathsf{CL}}$ is accepted if $\mathcal{F}_{\mathsf{V}}.\mathsf{VerifyClient}(x_{i,t}^{\mathsf{CL}}, e_{i,t}^{\mathsf{CL}}) = 1$, evidence e_t^{AG} is accepted if $\mathcal{F}_{\mathsf{V}}.\mathsf{VerifyAgg}(x_t^{\mathsf{AG}}, e_t^{\mathsf{AG}}) = 1$.

We next detail the main claim types used throughout this survey.

Client-side claims. These claims justify admitting a recorded submission into Adm_t. We group them into four recurring types:

- **(CL1) Data binding.** The client submission is associated with a dataset that is consistent with the previously published handle $h_i^D = \mathcal{F}_{\mathsf{DB}}.\mathsf{Bind}(D_i, \mathsf{PolicySpec})$.
- **(CL2) Local training correctness.** The submitted handle $h_{i,t}$ corresponds to an update computed from the prior checkpoint in ctx_t by executing the declared local rule in Spec, on data consistent with h_i^D, producing $e_{i,t}^{\mathsf{CL}}$.
- **(CL3) Update admissibility.** The submission satisfies declared constraints used for admission (e.g., shape checks, norm bounds, clipping ranges, or differential privacy noise addition). This claim is often weaker than full proof-of-training, but is sufficient for mechanisms that reject malformed or out-of-policy updates before aggregation, and may contribute to mitigating certain gradient-based inference or poisoning attacks.
- **(CL4) Uniqueness.** Evidence is bound to ctx_t (in particular tag_t and ckpt_{t-1}), so that reusing an old $(h_{i,t}, e_{i,t}^{\mathsf{CL}})$ across rounds or transcripts is detectable.

To connect these claims to actual FL training, Algorithm 1 restates a verifiable local step in a way that matches $\mathcal{F}_{\mathsf{CL}}.\mathsf{Train}$ and produces evidence verifiable via $\mathcal{F}_{\mathsf{V}}.\mathsf{VerifyClient}$. Many (non-)membership statements can be expressed by choosing the granularity at which data are committed and referenced in the client evidence. For intuition, if $\bar{x}$ is a data item to be excluded, one can enforce constraints of the following form inside the evidence generation for the local step: (i) BGD: a single commitment to the (full) batch and a non-membership condition; (ii) SGD: per-sample commitments and $b_i \neq \bar{x}$ for each committed item; (iii) mini-batch: per-mini-batch commitments and non-membership within each batch. These constraints can further be realized in zero-knowledge, so that batches remain private.

Algorithm 1 Verifiable client local step (round t)

Input:
Private: Dataset D_i
Public: ctx_t (pp, Spec, ckpt_{t-1}, tag_t); own data handle h_i^D
Output: submission $\mathsf{sub}_{i,t} = (i, h_{i,t}, m_{i,t})$; client evidence $e_{i,t}^{\mathsf{CL}}$

$\theta_{i,t} \leftarrow \mathcal{F}_{\mathsf{CL}}.\mathsf{Train}(D_i, \mathsf{ctx}_t)$ ▷ batching strategy per Spec
$h_{i,t} \leftarrow \mathsf{Encode}(\theta_{i,t}, \mathsf{Spec.encoding})$ ▷ plaintext, commitment, or ciphertext
$m_{i,t} \leftarrow \mathsf{DeriveMetadata}(D_i, \mathsf{Spec})$ ▷ e.g., $|D_i|$ for weighted aggregation
$e_{i,t}^{\mathsf{CL}} \leftarrow$ evidence artifact for $x_{i,t}^{\mathsf{CL}}$
$\mathsf{sub}_{i,t} \leftarrow (i, h_{i,t}, m_{i,t})$
Send $\mathsf{sub}_{i,t}$ and $e_{i,t}^{\mathsf{CL}}$ to AG
$\mathcal{F}_{\mathsf{TP}}.\mathsf{Append}(\mathsf{tag}_t, \mathsf{sub}_{i,t}, e_{i,t}^{\mathsf{CL}})$ ▷ independent record for auditability
return $(\mathsf{sub}_{i,t}, e_{i,t}^{\mathsf{CL}})$

Algorithm 2 Verifiable aggregation (round t)

Input:
ctx_t (pp, Spec, ckpt_{t-1}, tag_t); registered handles $\{h_i^D\}_{i\in[n]}$
$\mathsf{Sub}_t = \{(i, h_{i,t}, m_{i,t})\}$
$\{e_{i,t}^{\mathsf{CL}}\}$

Output: $\mathsf{Adm}_t, \mathsf{Agg}_t, \mathsf{Fin}_t; e_t^{\mathsf{AG}}$

$S_t \leftarrow \emptyset$
for each $(i, h_{i,t}, m_{i,t}) \in \mathsf{Sub}_t$ **do**
  if $\mathcal{F}_{\mathsf{V}}.\mathsf{VerifyClient}(x_{i,t}^{\mathsf{CL}}, e_{i,t}^{\mathsf{CL}}) = 1$ **then**
    $S_t \leftarrow S_t \cup \{i\}$
  end if
end for
$\mathsf{Adm}_t \leftarrow (S_t,$ Spec.policy)
Derive aggregation weights ω_t from Spec and $\{m_{i,t}\}_{i\in S_t}$
$(h_t^{\mathsf{agg}}, \mathsf{ckpt}_t) \leftarrow \mathcal{F}_{\mathsf{AG}}.\mathsf{Aggregate}(\mathsf{ctx}_t, \{h_{i,t}\}_{i\in S_t}, \omega_t)$
$\mathsf{Agg}_t \leftarrow (h_t^{\mathsf{agg}}, \omega_t)$
$\mathsf{Fin}_t \leftarrow \mathsf{ckpt}_t$ ▷ or commitment thereof
$e_t^{\mathsf{AG}} \leftarrow$ evidence artifact for x_t^{AG}, or $\perp$ if implicit
$\mathcal{F}_{\mathsf{TP}}.\mathsf{Append}(\mathsf{tag}_t, \mathsf{Sub}_t, \mathsf{Adm}_t, \mathsf{Agg}_t, \mathsf{Fin}_t)$
return $(\mathsf{Adm}_t, \mathsf{Agg}_t, \mathsf{Fin}_t, e_t^{\mathsf{AG}})$

Aggregator-side claims. Aggregator-side verifiability is about justifying that the recorded aggregation output and checkpoint are consistent with the declared rule and the admitted set. We again group the recurring claims:

- **(AG1) Correct admission set.** The admitted set S_t recorded in Adm_t matches the application of $\mathcal{F}_{\mathsf{V}}.\mathsf{VerifyClient}$ to submitted evidence with any declared admission policy tags. Namely, the aggregator neither adds nor omits submissions relative to the declared verification rule.

- **(AG2) Correct aggregation and checkpoint update.** The aggregator output handle h_t^{agg} are consistent with applying the declared aggregation rule in Spec to exactly the admitted submissions $\{h_{i,t}\}_{i \in S_t}$, and updating from ckpt_{t-1} as specified.
- **(AG3) Non-equivocation.** The aggregator cannot present different aggregates to different verifiers (or at different times) without detection. This can be enforced by binding h_t^{agg} or ckpt_t into the shared transcript via $\mathcal{F}_{\text{TP}}$.Append, so all verifiers read the same finalized record.

Algorithm 2 restates verifiable aggregation in a way that matches $\mathcal{F}_{\text{AG}}$.Aggregate and produces evidence verifiable via $\mathcal{F}_{\text{V}}$.VerifyAgg, while abstracting over on-chain and off-chain deployment patterns. In the former case, a smart contract performs the verification of $e_{i,t}^{\text{CL}}$ and the aggregation itself, so that $e_t^{\text{AG}} = \perp$ and non-equivocation follows directly from the public transcript. In the latter, aggregation is run off-chain and the aggregator publishes an evidence artifact e_t^{AG} that is checkable by the intended verifier class $\mathcal{V}$.

Relations among claims. Since the seven claims above are not at the same level of strength, it is important to clarify how they relate. Among the client-side claims, CL1 simply constrains h_i^D and certifies that it was produced by a function $\mathcal{F}_{\text{DB}}$.Bind under PolicySpec (Appendix A.1). The only relational claim is CL2 as it ties $h_{i,t}$ to ckpt_{t-1} and to data consistent with h_i^D through Spec. As a consequence, it can compose with CL1 into the natural guarantee that the recorded update was produced by the declared rule on the registered dataset, but taken on its own CL2 leaves h_i^D unattested unless CL1 also holds. Similarly, whether CL2 also entails CL3 depends on Spec, for example when clipping or DP noise belong to Spec. Otherwise, CL2 and CL3 certify different aspects, since a vector that satisfies the constraints can always be sampled without performing any training, as is clear from schemes that verify only structural predicates on the submitted vector like in [11]. In the same way, CL1 and CL3 are independent, as a bound dataset handle implies nothing about the submitted bytes and an admissible update carries no data provenance. CL4, unlike the rest, does not concern what the evidence asserts but how it is connected to the entire context ctx_t, hence it can be combined with any of the above. A similar reading applies to the aggregator side, where we have a collapse between AG1 and AG2 only if the aggregation proof internally re-runs verification on each input, otherwise these remain independent claims. Because aggregator claims effectively inherit whichever client-side predicates are enforced by $\mathcal{F}_{\text{V}}$.VerifyClient, removing either side weakens security guarantees in asymmetric ways. For example, without the aggregator side, even strong client evidence can still be recombined incorrectly before reaching the checkpoint (Appendix A.2).

4 Verifiable FL State-of-the-Art Solutions

We provide a systematization of the field based on 13 representative research papers. These contributions were selected both for the relevance of the frameworks they introduce, which address some of the challenges of FL discussed in Section 3, and for their instantiation of verifiability through ZKP or TEE-based primitives, consistently with the scope defined in *contributions*. Given the novelty of this research area, our literature search (see the Appendix 8) suggests that these 13 papers constitute the most recent and relevant contributions to date. A summary of each of the analyzed solutions is provided in 4.1, 4.2 and 4.3, while an algorithmic description is provided in the Appendix B.

Table 1: Selected papers.

Cryptographic			Hardware-based		
Ref.	**Year**	**Venue**	**Ref.**	**Year**	**Venue**
[98]	2023	IEEE TBDATA	[20]	2025	Prog. in AI
[2]	2024	IEEE ICAIC	[51]	2023	IEEE TII
[103]	2023	arXiv	[110]	2025	arXiv
[11]	2023	USENIX	[47]	2025	arXiv
[66]	2024	ACM	[62]	2024	IEEE MILCOM
[56]	2024	IEEE Access	[61]	2021	ACM/IEEE SEC
[13]	2025	IEEE NOMS			

Excluded papers. Several works in the literature provide verifiability for FL through custom cryptographic primitives. Representative examples include VerifyNet [105], which uses homomorphic hash functions and pseudorandom technologies to verify aggregation correctness; Zhang et al. [111], which employs bilinear aggregate signatures to detect aggregator misbehavior; and Fu et al. [36], which leverages Lagrange interpolation to verify the correctness of aggregated gradients. While these approaches provide a notion of aggregation correctness, they fall outside our scope for two reasons: (i) verification is restricted to protocol participants only: the correctness check relies on secret parameters distributed by a trusted authority during setup, so that any external party, such as a regulator or third-party auditor, who did not receive those parameters cannot verify anything, even ex-post over a recorded transcript. (ii) The proving methodology is structurally limited to a partial coverage of AG2 and cannot extend to AG1, since there is no mechanism to bind the aggregation output to a declared admission set, nor to client-side claims such as CL1 or CL2, as there is no general-purpose encoding of the local training computation. ZKPs and TEEs, by contrast, can in principle be instantiated for any claim in our taxonomy. Future works referring to our classification framework with a focus on different primitives, can still explore a hybrid approach as long as the custom primitive is paired with a mechanism that extends verification to the claims it cannot natively support.

4.1 Verifiable clients behavior only

These works implement client-side claims from classes CL1–CL4.

Li et al. [62]. This paper provides verifiability of the training procedure by leveraging server-side partial retraining, where the server (also acting as the task publisher) uses an SGX-TEE for non-linear operations and a GPU for linear operations. The approach follows Zhang et al. (TrustFL) [112], with the key difference that clients are not required to possess a TEE. After the retraining procedure, the enclave validates the client's behavior by comparing the model update obtained from the server-side retraining of the sampled epochs with the update submitted by the client. If the two match, the server accepts the contribution for aggregation; otherwise, the client is deemed misbehaving and its update is rejected.

Sentinel[110]. This paper by Zhang et al. leverages ARM TrustZone TEEs to provide verifiable training guarantees. Notably, enclaves are not used to execute the training itself; instead, they host a trusted training recorder that captures the control-flow graph CF and a set of critical variables CV during the training procedure. Concretely, the recorder captures (i) a control-flow trace, which reflects the structure of executed training routines (e.g., data loading, forward pass, loss computation, backward pass, optimizer step), and (ii) a set of critical-variable measurements, which records the usage and evolution of security-relevant variables during training (e.g., the received model parameters, training hyperparameters, loss values, gradients and/or parameter updates). This allows the server to detect behavior violation during the training procedure.

ACORN[11]. This work by Bell et al. does not provide an actual proof of training. Instead, it introduces input validation for secure aggregation by attaching succinct, verifiable statements to client updates via range zero-knowledge proofs, so that malformed or out-of-policy contributions can be detected and rejected prior to aggregation. Each masked update sent to the server is accompanied by three proofs: the first proof, $\pi_{i,t}^{\mathrm{Enc}(m_i,\theta_{i,t})}$, guarantees correct calculation of the masked update (i.e. $\hat{\theta}_{i,t} = \mathrm{Encode}(m_i, \theta_{i,t})$) and binds the values m_i and $\theta_{i,t}$ to their commitments d_{m_i} and $d_{\theta_{i,t}}$. The second proof, $\pi_{i,t}^{0\le\theta_{i,t}<\tau}$, enforces a range constraint on $\theta_{i,t}$, while the third proof, $\pi_{i,t}^{\mathrm{valid}(\theta_{i,t})}$, encodes additional application-specific validity constraints, such as L_0, L_2, and L_∞ bounds on the client updates. In particular, enforcing an L_2-norm bound can help mitigate certain classes of model-poisoning attacks by limiting the magnitude of malicious updates [91]. Overall, ACORN primarily targets robustness through verifiable update admissibility constraints, rather than providing auditability or correctness guarantees for the training procedure itself.

4.2 Verifiable aggregator behavior only

These works implement aggregator-side claims from classes AG1-AG3.

zkFL[98]. The first work we present for verifiable aggregation is by Wang et al.: they propose a blockchain-based FL framework in which the aggregator proves the correctness of the aggregations using ZKPs, which are verified on-chain. The system was deployed on the Ethereum public blockchain, with Halo2[21] employed as the zero-knowledge proving scheme. The aggregation strategy used, as reported by authors is FedAVG [69]. 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. So, security against global model tampering is guaranteed; 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.

zkFDL[2]. This work by Ahmadi and Nourmohammadi proposes another FL framework leveraging blockchain, in which the server performs a ZKP 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 system was deployed on the Ethereum blockchain, using the Groth16 [46] 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.

V-FLEX [20]. Proposed by Bouamama et al., targets verifiable aggregation in cross-silo FL by combining TEEs with threshold HE. It enforces aggregation correctness and integrity through hardware-backed attestation and algebraic consistency checks over encrypted updates. The aggregation function is not executed inside the enclave which, instead, is used to (i) attest correct protocol execution, (ii) protect verification logic and key material, and (iii) compute and verify integrity tags, while the computationally intensive homomorphic aggregation is performed on the untrusted host.

Kalapaaking et al. [51]. This paper proposes a blockchain-based cross-edge FL architecture where each blockchain node independently performs SGX-backed model aggregation, and a custom consensus mechanism based on attested global models is used to agree on the final global model, avoiding smart-contract-based aggregation.

4.3 Verifiable client and aggregator behavior

These works implement both client-side claims from classes CL1–CL4 and aggregator-side claims from classes AG1-AG3.

zkPPFL [103]. The work proposed by Xing et al. is, to the best of our knowledge, the first one to approach the problem of making both training and aggregation verifiable. In the process described by the paper, 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 training process. 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.

VPFL [66]. This work by Ma et al. targets third-party verifiability of FL while preserving data confidentiality by combining homomorphic commitments, Merkle trees, and non-interactive zero-knowledge range proofs (Bulletproofs). 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.Note that the approach used for (ii) is similar to ACORN's (4.1), even though the term *input validation* is not used to describe the technique.

Federify[56]. The notable aspect of this work by Keshavarzkalhori et al. is the use of ElGamal threshold encryption [79] to preserve the

privacy of model updates. 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 [46].

VerifBFL [13]. The work proposed by Bellachia et al. [13] is the most recent among the ones leveraging ZKP and the first to leverage a recursive ZKP for both the proof of training (effectively a proof of accuracy) and the proof of aggregation. This work, similarly to VPFL, proves only certain properties of the training process, specifically, whether a given level of accuracy has been achieved. In contrast, the proof of aggregation focuses on verifying the correct computation of the global model at each round using Nova [59].

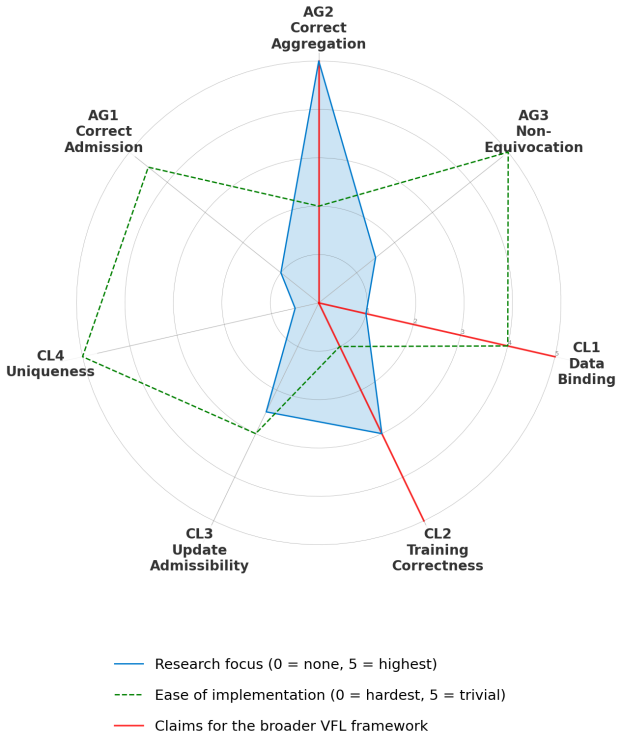

Figure 3: Research attention vs. ease of implementation of each claim

SecureFL [61]. This work by Kuznetsov et al. is a TEE-based FL framework where verifiability is realized as participant-verifiable integrity via remote attestation. In particular, (i) each worker remotely attests the SGX enclave hosting the cloud-side aggregator to verify that the expected SecureFL code is running, and (ii) workers obtain assurance about other workers because the attested cloud enclave performs TrustZone attestation of edge devices on their behalf. Only successfully attested workers are admitted as eligible participants. SecureFL thus provides code-identity evidence for (i) server-side aggregation and (ii) a protected portion of worker-side training, under both TEE vendors threat model.

VerifiableFL[47]. Guo et al. propose a complete verifiable and auditable federated learning framework based on a novel TEE abstraction, referred to as *exclaves*. An exclave is defined as an *integrity-only* and *secret-free* execution environment, whose sole responsibility is to attest the correct execution of a given computation. Secret keys used for attestation are assumed to be hardware-protected and managed by a dedicated secure co-processor, rather than by the exclave software itself. In an offline auditing phase, a third-party auditor constructs a Dataflow Graph where it evaluates the verifiable claims by checking (i) that each task is associated with an allowed code measurement and (ii) that the graph is structurally and dataflow-consistent across rounds, e.g., ensuring that DP tasks are not bypassed, client contributions are not dropped, and dataset commitments remain stable.

Claims coverage. Figure 3 presents a two-dimensional comparison between research attention and ease of implementation. The distribution suggests a concentration of prior work on verifiable aggregation, while alternative verifiability claims receive limited coverage in the literature, despite their practical implementability (e.g. CL1, CL4, AG1, AG3). We inferred the ease-of-implementation metric based on a qualitative assessment of task complexity. The corresponding estimates are reported in Table 2. The figure also highlights in red the claims corresponding to the core verifiability properties of a VFL framework, suggesting a natural priority ordering, even though an end-to-end verifiable deployment arguably requires integrating the remaining claims as well.

Table 2: Deployability of integrity claims.

Claim	Adoption (/13)	Complexity	Ease of implementation (/5)
CL1	2	$O(D)$	4
CL2	6	$O(E \cdot D \cdot N)$	1
CL3	5	$O(N)$	3
CL4	1	$O(1)$	5
AG1	2	$O(K)$	4.5
AG2	10	$O(K \cdot N)$	2
AG3	3	$O(1)$	5

N: model size; D: dataset size; E: number of epochs; K: number of admitted clients.

5 Comparative Analysis

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

5.1 Verifiability assessment

Besides mapping each work to our claim taxonomy, we also assess which verifiability dimensions the papers themselves target. Table 3 distinguishes verifiable data usage (VD), verifiable training (VT), and verifiable aggregation (VA) alongside the underlying construction. Aggregation attracts the most complete solutions because it fits comfortably inside ZK circuits, smart contracts, or TEE enclaves. Training and data verifiability remain largely unexplored or addressed only in limited forms. We also note that reasons for such gap is threefold. First, proving full training is expensive, so works either target toy models (e.g., lightweight CNN in [103], Naive Bayes in [56]) or offload most computation outside the trusted boundary

Table 3: Verifiability properties

	VD	VT	VA	Construction
zkFL [98]	-	-	●	ZKP (Halo2)
zkFDL [2]	-	-	●	ZKP (Groth16)
zkPPFL [103]	-	●	●	ZKP (Groth16, Σ-protocol) + SC
ACORN [11]	-	◐	-	ZKP (Bulletproof)
VPFL [66]	●	◐	◐	ZKP (Bulletproof)
Federify [56]	-	●	●	ZKP (Groth16) + SC
VerifBFL [13]	-	◐	●	ZKP (Nova)
V-FLEX [20]	-	-	●	TEE
Kalapaaking et. al. [51]	-	-	●	TEE + consensus mechanism
Sentinel [110]	-	●	-	TEE
VerifiableFL [47]	●	●	●	Modified TEE abstraction (exclaves)
Li et al. [62]	●	◐	-	TEE + GPU (SeTO protocol)
SecureFL [61]	-	◐	◐	TEE

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

Table 4: Role of Blockchain in Verifiable FL

	Blockchain usage and specific methodology	Main role
zkFL [98]	Off-chain aggregation ZKP verification	Active
zkFDL [2]	Off-chain aggregation ZKP verification	Both
zkPPFL[103]	Training ZKPs verification; SC-based aggregation	Active
VPFL [66]*	Input validation ZKPs verification; off-chain aggregation ZKP verification	Both
Federify [56]	Training ZKPs verification; SC-based aggregation	Active
VerifBFL [13]	Training accuracy ZKPs verification; off-chain aggregation ZKP verification	Active
Kalapaaking et al. [51]	Consensus on the TEEs aggregation outcomes	-

* VPFL does not mention a blockchain explicitly, but refers to a bulletin board.

Table 5: Mitigated attack classes.

Claim	Property	Mitigated attacks
CL2, CL3	Semantic integrity (client)	Model poisoning, free-riding
AG2	Semantic integrity (aggregator)	Global model tampering
CL1	Data provenance	Compliance bypass
CL4	Uniqueness / freshness	Replay attacks
AG1, AG3	Correct admission, non-equivocation	Selective dropping, split-view

(e.g., only final layers in TrustZone in [61], linear operations delegated to an untrusted GPU in [62]). Second, several proposals verify proxies rather than training itself (e.g., [11] and [66] validate inputs, [13] checks accuracy post-hoc, and [110] runs a training recorder inside the TEE rather than the training process itself. Third, the most complete architecture exists only as an abstraction without corresponding commodity hardware [47]. These patterns suggest 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 FL. A further distinction cuts across all TEE-based approaches. Attestation certifies platform identity and code integrity at load time, but confirming that the measured binary actually implements the intended FL logic requires separate code review and systems expertise [7]. TEE verification also depends on vendor-operated infrastructures and evolving certificate chains, complicating long-term audits. Therefore, we argue that evidence grounded in TEEs provides a designated verifier integrity under explicit trust anchors rather than universally verifiable proofs.

Blockchain-based verifiability. Several of the surveyed works leverage blockchain technology to support verifiability in FL. The blockchain's inherent properties (transparency, immutability, and decentralized consensus) make it a natural trust anchor for recording proofs, coordinating participants, and ensuring auditability of the training process. In this context, it is useful to distinguish between two roles that the blockchain can play. In a *passive* role, the blockchain acts as an auditability medium: events, commitments, or model checkpoints are recorded on-chain to enable ex-post inspection. In an *active* role, it acts as a verifiability medium: proofs are verified on-chain by smart contracts, which enforce protocol correctness as part of the execution flow itself. Among the surveyed works, the dominant pattern results the active one (Table 4). It is worth noting that the active role implicitly subsumes a degree of auditability as well: since proof verification events are recorded on-chain, any party can inspect the transcript ex-post and verify that such proofs, along with their public inputs, were accepted at each round. As can be noticed from Table 4, the majority of the works adopt an approach based on off-chain executions with ZKPs to be verified on-chain later on. Alternatively, proof verification could occur on a specialized public chain and only attestation roots are posted to the Ethereum settlement layer [48]. This is a reasonable choice for training, which is a computationally heavy operation and would also reveal sensitive information if executed on-chain due to the transparency property of the blockchain. Similarly, performing aggregation on-chain would incur high fee costs when running on public blockchains. In *zkFL* and *zkFDL*, the server performs aggregation off-chain while generating a ZKP in parallel, which can then be verified on-chain. *zkPPFL* and *Federify* integrates verifiable training by producing a ZKP for the training process that is verified on-chain; moreover, the aggregation itself takes place within the same smart contract used for verification. *VPFL* performs training off-chain and generates ZKPs for input validation to be verified on-chain, while aggregation also occurs off-chain with subsequent on-chain verification. Similarly, *VerifBFL* executes training off-chain and produces ZKPs for model accuracy verification on-chain; the aggregation follows the same pattern, being performed off-chain and later verified on-chain. *Kalapaaking et al.* is the only TEE-based work that relies on the blockchain. However, unlike the previously discussed works, the blockchain is not used as a verifiability medium, but rather to run a custom consensus protocol that allows nodes to agree on the outcomes produced by their TEEs. The works that instead do address auditability most explicitly are *VPFL*, mentioning a generic bulletin board that can be instantiated through a blockchain, and VerifiableFL [47], relying on an authenticated log.

5.2 Security from verifiable claims

A central observation is that verifiability mechanisms can contribute to mitigating a subset of well-known FL attack classes.

While the strength of these mitigations depends on the assumed adversary model as in Appendix A.2, here we identify two complementary protection categories.

Semantic integrity and deviation attacks. Claims CL2 (local training correctness), CL3 (update admissibility), and AG2 (correct aggregation) enforce that recorded actions match the declared specification Spec. When a verifier can check that a client's submission results from applying the specified optimizer to an authorized dataset, or that the aggregator combined exactly the admitted updates according to the stated rule, the protocol rejects arbitrary deviations before they influence the model. With this property, *model poisoning* becomes detectable because malicious gradients cannot satisfy a proof that ties output updates to the declared training procedure. Several works illustrate this constraint through different evidence types [13, 56, 103, 110]. Mitigation could also be achieved partially, like in [61] where only the last layers' training logic is confined to a TEE. Furthermore, *free-riding* is arguably prevented when CL2 requires proof that any meaningful computation occurred, even just a proof-of-accuracy mechanism in [13], the attested per-task records [47], or a partial re-execution strategy [62]. Lastly, *global model tampering* fails when AG2 holds, since the aggregator must demonstrate that ckpt_t follows from applying the aggregation rule to exactly the admitted set [2, 20, 51, 98].

Provenance, consistency, and transcript-level attacks. Claims CL1 (data binding), CL4 (uniqueness), AG1 (correct admission set), and AG3 (non-equivocation) establish provenance and consistency across the execution. Since evidence is bound to the round context ctx_t, so that resubmitting a stale $(h_{i,t}, e^{\mathrm{CL}}_{i,t})$ tuple in a later round fails verification, defenses against *replay attacks* can be established. For example, challenge-response freshness mechanisms and signature-and-commitment binding can both make duplicate submissions detectable [66, 110]. As for *selective dropping*, if each client can verify that its contribution appears in the finalized transcript, the aggregator cannot silently discard updates. We note that AG1 and AG3 jointly serve this aim when passing each client's gradient, or its hash, as a public input to the aggregation proof in [2, 20, 66, 98], unlike [47] where a dataflow graph enables ex-post detection of dropped contributions. Finally, if clients only accept global models whose commitments match a shared log, blockchain-backed coordination can prevent a malicious server from sustaining *split-view (equivocation)* attacks, since divergent model views become unverifiable [82]. This can be achieved by committing aggregation outputs and proofs [13, 56] or by running aggregation inside SGX enclaves at each blockchain node and finalize the global model through consensus [51].

Verifiability alone clearly does not address all FL threats, and the guarantees above hold only under their respective threat models. When training on adversarially curated samples, poisoning remains possible even if these samples were validly committed, namely CL1 and CL2 are both verified. The privacy consequence is analogous to the role of integrity in encryption where confidentiality is much weaker when an attacker can choose or tamper with the objects being protected. Therefore, privacy attacks require mechanisms such as differential privacy or homomorphic encryption that we discuss next in Section 5.3. Lastly, none of the surveyed works fully resolve sybil and coalition attacks that exploit open enrollment by creating many identities so that the attacker increases its weight in Adm_t. In our taxonomy, AG1 and CL4 do not provide sybil resistance without external trust anchors. Practical mitigation therefore still require a form of stake or quota based participation, attested identities, or cross silo governance, layered on top of robust aggregation [60].

5.3 Methodologies for privacy

Research shows that zero-knowledge proofs are well suited to FL because clients can justify correct local behavior while keeping training data private [104]. Consequently, the literature no longer views verifiability as an optional extension but as a necessary complement to privacy-preserving computation [19]. The surveyed works demonstrate various strategies in this respect.

Table 6: Overview of the updates privacy strategies

	Methodology for privacy
zkPPFL [103]	Noise injection
ACORN [11]	Secure aggregation with masking
Federify [56]	Secure aggregation threshold encryption
VerifBFL [13]	Differential Privacy
V-FLEX [20]	Secure aggregation with threshold HE
Kalapaaking et al. [51]	Secure encrypted channel with the TEE
VerifiableFL [47]	Differential Privacy
Li et al. [62]	Secure encrypted channel with the TEE
SecureFL [61]	Secure encrypted channel with the TEE

Claims CL1 and CL2 carry the highest privacy stakes since they involve raw datasets and model updates. To address the challenge of not leaking statistical properties of the underlying samples, schemes like [103] modify public statements across epochs so that intermediate gradients remain hidden, while final outputs undergo perturbation and encryption before reaching the smart contract. Another approach is to allow third-party auditors to verify structural properties of local training without observing gradient values, by committing model updates in Merkle trees and proving range constraints in zero knowledge, as in [66]. Similarly, privacy emerges directly for the claim concerning update admissibility as the exact magnitudes of gradient norms are not disclosed [11].

Aggregator-side claims involve a different privacy context than client-side claims. The aggregator often handles encrypted or masked inputs and any integrity check must operate over these protected representations. Table 6 summarizes how the surveyed works achieve such update confidentiality. For example, homomorphic encryption, with ElGamal threshold schemes or CKKS, allows the aggregator to combine ciphertexts without decrypting them [20, 56]. In TEE-based designs, clients establish secure channels directly with the enclave, where plaintext aggregation occurs in a protected environment [51, 61, 62]. As for differential privacy, while adding noise to gradients prevents statistical inference attacks, it introduces new challenge for verifiability, because if clients freely choose their noise, malicious actors can inject arbitrary values. This maps to CL2 for local noise generation and CL3 for proving noise stays within bounds, while central DP at the aggregator falls under AG2. In the recent literature, solutions range from committing to random seeds before execution [75] to fully arithmetizing noise generation into

zkSNARK circuits with Pedersen-committed outputs [99]. Within the surveyed FL works, [2] and [13] use verifiable random functions so auditors can confirm noise was drawn correctly, while [103] modifies verification keys across rounds to prove noise addition without revealing exact perturbations.

5.4 Verifiability overhead across different settings

We examine here the computational and communication overhead derived from local training, aggregation, and auxiliary operations performed by our system model entities in Section 3.1. A strict comparison isn't feasible as experimental setups vary widely and key metrics are often missing. We therefore collect and systematize metrics for each verifiable claim in Table 7, and provide insights into the practical implications associated with each approach.

The cost of client-side claims. Most overhead comes from CL2 and CL3, which require encoding expensive computations into verifiable statements. For CL2 (local training correctness), translating a neural network training loop into an arithmetic circuit explodes the constraint count, even for small models and single training epochs [103]. Similarly, CL3 (update admissibility) incurs a high cost when it must compute norm bounds or clipping constraints over high-dimensional update vectors, for instance when verifying noise addition [47]. In contrast, CL1 (data binding) reduces to a one-time commitment over the local dataset, whose cost is reported in [66] but not in [62], and CL4 (uniqueness) seems to piggybacks on existing infrastructure with minimal added cost. We argue that in TEE-based systems CL4 could reduce to binding the attestation report to a fresh challenge or nonce issued by the server at the start of each round, but such neglegible cost could be inferred only from [47]. Alternatively, uniqueness is achieved through lightweight Σ-protocol proofs that demonstrate the inputs to one computation piece match the outputs of the preceding piece, reusing the common reference string already generated for the proofs in CL2 as in [103]. As for verifier cost, it is underreported in several works, since some papers do not isolate it as a separate metric and instead embed it in the aggregation control flow. When reported, it ranges from millisecond-scale pairing checks for zkSNARKs to on-chain gas costs when verification is delegated to a smart contract. An exception is [62], where verification consists of re-executing randomly sampled training rounds inside the server's enclave.

The cost of aggregator-side claims. Correct aggregation and checkpoint update (AG2) absorbs most of overhead as it encodes the aggregation rule itself, which touches every admitted update. When the aggregator must prove correctness of a weighted average over hundreds of parameters from dozens of clients, the resulting circuit or enclave workload scales with both the model size and the admitted set [2, 51, 98]. Whenever aggregation is executed entirely on-chain via a secure sum protocol the prover cost becomes irrelevant [56, 103]. Gas consumption, not latency, becomes the dominant metric once verification is delegated to a smart contract as in [56] and [2]. In these cases, the storage overhead reduces to a ledger entry of $O(1)$ or $O(m)$, depending on whether only a hash or the full aggregated model is posted.

As for AG1, it is often invisible arguably because its cost folds naturally into AG2. The claim merely asserts that the admitted set S_t is non-arbitrary and that someone actually executed the admission rule $\mathcal{F}_V$.VerifyClient on the round's evidence. Whether that cost appears as a separate line depends on who runs the rule and how. For example, in [51] the TEE verifies client attestations before aggregating, so the 60-180 ms latency captures AG1 on the verifier side. Instead, [47] report a 1.5 s prover cost for generating exclave dataflow records that bind S_t to auditable evidence. Finally, AG3 costs are difficult to standardize because non-equivocation depends on how the shared transcript is instantiated. Some protocols report small latencies when finalization consists of signing a checkpoint inside an enclave, as in [51] where global model deployment takes a few hundred milliseconds depending on the number of blockchain nodes. The heterogeneity of reported figures reflects this variety. Other works report gas units when the checkpoint commitment is written on-chain [98], or prover time when outputs are committed to secure storage with runtime attestation [47]. As for communication overhead, it is driven less by proof size than by the encrypted or committed update itself, which scales as explicitly reported in [98]. By contrast, TEE-based approaches incur lower communication costs, since attestation reduces to a signed enclave quote verified by a trusted attestation service rather than a full cryptographic encoding of the update [51]. Given the above, the cost of verifiability depends on the targeted claim, the level of evidence required and on which infrastructure network verification is executed.

6 Remarks and Research Outlook

Our systematization shows that the toolbox for verifiable FL is maturing although it is still unevenly distributed across all the possible integrity claims that we identified (cf. Table 3). We close the paper with a set of forward-looking observations.

Scaling verifiable training to realistic models. The most visible bottleneck of the current literature is the gap between the model sizes that can be trained in the open and those that can be proved. State-of-the-art proof of training systems (e.g., [1, 90, 96]) demonstrate subsecond verification at the order of 10^7 parameters, 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 (some authors report 6–22 min per step on mid-size CNNs [1]). (ii) Non-arithmetic activations (e.g. ReLU) lie outside the algebraic frontends used by SNARKs (a gadget dedicated to ReLU was designed in [90], yet other activations remain unsupported). (iii) Modern DNNs mix layers with wildly different tensor shapes introducing padding overheads. Within FL, this gap is compounded because every admitted client must produce CL2 evidence under the same constraint. Aside from TEE solutions, we propose the following promising trajectories. Firstly, explore more verifiable FL work leveraging GPU-friendly recursion and lookup-based protocols [81]. Second, incrementally verifiable computation and proof-carrying data [29, 95], instantiated by modern folding schemes such as Nova [59], could let each gradient step extend a running CL2 certificate, with $O(1)$ verification in the number of training steps. A third trajectory comes from methods such

Table 7: Verifiability Overhead in Federated Learning

Claim	Setting	Scale	Setup	Verifier	Prover	Upload	Broadcast	Evidence	System Spec
CL1	[66] CNN1, MNIST	100 × 1	130 ms	39 ms	26 ms	n.d.	n.d.	n.d.	Intel Core i7-13700H, 16GB
	[47] GPT2, Wikitext103	4 × 10	257 s	n.d.	0.62 s	n.d.	n.d.	n.d.	AMD EPYC 7763v; NVIDIA H100
CL2	[103] LR, Price Pred.	n.d. × 500	514 s	1.5 s	50 s	5 KB	n.d.	5 KB	Intel Xeon Gold 5128 CPU, 4GB
	[103] NN, Iris Classif.	n.d. × 90k★	950 s	26 s	90 s	5 KB	n.d.	5 KB	Intel Xeon Gold 5128 CPU, 4GB
	[56] Gaussian NB, n.d.	n.d. × n.d.	n.d.	>0.9 M gas	n.d.	22 KB	n.d.	22 KB	n.d.
	[62] VGG19, CIFAR-10	n.d.	n.d.	21.4 s	–	n.d.	n.d.	n.d.	Intel Core i7-9700K; NVIDIA RTX 2080
	[110] MobileNetV3, CIFAR-10	100 × 30	n.d.	5.4 ms	1.5 s	59 KB	–	59 KB	Intel Core i7; NVIDIA RTX 3080
	[61] LeNet, VGG-7	10 × 1	n.d.	n.d.	5.02 s	n.d.	n.d.	n.d.	Intel Core i7-7700, 16GB
	[47] GPT2, Wikitext103	4 × 10	257 s	n.d.	866 s	n.d.	n.d.	n.d.	AMD EPYC 7763v, NVIDIA H100
CL3	[11] CNN, MNIST	500 × 160	–	18 ms	3.9 s	1.8 KB	n.d.	1.8 KB	Intel Core i7-1185G7, 16GB
	[11] LeNet-5, CIFAR-10	500 × 100	–	72 ms	17.4 s	1.8 KB	n.d.	1.8 KB	Intel Core i7-1185G7, 16GB
	[11] ResNet-20, CIFAR-10	500 × 160	–	286 ms	75.6 s	2.0 KB	n.d.	2.0 KB	Intel Core i7-1185G7, 16GB
	[11] LSTM, Shakespeare	500 × 20	–	n.d.	226 s	2.2 KB	n.d.	2.2 KB	Intel Core i7-1185G7, 16GB
	[13] CNN, MNIST	n.d. × n.d.	785.53 s	0.642 s	81.95 s	33.3 Kb	–	33.3 Kb	Intel Core i7-10750H, 16 GB.
	[66] CNN1, MNIST	100 × 1	130 ms	41 ms	92 ms	n.d.	n.d.	n.d.	Intel i7-13700H, 16GB
	[20] CNN, MNIST	10 × 1	n.d.	5 ms	10 ms	63 KB	n.d.	63 KB	Intel Core i5, 8GB
	[47] GPT2, Wikitext103	4 × 10	–	n.d.	117 s	n.d.	n.d.	n.d.	AMD EPYC 7763v, NVIDIA H100
CL4	[47] GPT2, Wikitext103	4 × 10	–	n.d.	0.69 s	n.d.	n.d.	n.d.	AMD EPYC 7763v, NVIDIA H100
AG1	[51] Global model verification	5-20 × 1	n.d.	60–180 ms	n.d.	–	n.d.	n.d.	Intel Xeon E-2288 G, 16GB
	[47] GPT2, Wikitext103	4 × 10	–	n.d.	1.5 s	n.d.	n.d.	n.d.	AMD EPYC 7763v, NVIDIA H100
AG2	[98] ResNet18, CIFAR-10	32 × 100	n.d.	7.5 min	15 min	44 MB	45 MB	299 KB	NVIDIA Tesla T4, 16GB
	[98] ResNet50, CIFAR-10	32 × 100	n.d.	27 min	55 min	491 MB	492 MB	628 KB	NVIDIA Tesla T4, 16GB
	[98] LSTM-3L, PTB	32 × 100	n.d.	20 min	40 min	n.d.	n.d.	619 KB	NVIDIA Tesla T4, 16GB
	[2] MLP, Activity Classif.	10/20 × 1	n.d.	1.4 / 1.8 M gas	n.d.	n.d.	n.d.	n.d.	n.d.
	[56] Gaussian NB, n.d.	n.d. × n.d.	n.d.	>0.7 M gas	–	7 KB	n.d.	7 KB	n.d.
	[103] NN, Iris Classif.	n.d. × 90k★	n.d.	n.d.	–	n.d.	n.d.	n.d.	Intel Xeon Gold 5128 CPU, 4GB
	[13] CNN, MNIST	5 × 1	105.64 s	0.536 s	2.189 s	–	35.4 Kb	35.4 Kb	Intel Core i7-10750H, 16 GB
	[66] CNN1, MNIST	100 × 1	130 ms	0.7 s	1.3 s	n.d.	n.d.	n.d.	Intel i7-13700H, 16GB
	[20] CNN, MNIST	10 × 1	n.d.	10 ms	80 ms	–	168 KB	168 KB	Intel Core i5, 8GB
	[51] LeNet, F-MNIST	2-40 × 1	n.d.	n.d.	1.2 s	n.d.	n.d.	n.d.	Intel Xeon E-2288 G, 16GB
	[51] VGG16, CIFAR-10	40 × 1	n.d.	n.d.	2.7 s	n.d.	n.d.	n.d.	Intel Xeon E-2288 G, 16GB
	[61] LeNet, VGG-7	10-100 × 1	n.d.	n.d.	1.73-16.4 s	n.d.	n.d.	n.d.	Intel Core i7-7700, 16GB
	[47] GPT2, Wikitext103	4 × 10	–	n.d.	5.0 s	n.d.	n.d.	n.d.	AMD EPYC 7763v, NVIDIA H100
AG3	[98] LSTM-3L, PTB	32 × 100	–	O(1)	0.02 M gas	–	32 B	32 B	NVIDIA Tesla T4, 16GB
	[51] Global model deployment	5-20 × 1	n.d.	n.d.	100-230 ms	–	n.d.	n.d.	Intel Xeon E-2288 G, 16GB
	[47] GPT2, Wikitext103	4 × 10	–	n.d.	2.9 s	n.d.	n.d.	n.d.	AMD EPYC 7763v, NVIDIA H100

Each row reports the marginal overhead required to make the specified claim verifiable, according to our claim taxonomy (CL1-CL4, AG1-AG3) in Section 3.2. All reported numeric values are considered approximate. Scale is the number of participating clients × number of FL rounds. Setup denotes one-time costs amortized over the protocol execution. Verifier and Prover report per-round computation overhead. For TEE-based solutions, Prover cost includes TEE execution overhead and Verifier cost includes proof checking or attestation verification. For CL claims, times are per-client, while for AG claims, the aggregator acts as prover. The communication overhead is per-client per-round beyond plaintext FL model updates. For AG-claims, this may include client-side payload needed by the verifier. Broadcast is the per-round communication overhead sent by the server beyond broadcasting the global model (i.e. payload to clients and payload propagated to external verifiers). Whenever the values in Upload, Broadcast and Evidence are identical, it indicates that the metric reported by the source combines both the payload and the proof artifact. We use '–' for not applicable and 'n.d.' when the paper reports no quantitative data.
★The n.d. verifier entry follows from treating each sample-iteration pair as one proof piece (75 training samples × 1200 iterations yields 90,000 proofs), making per-round isolation impractical.

as LoRA that freeze the original model weights and only train a small pair of low-rank matrices inserted at each layer, which consequently make up a smaller fraction of what a dense update would require as witness. Recent work has shown that this is enough to push verifiable training to LLM scale which, interestingly for our FL claims, means that the same proving circuit that achieves CL2 also concurrently enforces the corresponding CL1 (Merkle commitment over digests carrying license and preprocessing tags), CL3 (linear constraints on sampling quotas and verified fixed-point gradient clipping) and CL4 (a hash chain that ingests, at every step, the manifest digest and the round identifier) claims, at almost no additional asymptotic cost [4]. Other adjacent lines of research, like collaborative snarks and decentralized model-parallel training are presented in Appendix C.

Overall, we argue that the community would benefit from a shared benchmark where verifiable training is evaluated within a FL setting, which would also help clarify whether the more costly claims are achievable at cross-device scale or only in cross-silo settings.

Practical claims with minimal overhead. A second observation emerged from our survey is that not all claims are necessarily expensive. If we take the case of data binding (CL1), in practice it reduces to a one-time commitment over the local dataset and per-round Merkle openings. Uniqueness (CL4) is arguably even cheaper, since it only requires binding the client evidence to the current round context (e.g. reusing the CRS already produced for CL2 or injecting a fresh challenge into a TEE attestation report). On the aggregator side, correct admission (AG1) may even be considered invisible if the same circuit that proves AG2 already re-runs the admission rule, while AG3 can leverage the transcript at a constant verification cost. Yet, as Table 3 shows, none of the surveyed systems satisfies all of these guarantees simultaneously, even though the marginal cost of adding the missing ones is, in many cases, dominated by other components of the protocol. Concretely, deployments can already today exploit settling verification on low-fee rollups, anchoring aggregates via data availability layers, and batching recursive proofs to amortize gas in blockchain-based settings [21, 48]. On this regard, emerging modular verification networks such as zkVerify [48] suggest that the gas figures reported in Table 7 have substantial headroom for cost reduction.

Toward a composable framework for verifiable FL. The level of abstraction of our classification framework is sufficient to compare existing systems and investigate gaps, but it cannot by itself state when several claims compose into a single security guarantee. An interesting next step would be to ask whether the transcript model in Figure 2 can be given a formal semantics, either through property-specific games or in the spirit of Universal Composability [24]. In such a formulation, an ideal functionality would provide the common object over which games are played. One could define separate games for data binding, local training correctness, admission soundness, aggregation correctness and transcript consistency and then prove restricted composition statements (i.e. no arbitrary composition). Alternatively, one could define small ideal subfunctionalities for these claims and study when replacing a real evidence mechanism by its ideal version preserves the relevant property, rather than all possible security properties in all possible environments. Existing work on practical UC zero-knowledge provides specification languages and compilers that turn high-level relation statements into UC-secure ZK protocols in a modular way [23], and could plausibly serve as a building block to formalise the relations described in paragraph 3.2. For TEE-backed claims, the model would also need to state which global attestation assumptions are being used, since formal abstractions treat the TEE as a globally shared setup to reflect that manufacturer attestation keys are shared across protocol instances rather than sampled at each session [78], and vary across abstractions in what an enclave is allowed to do inside its trusted runtime, hence which attacks the adversary can perform [68]. Developing such a model remains an open problem and a natural follow-up to the systematization proposed here.

7 Related work

While several works have surveyed verifiability in ML broadly [55, 72, 81, 89, 104], there is still a lack of systematization of verifiability as a first class dimension specific to FL settings. For example, Xing et al. [104] attempt to systematize verifiability in distributed ML settings (i.e., single- vs. multi-worker, and training vs. inference), but they focus primarily on circuit optimizations and proof efficiency, with FL simply as an application scenario. The closest work in scope is another recent survey that focuses on cross-silo settings only [58]. On the secure-aggregation side, a recent SoK [67] makes observations related to ours, though without considering instantiations based on zero-knowledge proofs or TEEs, namely that hiding individual updates is not by itself enough to protect the underlying training data and that a dedicated mechanism is needed to let parties check that the published aggregate matches the one prescribed by the protocol. More broadly, the FL literature offers high availability of surveys. These are distributed, though not limited, across several distinct perspectives, including privacy-preserving protocols [19, 27, 30, 64, 107], security threats and defenses [10, 25, 60, 88], fairness, bias and data heterogeneity [17, 85, 93], network topology and decentralization models [14, 82, 108], scalability and convergence efficiency [6, 8, 101]. Notably, Bontekoe et al. recently provided a comprehensive survey of verifiable privacy-preserving computation over distributed data, cataloging a wide range of cryptographic mechanisms [19]. However, their analysis prioritizes low-level constructions and remains agnostic to the nuances of FL protocols. In terms of FL evaluation, existing frameworks organize goals around utility, efficiency, security and privacy, capturing model quality, system costs, and resilience to attacks or leakage respectively [26]. However, these categories measure observed outcomes rather than process integrity. High accuracy or low leakage does not prove the protocol ran as specified, nor can such metrics detect deviations in real time. The authors themselves flag this limitation for deployments beyond the semi-honest model. We therefore treat verifiability as a distinct evaluation dimension that is constrained by, and interacts with, privacy preservation.

8 Conclusion

This SoK provides a unified classification framework for reasoning about verifiability in FL. By introducing a claim taxonomy that separates client-side from aggregator-side guarantees, we enable systematic comparison of 13 representative systems spanning ZKP- and TEE-based approaches. Our analysis reveals a marked imbalance between research effort and implementability. Aggregation correctness attracts the most attention, whereas several integrity claims remain underexplored despite their arguably lower engineering effort. We observe that progress hinges on reducing computation, communication and blockchain-related expenses, but also on giving the field a more stable common ground. Looking forward, verifiable FL at realistic scales will likely require advances along multiple fronts. In Section 6 we highlight some promising trajectories to bridge outstanding gaps. We hope this work helps the privacy community evaluate and design mechanisms that strengthen, rather than undermine, the data protection goals of federated learning.

Acknowledgments

In this work LLMs (specifically, ChatGPT) were used exclusively for grammar checks and the stylistic polishing of certain sentences. All outputs were reviewed by the authors to ensure that the original meaning was preserved and that no additional content was

introduced by the LLM. In this paper, LLMs were NOT used for brainstorming or for generating original, non-human content.

References

[1] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. 2024. Zero-Knowledge Proofs of Training for Deep Neural Networks. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 4316–4330. https://doi.org/10.1145/3658644.3670316

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

[3] Thalaiyasingam Ajanthan, Sameera Ramasinghe, Yan Zuo, Gil Avraham, and Alexander Long. 2025. Nesterov Method for Asynchronous Pipeline Parallel Optimization. arXiv:2505.01099 [cs.LG] https://arxiv.org/abs/2505.01099

[4] Hasan Akgul, Daniel Borg, Arta Berisha, Amina Rahimova, Andrej Novak, and Mila Petrov. 2025. Verifiable Fine-Tuning for LLMs: Zero-Knowledge Training Proofs Bound to Data Provenance and Policy. *arXiv preprint arXiv:2510.16830* (2025).

[5] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. 2016. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In *Advances in Cryptology – ASIACRYPT 2016*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 191–219.

[6] Omair Rashed Abdulwareth Almanifi, Chee-Onn Chow, Mau-Luen Tham, Joon Huang Chuah, and Jeevan Kanesan. 2023. Communication and computation efficiency in Federated Learning: A survey. *Internet of Things* 22 (2023), 100742. https://doi.org/10.1016/j.iot.2023.100742

[7] ANSSI. 2025. *Technical Position Paper on Confidential Computing*. Technical Report. Agence nationale de la sécurité des systèmes d'information (ANSSI). https://messervices.cyber.gouv.fr/documents-guides/anssi-technical-position-paper-coco-v1.0.pdf

[8] Meriem Arbaoui, Mohamed-el-Amine Brahmia, Abdellatif Rahmoun, and Mourad Zghal. 2024. Federated Learning Survey: A Multi-Level Taxonomy of Aggregation Techniques, Experimental Insights, and Future Frontiers. *ACM Trans. Intell. Syst. Technol.* 15, 6, Article 113 (Nov. 2024), 69 pages. https://doi.org/10.1145/3678182

[9] Ariful Azad and Afeefa Banu. 2024. Publication Trends in Artificial Intelligence Conferences: The Rise of Super Prolific Authors. arXiv:2412.07793 [cs.DL] https://arxiv.org/abs/2412.07793

[10] Li Bai, Haibo Hu, Qingqing Ye, Haoyang Li, Leixia Wang, and Jianliang Xu. 2024. Membership Inference Attacks and Defenses in Federated Learning: A Survey. *ACM Comput. Surv.* 57, 4, Article 89 (Dec. 2024), 35 pages. https://doi.org/10.1145/3704633

[11] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. 2023. ACORN: Input Validation for Secure Aggregation. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 4805–4822. https://www.usenix.org/conference/usenixsecurity23/presentation/bell

[12] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1253–1269. https://doi.org/10.1145/3372297.3417885

[13] Ahmed Ayoub Bellachia, Mouhamed Amine Bouchiha, Yacine Ghamri-Doudane, and Mourad Rabah. 2025. VerifBFL: Leveraging zk-SNARKs for A Verifiable Blockchained Federated Learning. arXiv:2501.04319 [cs.CR] https://arxiv.org/abs/2501.04319

[14] Paolo Bellavista, Luca Foschini, and Alessio Mora. 2021. Decentralised Learning in Federated Deployment Environments: A System-Level Survey. *ACM Comput. Surv.* 54, 1, Article 15 (Feb. 2021), 38 pages. https://doi.org/10.1145/3429252

[15] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046. https://eprint.iacr.org/2018/046

[16] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. 2013. SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In *Advances in Cryptology – CRYPTO 2013*, Ran Canetti and Juan A. Garay (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 90–108.

[17] Nawel Benarba and Sara Bouchenak. 2025. Bias in Federated Learning: A Comprehensive Survey. *ACM Comput. Surv.* 57, 11, Article 291 (June 2025), 36 pages. https://doi.org/10.1145/3735125

[18] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1175–1191. https://doi.org/10.1145/3133956.3133982

[19] Tariq Bontekoe, Dimka Karastoyanova, and Fatih Turkmen. 2025. Verifiability for Privacy-Preserving Computing on Distributed Data — a Survey. *International Journal of Information Security* 24, 3 (May 2025), 141. https://doi.org/10.1007/s10207-025-01047-7

[20] Jaouhara Bouamama, Yahya Benkaouz, and Mohammed Ouzzif. 2025. V-FLEX: Verifiable cross-silo federated learning using trusted execution environment. *Progress in Artificial Intelligence* (07 2025). https://doi.org/10.1007/s13748-025-00386-9

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

[22] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy (SP)*. 315–334. https://doi.org/10.1109/SP.2018.00020

[23] Jan Camenisch, Stephan Krenn, and Victor Shoup. 2011. A framework for practical universally composable zero-knowledge protocols. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 449–467.

[24] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 136–145.

[25] Vincenzo Carletti, Pasquale Foggia, Carlo Mazzocca, Giuseppe Parrella, and Mario Vento. 2025. *SoK: gradient inversion attacks in federated learning*. USENIX Association, USA.

[26] Di Chai, Leye Wang, Liu Yang, Junxue Zhang, Kai Chen, and Qiang Yang. 2024. A Survey for Federated Learning Evaluations: Goals and Measures. *IEEE Transactions on Knowledge and Data Engineering* 36, 10 (2024), 5007–5024. https://doi.org/10.1109/TKDE.2024.3382002

[27] Jingxue Chen, Hang Yan, Zhiyuan Liu, Min Zhang, Hu Xiong, and Shui Yu. 2024. When Federated Learning Meets Privacy-Preserving Computation. *ACM Comput. Surv.* 56, 12, Article 319 (Oct. 2024), 36 pages. https://doi.org/10.1145/3679013

[28] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas Ward. 2019. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. Cryptology ePrint Archive, Paper 2019/1047. https://eprint.iacr.org/2019/1047

[29] Alessandro Chiesa and Eran Tromer. 2010. Proof-Carrying Data and Hearsay Arguments from Signature Cards.. In *ICS*, Vol. 10. 310–331.

[30] Christos Chronis, Iraklis Varlamis, Yassine Himeur, Aya N. Sayed, Tamim M. AL-Hasan, Armstrong Nhlabatsi, Faycal Bensaali, and George Dimitrakopoulos. 2024. A Survey on the use of Federated Learning in Privacy-Preserving Recommender Systems. *IEEE Open Journal of the Computer Society* 5 (2024), 227–247. https://doi.org/10.1109/OJCS.2024.3396344

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

[32] Romain de Laage, Peterson Yuhala, François-Xavier Wicht, Pascal Felber, Christian Cachin, and Valerio Schiavoni. 2025. 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, New York, NY, USA, 152–163. https://doi.org/10.1145/3701717.3730543

[33] Jiacheng Du, Jiahui Hu, Zhibo Wang, Peng Sun, Neil Zhenqiang Gong, Kui Ren, and Chun Chen. 2025. SoK: on gradient leakage in federated learning. In *Proceedings of the 34th USENIX Conference on Security Symposium* (Seattle, WA, USA) *(SEC '25)*. USENIX Association, USA, Article 157, 20 pages.

[34] Haokun Fang and Quan Qian. 2021. Privacy Preserving Machine Learning with Homomorphic Encryption and Federated Learning. *Future Internet* 13, 4 (2021). https://doi.org/10.3390/fi13040094

[35] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. 2020. Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1605–1622. https://www.usenix.org/conference/usenixsecurity20/presentation/fang

[36] Anmin Fu, Xianglong Zhang, Naixue Xiong, Yansong Gao, Huaqun Wang, and Jing Zhang. 2022. VFL: A Verifiable Federated Learning With Privacy-Preserving for Big Data in Industrial IoT. *IEEE Transactions on Industrial Informatics* 18, 5 (2022), 3316–3326. https://doi.org/10.1109/TII.2020.3036166

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

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

[39] Sanjam Garg, Aarushi Goel, Abhishek Jain, Bhaskar Roberts, and Sruthi Sekar. 2025. Malicious Security in Collaborative zk-SNARKs: More than Meets the Eye. Cryptology ePrint Archive, Paper 2025/1026. https://eprint.iacr.org/2025/1026

[40] Sanjam Garg, Aarushi Goel, Dimitris Kolonelos, and Rohit Sinha. 2025. Jigsaw: Doubly Private Smart Contracts. Cryptology ePrint Archive, Paper 2025/1147. https://eprint.iacr.org/2025/1147

[41] Rosario Gennaro, Craig Gentry, and Bryan Parno. 2010. Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In *Advances in Cryptology – CRYPTO 2010*, Tal Rabin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 465–482.

[42] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic Span Programs and Succinct NIZKs without PCPs. In *Advances in Cryptology – EUROCRYPT 2013 (Lecture Notes in Computer Science, Vol. 7881)*. Springer, 626–645. https://doi.org/10.1007/978-3-642-38348-9_37

[43] Shafi Goldwasser, Michael P. Kim, Vinod Vaikuntanathan, and Or Zamir. 2022. Planting Undetectable Backdoors in Machine Learning Models : [Extended Abstract]. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*. 931–942. https://doi.org/10.1109/FOCS54457.2022.00092

[44] S Goldwasser, S Micali, and C Rackoff. 1985. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing* (Providence, Rhode Island, USA) *(STOC '85)*. Association for Computing Machinery, New York, NY, USA, 291–304. https://doi.org/10.1145/22145.22178

[45] Shafi Goldwasser, Guy N. Rothblum, Jonathan Shafer, and Amir Yehudayoff. 2021. Interactive Proofs for Verifying Machine Learning. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 185)*, James R. Lee (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 41:1–41:19. https://doi.org/10.4230/LIPIcs.ITCS.2021.41

[46] Jens Groth. 2016. On the Size of Pairing-based Non-interactive Arguments. Cryptology ePrint Archive, Paper 2016/260. https://eprint.iacr.org/2016/260

[47] Jinnan Guo, Kapil Vaswani, Andrew Paverd, and Peter Pietzuch. 2025. VerifiableFL: Verifiable Claims for Federated Learning using Exclaves. arXiv:2412.10537 [cs.CR] https://arxiv.org/abs/2412.10537

[48] Horizen Labs Research. 2024. *zkVerify Protocol: A Modular & Decentralized Proof Verification Network*. Technical Report. Horizen Labs. https://downloads.horizenlabs.io/file/labs-web-assets/zkverify-protocol-whitepaper.pdf Draft v0.1.0.

[49] Zhanglong Ji, Zachary C. Lipton, and Charles Elkan. 2014. Differential Privacy and Machine Learning: a Survey and Review. arXiv:1412.7584 [cs.LG] https://arxiv.org/abs/1412.7584

[50] Weizhao Jin, Yuhang Yao, Shanshan Han, Jiajun Gu, Carlee Joe-Wong, Srivatsan Ravi, Salman Avestimehr, and Chaoyang He. 2024. FedML-HE: An Efficient Homomorphic-Encryption-Based Privacy-Preserving Federated Learning System. arXiv:2303.10837 [cs.LG] https://arxiv.org/abs/2303.10837

[51] Aditya Pribadi Kalapaaking, Ibrahim Khalil, Mohammad Saidur Rahman, Mohammed Atiquzzaman, Xun Yi, and Mahathir Almashor. 2023. Blockchain-Based Federated Learning With Secure Aggregation in Trusted Execution Environment for Internet-of-Things. *IEEE Transactions on Industrial Informatics* 19, 2 (2023), 1703–1714. https://doi.org/10.1109/TII.2022.3170348

[52] Aditya Pribadi Kalapaaking, Veronika Stephanie, Ibrahim Khalil, Mohammed Atiquzzaman, Xun Yi, and Mahathir Almashor. 2022. SMPC-Based Federated Learning for 6G-Enabled Internet of Medical Things. *IEEE Network* 36, 4 (2022), 182–189. https://doi.org/10.1109/MNET.007.2100717

[53] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. 2020. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 5132–5143. https://proceedings.mlr.press/v119/karimireddy20a.html

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

[55] Vid Keršič, Sašo Karakatič, and Muhamed Turkanović. 2024. On-chain Zero-Knowledge Machine Learning: An Overview and Comparison. *Journal of King Saud University – Computer and Information Sciences* 36, 9 (2024), 102207. https://doi.org/10.1016/j.jksuci.2024.102207

[56] Ghazaleh Keshavarzkalhori, Cristina Pérez-Solà, Guillermo Navarro-Arribas, Jordi Herrera-Joancomartí, and Habib Yajam. 2023. Federify: A Verifiable Federated Learning Scheme Based on zkSNARKs and Blockchain. *IEEE Access* PP (01 2023), 1–1. https://doi.org/10.1109/ACCESS.2023.3347039

[57] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2017. Federated Learning: Strategies for Improving Communication Efficiency. arXiv:1610.05492 [cs.LG] https://arxiv.org/abs/1610.05492

[58] Aleksei Korneev and Jan Ramon. 2025. A Survey on Verifiable Cross-Silo Federated Learning. *Transactions on Machine Learning Research* (June 2025). https://doi.org/10.1000/TMLR.2025.XXXX arXiv:2410.09124

[59] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. 2022. 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, Berlin, Heidelberg, 359–388. https://doi.org/10.1007/978-3-031-15985-5_13

[60] Naveen Kumar Kummari, Krishna Mohan Chalavadi, and Linga Reddy Cenkeramaddi. 2024. The Impact of Adversarial Attacks on Federated Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46, 5 (May 2024), 2672–2691. https://doi.org/10.1109/TPAMI.2023.3322785

[61] Eugene Kuznetsov, Yitao Chen, and Ming Zhao. 2021. SecureFL: Privacy Preserving Federated Learning with SGX and TrustZone. In *2021 IEEE/ACM Symposium on Edge Computing (SEC)*. 55–67. https://doi.org/10.1145/3453142.3491287

[62] Jiarui Li, Nan Chen, Shucheng Yu, and Thitima Srivatanakul. 2024. Efficient and Privacy-Preserving Integrity Verification for Federated Learning with TEEs. In *MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM)*. 999–1004. https://doi.org/10.1109/MILCOM61039.2024.10773815

[63] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. arXiv:1812.06127 [cs.LG] https://arxiv.org/abs/1812.06127

[64] Fengxia Liu, Zhiming Zheng, Yexuan Shi, Yongxin Tong, and Yi Zhang. 2023. A survey on federated learning: a perspective from multi-party computation. *Frontiers of Computer Science* 18 (12 2023). https://doi.org/10.1007/s11704-023-3282-7

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

[66] Juan Ma, Hao Liu, Mingyue Zhang, and Zhiming Liu. 2024. VPFL: Enabling verifiability and privacy in federated learning with zero-knowledge proofs. *Knowledge-Based Systems* 299 (06 2024), 112115. https://doi.org/10.1016/j.knosys.2024.112115

[67] Mohamad Mansouri, Melek Önen, Wafa Ben Jaballah, and Mauro Conti. 2023. Sok: Secure aggregation based on cryptographic schemes for federated learning. *Proceedings on Privacy Enhancing Technologies* (2023).

[68] Lorenzo Martinico and Markulf Kohlweiss. 2025. AGATE: Augmented Global Attested Trusted Execution in the Universal Composability framework. In *2025 IEEE 38th Computer Security Foundations Symposium (CSF)*. IEEE, 49–64.

[69] H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2016. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics*. https://api.semanticscholar.org/CorpusID:14955348

[70] J. Menandas and Mary Subaja Christo. 2025. Analysis of various homomorphic encryption algorithms based on primitive functions and applications. *OPSEARCH* (06 2025). https://doi.org/10.1007/s12597-025-00965-3

[71] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services* (Virtual Event, Wisconsin) *(MobiSys '21)*. Association for Computing Machinery, New York, NY, USA, 94–108. https://doi.org/10.1145/3458864.3466628

[72] Fan Mo, Zahra Tarkhani, and Hamed Haddadi. 2024. Machine Learning with Confidential Computing: A Systematization of Knowledge. *ACM Comput. Surv.* 56, 11, Article 281 (June 2024), 40 pages. https://doi.org/10.1145/3670007

[73] Antonio Muñoz, Ruben Ríos, Rodrigo Román, and Javier López. 2023. A survey on the (in)security of trusted execution environments. *Computers & Security* 129 (2023), 103180. https://doi.org/10.1016/j.cose.2023.103180

[74] Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. (May 2009). http://www.bitcoin.org/bitcoin.pdf

[75] Arjun Narayan, Ariel Feldman, Antonis Papadimitriou, and Andreas Haeberlen. 2015. Verifiable Differential Privacy. In *Proceedings of the 10th European Conference on Computer Systems (EuroSys '15) (EuroSys)*. ACM, Bordeaux, France, 28:1–28:14. https://doi.org/10.1145/2741948.2741978

[76] Alex Ozdemir and Dan Boneh. 2022. Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 4291–4308. https://www.usenix.org/conference/usenixsecurity22/presentation/ozdemir

[77] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2013. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy*. 238–252. https://doi.org/10.1109/SP.2013.47

[78] Rafael Pass, Elaine Shi, and Florian Tramer. 2017. Formal abstractions for attested execution secure processors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 260–289.

[79] Torben Pryds Pedersen. 1991. A threshold cryptosystem without a trusted party. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques* (Brighton, UK) *(EUROCRYPT'91)*. Springer-Verlag, Berlin, Heidelberg, 522–526.

[80] Torben Pryds Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology — CRYPTO '91*, Joan Feigenbaum (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 129–140.

[81] Zhizhi Peng, Taotao Wang, Chonghe Zhao, Guofu Liao, Zibin Lin, Yifeng Liu, Bin Cao, Long Shi, Qing Yang, and Shengli Zhang. 2025. A Survey of Zero-Knowledge Proof Based Verifiable Machine Learning. arXiv:2502.18535 [cs.CR] https://arxiv.org/abs/2502.18535
[82] Attia Qammar, Ahmad Karim, Huansheng Ning, and Jianguo Ding. 2023. Securing federated learning with blockchain: a systematic literature review. *Artificial Intelligence Review* 56, 5 (2023), 3951–3985. https://doi.org/10.1007/s10462-022-10271-9
[83] Pian Qi, Diletta Chiaro, Antonella Guzzo, Michele Ianni, Giancarlo Fortino, and Francesco Piccialli. 2024. Model aggregation techniques in federated learning: A comprehensive survey. *Future Generation Computer Systems* 150 (2024), 272–293. https://doi.org/10.1016/j.future.2023.09.008
[84] Youyang Qu, Md Palash Uddin, Chenquan Gan, Yong Xiang, Longxiang Gao, and John Yearwood. 2022. Blockchain-Enabled Federated Learning: A Survey. *Comput. Surveys* 55, 4 (2022). https://doi.org/10.1145/3524104
[85] Taki Hasan Rafi, Faiza Anan Noor, Tahmid Hussain, and Dong-Kyu Chae. 2024. Fairness and privacy preserving in federated learning: A survey. *Information Fusion* 105 (2024), 102198. https://doi.org/10.1016/j.inffus.2023.102198
[86] Sameera Ramasinghe, Thalaiyasingam Ajanthan, Gil Avraham, Yan Zuo, and Alexander Long. 2025. Protocol Models: Scaling Decentralized Training with Communication-Efficient Model Parallelism. arXiv:2506.01260 [cs.LG] https://arxiv.org/abs/2506.01260
[87] Srinath Setty. 2019. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2019/550. https://eprint.iacr.org/2019/550
[88] Geetanjli Sharma, M.A.P. Chamikara, Mohan Baruwal Chhetri, and Yi-Ping Phoebe Chen. 2023. SoK: Systematizing Attack Studies in Federated Learning – From Sparseness to Completeness. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security* (Melbourne, VIC, Australia) *(ASIA CCS '23)*. Association for Computing Machinery, New York, NY, USA, 579–592. https://doi.org/10.1145/3579856.3590328
[89] Ilia Shumailov, Daniel Ramage, Sarah Meiklejohn, Peter Kairouz, Florian Hartmann, Borja Balle, and Eugene Bagdasarian. 2025. Trusted Machine Learning Models Unlock Private Inference for Problems Currently Infeasible with Cryptography. arXiv:2501.08970 [cs.CR] https://arxiv.org/abs/2501.08970
[90] Haochen Sun, Tonghe Bai, Jason Li, and Hongyang Zhang. 2023. zkDL: Efficient Zero-Knowledge Proofs of Deep Learning Training. arXiv:2307.16273 [cs.LG] https://arxiv.org/abs/2307.16273
[91] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. Brendan McMahan. 2019. Can You Really Backdoor Federated Learning? arXiv:1911.07963 [cs.LG] https://arxiv.org/abs/1911.07963
[92] Nick Szabo. 1997. Formalizing and Securing Relationships on Public Networks. *First Monday* 2 (1997). https://api.semanticscholar.org/CorpusID:33773111
[93] Alysa Ziying Tan, Han Yu, Lizhen Cui, and Qiang Yang. 2023. Towards Personalized Federated Learning. *IEEE Transactions on Neural Networks and Learning Systems* 34, 12 (2023), 9587–9603. https://doi.org/10.1109/TNNLS.2022.3160699
[94] Nguyen Truong, Kai Sun, Siyao Wang, Florian Guitton, and YiKe Guo. 2021. Privacy preservation in federated learning: An insightful survey from the GDPR perspective. *Computers & Security* 110 (2021), 102402.
[95] Paul Valiant. 2008. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Theory of cryptography conference*. Springer, 1–18.
[96] Suppakit Waiwitlikhit, Ion Stoica, Yi Sun, Tatsunori Hashimoto, and Daniel Kang. 2024. Trustless Audits without Revealing Data or Models. arXiv:2404.04500 [cs.CR] https://arxiv.org/abs/2404.04500
[97] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. 2020. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization. arXiv:2007.07481 [cs.LG] https://arxiv.org/abs/2007.07481
[98] Zhipeng Wang, Nanqing Dong, Jiahao Sun, William Knottenbelt, and Yike Guo. 2025. zkFLzkFL: Zero-Knowledge Proof-Based Gradient Aggregation for Federated Learning. *IEEE Transactions on Big Data* 11, 2 (2025), 447–460. https://doi.org/10.1109/TBDATA.2024.3403370
[99] Jianqi Wei, Yuling Chen, Xiuzhang Yang, Yun Luo, and Xintao Pei. 2025. A verifiable scheme for differential privacy based on zero-knowledge proofs. *Journal of King Saud University – Computer and Information Sciences* 37, 14 (2025). https://doi.org/10.1007/s44443-025-00028-z
[100] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farokhi Farhad, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. 2019. Federated Learning with Differential Privacy: Algorithms and Performance Analysis. arXiv:1911.00222 [cs.LG] https://arxiv.org/abs/1911.00222
[101] Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. 2022. A survey on federated learning: challenges and applications. *International Journal of Machine Learning and Cybernetics* 14 (11 2022), 513–535. https://doi.org/10.1007/s13042-022-01647-y
[102] Herbert Woisetschläger, Simon Mertel, Christoph Krönke, Ruben Mayer, and Hans-Arno Jacobsen. 2024. Federated Learning and AI Regulation in the European Union: Who is Responsible? – An Interdisciplinary Analysis. arXiv:2407.08105 [cs.AI] https://arxiv.org/abs/2407.08105
[103] Zhibo Xing, Zijian Zhang, Meng Li, Jiamou Liu, Liehuang Zhu, Giovanni Russello, and Muhammad Rizwan Asghar. 2023. Zero-Knowledge Proof-based Practical Federated Learning on Blockchain. arXiv:2304.05590 [cs.CR] https://arxiv.org/abs/2304.05590
[104] Zhibo Xing, Zijian Zhang, Ziang Zhang, Zhen Li, Meng Li, Jiamou Liu, Zongyang Zhang, Yi Zhao, Qi Sun, Liehuang Zhu, and Giovanni Russello. 2026. Zero-Knowledge Proof-Based Verifiable Decentralized Machine Learning in Communication Network: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* 28 (2026), 985–1024. https://doi.org/10.1109/COMST.2025.3561657
[105] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. 2020. VerifyNet: Secure and Verifiable Federated Learning. *IEEE Transactions on Information Forensics and Security* 15 (2020), 911–926. https://doi.org/10.1109/TIFS.2019.2929409
[106] Yi Xu, Changgen Peng, Weijie Tan, Youliang Tian, Minyao Ma, and Kun Niu. 2022. Non-interactive verifiable privacy-preserving federated learning. *Future Generation Computer Systems* 128 (2022), 365–380. https://doi.org/10.1016/j.future.2021.10.017
[107] Xuefei Yin, Yanming Zhu, and Jiankun Hu. 2021. A Comprehensive Survey of Privacy-preserving Federated Learning: A Taxonomy, Review, and Future Directions. *ACM Comput. Surv.* 54, 6, Article 131 (July 2021), 36 pages. https://doi.org/10.1145/3460427
[108] Liangqi Yuan, Ziran Wang, Lichao Sun, Philip S. Yu, and Christopher G. Brinton. 2024. Decentralized Federated Learning: A Survey and Perspective. *IEEE Internet of Things Journal* 11, 21 (2024), 34617–34638. https://doi.org/10.1109/JIOT.2024.3407584
[109] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. 2022. Baloo: Nearly Optimal Lookup Arguments. Cryptology ePrint Archive, Paper 2022/1565. https://eprint.iacr.org/2022/1565
[110] Chaoyu Zhang, Heng Jin, Shanghao Shi, Hexuan Yu, Sydney Johns, Y. Thomas Hou, and Wenjing Lou. 2025. Enabling Trustworthy Federated Learning via Remote Attestation for Mitigating Byzantine Threats. arXiv:2509.00634 [cs.CR] https://arxiv.org/abs/2509.00634
[111] Xianglong Zhang, Anmin Fu, Huaqun Wang, Chunyi Zhou, and Zhenzhu Chen. 2020. A Privacy-Preserving and Verifiable Federated Learning Scheme. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 1–6. https://doi.org/10.1109/ICC40277.2020.9148628
[112] Xiaoli Zhang, Fengting Li, Zeyu Zhang, Qi Li, Cong Wang, and Jianping Wu. 2020. Enabling Execution Assurance of Federated Learning at Untrusted Participants. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 1877–1886. https://doi.org/10.1109/INFOCOM41043.2020.9155414
[113] Lianying Zhao, He Shuang, Shengjie Xu, Wei Huang, Rongzhen Cui, Pushkar Bettadpur, and David Lie. 2024. A Survey of Hardware Improvements to Secure Program Execution. *ACM Comput. Surv.* 56, 12, Article 311 (Oct. 2024), 37 pages. https://doi.org/10.1145/3672392

Ethical Considerations

In this section we discuss the ethical impact and consequences of our research. We first identify the stakeholders who may be affected by this work and discuss potential positive and negative impacts. We then describe mitigations to reduce the risk of negative impacts and conclude by justifying the need for this research.

Identified Stakeholders. The results of our Systematization of Knowledge paper can potentially impact four groups of stakeholders: (1) researchers, including both the authors of the papers we analyzed and those working on machine learning and verifiable technologies who are seeking up-to-date state-of-the-art solutions; (2) companies seeking verifiable federated learning solutions for potential production deployments; (3) developers interested in verifiable FL for contributions and collaborations; (4) data owners and entities whose data contributed to and were included in the datasets.

Impact of the research. As a Systematization of Knowledge, we believe that our research can have a mainly positive impact on the stakeholders mentioned in the previous paragraph. Our work presents a rigorous analysis of each surveyed study, comparing them across multiple dimensions. Furthermore, for each work, we assess how closely it aligns with what we define as a complete

verifiable framework. Stakeholders of the group (1): The authors of the works we included would have the possibility to compare theirs with the other SOTA approaches, assessing what's possible to improve, where they failed or where they excelled. Researchers working on machine learning and verifiability would be aided in identifying relevant literature and obtaining an initial, high-level overview of the papers of interest. Stakeholders of the group (2): Companies seeking solutions for deployment would have access to a clear overview of each protocol, along with useful comparisons that they could leverage to choose the best-fitting solution. Stakeholder of the group (3): Developers interested in contributing would have access to the most recent works, analyzed and compared, which would help them choose which line of work to pursue based on their interests.
On the other hand, as a downside, the SoK may highlight certain works that emerge as the most complete under our unbiased analysis, while others may receive less attention because they provide fewer features. This follows from the intrinsic nature of an SoK paper, whose goal is not only to survey prior work but also to provide a critical assessment. Furthermore, a potential risk concerns stakeholders of the group (4), who might be exposed to data leakage due to known attacks in the Federated Learning context.

Mitigations. In light of the risks described above, we deliberately avoid personal opinions or overly assertive statements (e.g., "Paper A is better than Papers B and C") related to the effectiveness of proposed mitigation techniques. Instead, we present the evidence in an *unbiased* manner so that readers can draw their own conclusions based solely on the reported data. We emphasize that our analysis (except for the "ease of implementation" assessment in Figure 3, which is done according to our subjective evaluation, but is not related to security anyway) relies only on objective, verifiable information contained in the reviewed papers, and that all comparisons follow directly from these documented elements rather than from subjective assumptions, thereby making our framework reproducible.
Concerning data leakage risks affecting stakeholders of the group (4): all datasets used in the works we analyzed are open source, and the majority of the studies implement privacy-preserving techniques, such as Differential Privacy and secure aggregation, to further reduce the attack surface. For stakeholders of the group (2), we strongly recommend leveraging privacy-preserving techniques and paying particular attention to data privacy throughout the entire framework. We also caution against treating FL, or even privacy-preserving FL, as a privacy guarantee by itself. Privacy mechanisms such as secure aggregation and differential privacy should be deployed together with integrity checks, since poisoned updates, incorrect admission, selective dropping, or unverifiable aggregation can enlarge the practical attack surface against the very data these mechanisms are meant to protect.

Justification for the research. Following recent advances in verifiability technologies (e.g., zero-knowledge proofs and TEEs), these techniques have begun to find concrete applications in machine learning. We believe this trend can be a meaningful breakthrough for the trustworthiness of end-to-end ML processes. Evidence of this momentum is the growing interest from the research community: many approaches have been proposed only recently, especially in federated contexts, leading to fragmentation in threat models and, in some cases, the introduction of complex and non-general notation. For this reason, we believe this is the right time to provide a Systematization of Knowledge on this topic, bringing structure to the emerging landscape and helping practitioners and researchers identify the most suitable solutions for their specific requirements.

This work did not go through an external ethics panel.

Reproducibility

This Systematization of Knowledge paper is based exclusively on published, publicly available literature. We did not use or generate experimental artifacts (e.g., source code, datasets, binaries, or benchmarks). The contribution is a novel classification framework for Verifiable Federated Learning and a synthesis and classification of prior work, which can be assessed from the cited sources.

Images. All images included in this paper were generated by the authors or derived from public work:

- Figure 1 and Figure 2 were created using the draw.io diagramming tool (https://www.drawio.com/) without the use of AI prompting.
- Some clipart images in Figure 1 were taken from Flaticon (https://www.flaticon.com/) with permissible license. Attribution:
  (1) https://www.flaticon.com/free-icons/boxes
  (2) https://www.flaticon.com/free-icons/machine-learning
  (3) https://www.flaticon.com/free-icons/computer
- Figure 3 was generated using the Matplotlib Python library (https://matplotlib.org/).

You can check this repository (https://github.com/xxxx-yyyy-zzzz/SoK_Material) for the draw.io diagrams and Python code we used.

Literature search and selection procedure reproducibility. To broaden coverage beyond the works already known, we performed a structured search using multiple scholarly search engines, including Google Scholar. We used combinations of keywords such as: "verifiable federated learning," "verifiability in federated learning," "verifiable TEE-based federated learning," "blockchain-based federated learning," and related variants/synonyms. Candidate papers were collected from search results and then filtered according to the scope and framework defined in the paper. To further broaden coverage beyond conventional keyword search, we also used AI-assisted "deep research" tools (ChatGPT, Gemini, and Claude) as supplementary discovery mechanisms to identify potentially relevant papers and venues.

Data usage. All quantitative and qualitative information reported in this paper is extracted from publicly available published works and can be independently verified by consulting the cited sources. In particular, the overhead figures reported in Table 7 were manually extracted from the evaluation/experiment sections of the corresponding papers (cited in the table). We report the values as stated by the original authors, without re-running experiments, and we rely on the original publications for authenticity.

A Tables

A.1 Black-box interfaces for the Verifiable Framework

Table 8: Black-box interfaces referenced by our framework.

Interface	Entity	Inputs		Public outputs
		Private	*Public*	
$\mathcal{F}_{\mathsf{TP}}$.Append/Read[1]	ledger	–	Round records	Transcript state; record IDs; inclusion proofs
$\mathcal{F}_{\mathsf{DB}}$.Bind[2]	DO or CL	D_i	PolicySpec	Data handle h_i^D
$\mathcal{F}_{\mathsf{CL}}$.Train[3]	CL	D_i	ctx_t, h_i^D	$\theta_{i,t}$
$\mathcal{F}_{\mathsf{V}}$.VerifyClient[4]	$\mathcal{V}$	–	ctx_t, i, h_i^D, $h_{i,t}$, $m_{i,t}$, $e_{i,t}^{\mathsf{CL}}$	Decision bit $b \in \{0,1\}$
$\mathcal{F}_{\mathsf{AG}}$.Aggregate[5]	AG	$\{\theta_{i,t}\}_{i\in S_t}$ ⋆	ctx_t, S_t, $\{h_{i,t}\}_{i\in S_t}$	$(h_t^{\mathsf{agg}}, \mathsf{ckpt}_t, e_t^{\mathsf{AG}})$
$\mathcal{F}_{\mathsf{V}}$.VerifyAgg[4]	$\mathcal{V}$	–	ctx_t, S_t, $\{h_{i,t}\}_{i\in S_t}$, h_t^{agg}, ckpt_t, e_t^{AG}	Decision bit $b \in \{0,1\}$

[1]Round records include submissions, admissions, aggregation, and finalization. Append is overloaded: clients may append individual submissions (Algorithm 1), while the aggregator appends the complete round record (Algorithm 2). This interface provides a shared transcript view and, when supported, proofs that a record is included. [2]Binds a declared data usage policy to a dataset by producing a public commitment or digest, while the underlying dataset D_i remains private. Here, PolicySpec denotes the data-relevant subset of Spec (e.g., consent rules, sampling constraints, preprocessing requirements). The binding is typically performed once during setup and the resulting handle h_i^D is registered in Setup or maintained out-of-band. [3]Core training interface that takes the private dataset D_i and the public round context (which includes ckpt_{t-1}) and returns the updated model weights $\theta_{i,t}$. The subsequent encoding, metadata derivation, and evidence generation steps (Algorithm 1, lines 2–4) are not part of this interface. [4]Verification operates entirely on public inputs. The verifier class $\mathcal{V}$ is defined in Section 3.1. Outputs accept/reject decisions to justify Adm_t and Fin_t. Evidence artifacts $e_{i,t}^{\mathsf{CL}}$ and e_t^{AG} may be zero-knowledge proofs, TEE attestation quotes, or other verifiable artifacts as determined by the instantiation. [5]The aggregator can be a centralized server, a committee, a TEE-backed enclave, or smart contract logic that runs over the admitted set S_t. ⋆Under secure aggregation, raw updates remain private (masked or encrypted), otherwise they may be visible to the aggregator.

A.2 Overview of extra security guarantees obtained thanks to verifiability properties

	Threat Model	Verifiability-enabled mitigations
zkFL [98]	Byzantine aggregator; honest clients	Global model tampering; selective dropping
zkFDL [2]	Byzantine aggregator; honest clients	Global model tampering; selective dropping
zkPPFL[103]	Byzantine aggregator; lazy-but-curious clients[a]	Model poisoning *; free riding; global model tampering; selective dropping
ACORN [11]	Byzantine aggregator; Byzantine clients	Model poisoning
VPFL [66]	Byzantine aggregator; Byzantine clients	Poisoning attacks*; selective dropping
Federify [56]	Byzantine and curious aggregator; Byzantine clients	Model poisoning *; free riding; global model tampering
VerifBFL [13]	Byzantine and curious aggregator; Byzantine and curious clients	Free riding; global model tampering
V-FLEX [20]	Byzantine and curious aggregator; Byzantine clients	Global model tampering; selective dropping
Kalapaaking et al. [51]	Byzantine and curious aggregator; honest clients	Global model tampering
Sentinel [110]	Trusted aggregator; Byzantine clients	Poisoning attacks *
VerifiableFL [47]	Byzantine and curious aggregator; Byzantine clients	Poisoning attacks; global model tampering; selective dropping; free riding
Li et al. [62]	Honest but curious aggregator; Byzantine clients	Model poisoning; free riding
SecureFL [61]	Honest but curious aggregator; honest but curious clients	Model poisoning *

[a] The authors use this term to indicate an honest-but-curious client that can also perform free-riding attacks.
* Indicates that the attack is only partially mitigated.

B Algorithmic analyses

This appendix provides an algorithmic analysis of each work deemed relevant to the objectives of this Systematization of Knowledge. For works whose algorithms are particularly sophisticated, we include brief clarifying remarks immediately following the corresponding algorithm.

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)
- q_i : a quote by enclave i
- 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.

B.1 Li et al. [62]

Algorithm B.1

Setup: prior to training, each client commits to its quantized local dataset. The task publisher specifies a sampling rate β to select training epochs for verification.
Per round t, each client C_i:

(1) Receives the current global model $\theta_{global,t-1}$ from the server.
(2) Performs local training obtaining the update $\theta_{i,t}$.
(3) Encrypts and uploads the local update and training data to the server.

For each sampled epoch, the server:

(1) Upload the encrypted training material to the TEE.
(2) Re-executes the training process inside the enclave:
   - non-linear operations are executed inside the enclave;
   - linear operations are securely offloaded to a co-located GPU via a custom protocol named SETO.

After the retraining procedure, the enclave validates the client's behavior by comparing the model update obtained from the server-side retraining of the sampled epochs with the update submitted by the client. If the two match, the server accepts the contribution for aggregation; otherwise, the client is deemed misbehaving and its update is rejected.

SETO, the paper's main contribution, is a protocol designed to offload linear computations from the enclave to the GPU, and it acts as a one-time-pad encryptor; considering $x \in \mathbb{F}^n$ the data to be sent to the GPU, the enclave:

(1) Pick $r \in \mathbb{F}^n$ from a Pseudo Random Number Generator (PRNG).
(2) $c_x = Enc(x, r) = x + r \in \mathbb{F}^n$
(3) Prepare a decryption mask $\alpha = r * w$, where $w \in \mathbb{F}^n$ is the weights vector.
(4) Send $(p(c_x), p(w))$ to the GPU, where p is pseudo random permutation (PRP) function.

Now the GPU performs $c_y^p = c_x^p * w^p = Enc(y^p)$ and sends back c_y^p to the enclave. The p as a superscript denote "permuted".

The enclave can now decrypt leveraging the mask α in the following way:

$$\begin{aligned} y &= p^{-1}(c_y^p) - \alpha \\ &= c_y - \alpha \\ &= (x + r)\, w - r\, w \\ &= w\,(x + r - r) \\ &= x\, w\,. \end{aligned}$$

This process is repeated for each linear layer.

B.2 zkFL[98]

Algorithm B.2

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 [80] 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 $\theta_{global,t}$.

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

B.3 V-FLEX [20]

Algorithm B.3

Setup:

(1) Clients jointly execute a DKG protocol to obtain a CKKS public key pk and threshold secret shares $\{sk_i\}$.
(2) Client and server enclaves perform remote attestation to establish mutual trust and secure channels.

At each round t:

(1) Each client C_i trains locally on D_i, producing a model update $\theta_{i,t}$.
(2) Client C_i encrypts the update using CKKS, $c_{i,t} \leftarrow \mathrm{Enc}_{pk}(\theta_{i,t})$, and computes inside the enclave an integrity tag $h_{i,t} \leftarrow H(c_{i,t})$ using a SWIFFT-based homomorphic hash.
(3) The pair $(c_{i,t}, h_{i,t})$ is sent to the server.
(4) Inside the enclave, integrity checks verify that each $c_{i,t}$ matches its $h_{i,t}$ to prevent data tampering before aggregation.
(5) Aggregation:
   - On the untrusted host, the server performs the computationally intensive homomorphic sum: $c_{\mathrm{agg},t} = \sum_{i=1}^{n} c_{i,t}$.
   - Simultaneously, the enclave computes the aggregate hash: $h_{\mathrm{agg},t} = \sum_{i=1}^{n} h_{i,t}$.
(6) The server enclave generates a remote attestation quote that binds $h_{\mathrm{agg},t}$ to the enclave's identity.
(7) Clients verify the enclave's quote and check the integrity of the host's work by verifying:
$$H(c_{\mathrm{agg},t}) \stackrel{?}{=} h_{\mathrm{agg},t}$$
If valid, they collaboratively threshold-decrypt $c_{\mathrm{agg},t}$ to recover the global update.

The verifiability guarantee of V-FLEX relies on the homomorphic consistency of the hash function:

$$H\left(\sum_i c_{i,t}\right) = \sum_i H(c_{i,t}),$$

which allows clients to detect any deviation from honest aggregation, such as ciphertext substitution, omission, or tampering after submission. TEEs ensure that integrity verification and key-handling logic execute as intended, but correctness ultimately depends on the security of the enclave hardware.

B.4 Sentinel[110]

Algorithm B.4

Per round t, each client C_i:

(1) Receives from the server the global model $\theta_{global,t-1}$ and an attestation challenge $ch_{i,t}$.
(2) Initializes the trusted training recorder inside the TEE and starts a new measurement session.
(3) Trains locally on dataset D_i in the normal world to obtain $\theta_{i,t}$, while the instrumented training code interacts with the TEE to record:
   - the executed control-flow graph $CF_{i,t}$,
   - the usage of critical variables $CV_{i,t}$
(4) Finalizes the measurement session by invoking the TEE to generate an attestation report:
$$\sigma_{i,t} = \mathrm{Sign}\big(sk_{TEE}, H\big(CF_{i,t}, CV_{i,t}, \theta_{i,t}, ch_{i,t}\big)\big).$$
(5) Sends to the server: $\big(\theta_{i,t}, CF_{i,t}, CV_{i,t}, \sigma_{i,t}\big)$.

On the server side:

(1) Verifies the TEE attestation $\sigma_{i,t}$.
(2) Checks that $CF_{i,t}$ and $CV_{i,t}$ match the expected reference execution (CF^*, CV^*).
(3) Aggregates $\theta_{i,t}$ into the global model only if all verification checks succeed.

B.5 ACORN [11]

Algorithm B.5

Per round t, each client C_i:

(1) Trains on its local data D_i obtaining the update $\theta_{i,t}$.
(2) Engages in *ShareSeed*, a procedure to obtain seeds and shares needed for the masking procedure for secure aggregation.
(3) Computes a masking value m_i from the obtained seeds.
(4) Encodes the update for secure aggregation:
$$\hat{\theta}_{i,t} = Encode(m_i, \theta_{i,t}).$$
(5) Commits to m_i and $\theta_{i,t}$ obtaining d_{m_i} and $d_{\theta_{i,t}}$
(6) Computes proofs
$$\{\pi_{i,t}^{Enc(m_i,\theta_{i,t})}, \pi_{i,t}^{0\le\theta_{i,t}<t}, \pi_{i,t}^{valid(\theta_{i,t})}\}$$
of encoding, smallness and validity. The latter represents a generic constraint.
(7) Sends to the server
$$(d_{m_i}, d_{\theta_{i,t}}, \pi_{i,t}^{Enc(m_i,\theta_{i,t})}, \pi_{i,t}^{0\le x_i<t}, \pi_{i,t}^{valid(\theta_{i,t})})$$
(8) The server verifies the proofs and eventually discard the updates that don't respect the constraints, if any.
(9) The server and the clients engage in *RecoverAggKey*, which allows the server to recover the aggregate key m at the end of the protocol.
(10) The server outputs the aggregated model
$$\sum_{i\in n} \theta_{i,t} = (Decode(m, \sum_{i\in n} \theta_{i,t}))$$

The first proof, $\pi_{i,t}^{\mathrm{Enc}(m_i,\theta_{i,t})}$, guarantees that $\hat{\theta}_{i,t} = \mathrm{Encode}(m_i, \theta_{i,t})$ and binds the values m_i and $\theta_{i,t}$ to those committed in d_{m_i} and $d_{\theta_{i,t}}$. The second proof, $\pi_{i,t}^{0\le\theta_{i,t}<\tau}$, enforces a range constraint on $\theta_{i,t}$, while the third proof, $\pi_{i,t}^{\mathrm{valid}(\theta_{i,t})}$, encodes additional application-specific validity constraints, such as L_0, L_2, and L_∞ bounds on the client updates. In particular, enforcing an L_2-norm bound can help mitigate certain classes of model-poisoning attacks by limiting the magnitude of malicious updates [91].

B.6 zkFDL[2]

Algorithm B.6

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 [5]:
$H_{i,t} = MiMC7(\theta_{i,t}), \forall i \in [1,n]$ 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 $Contract_p$.

Verifiable aggregation, 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) $H_{\theta_{global,t}} = MiMC7(\theta_{global,t})$

Then, 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 allows each client to verify that its gradient was included, avoiding selective dropping.

B.7 zkPPFL [103]

Algorithm B.7

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 $\bar{c_{sum}} = \frac{\mathrm{Dec}_{\mathrm{sk}_h}(c_{sum})}{n}$.

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.

B.8 Federify [56]

Algorithm B.8

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.

Proof of training, 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 $\zeta_{j,t}$

B.9 VPFL [66]

Algorithm B.9

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 Bulletproof [22]) 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.

B.10 SecureFL [61]

Algorithm B.10

Per round t (training and aggregation under TEEs)

(1) The aggregator broadcasts the model
$$\theta_{\text{global},t-1} = \left(\theta^{(1)}_{t-1}, \theta^{(2)}_{t-1}\right)$$
to each worker over the established secure channel, where $\theta^{(1)}_{t-1}$ and $\theta^{(2)}_{t-1}$ collect the parameters of the early and late layers, respectively.
(2) Due to TrustZone memory limits, SecureFL partitions the DNN at a split index ℓ: layers $1, \ldots, \ell$ with parameters $\theta^{(1)}$ run in the normal-world, while layers $\ell + 1, \ldots, L$ with parameters $\theta^{(2)}$ and their training logic execute inside the secure world.
(3) After local training on data D_i, the updated sensitive-layer parameters $\theta^{(2)}_{i,t}$ are encrypted inside the TrustZone before exporting them to the host
(4) The host-side coordinator combines them with the non-sensitive parameters $\theta^{(1)}_{i,t}$ and encrypts the full local model update $\theta_{i,t} = \left(\theta^{(1)}_{i,t}, \theta^{(2)}_{i,t}\right)$ for upload.
(5) The aggregator buffers encrypted updates in the host but performs decryption, aggregation, and re-encryption inside the SGX enclave.
(6) Workers receive the encrypted aggregated model $\theta_{\text{global},t}$; non-sensitive layers $\theta^{(1)}_{\text{global},t}$ are decrypted and installed in the host, while sensitive layers $\theta^{(2)}_{\text{global},t}$ are handed to the TrustZone secure world for decryption and update of the secure model portion.

B.11 VerifBFL [13]

Algorithm B.11

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, 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.

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 [59]; 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.

B.12 VerifiableFL[47]

Algorithm B.12

Per round t, the protocol executes a sequence of tasks, each running inside an exclave and producing a runtime attestation proof called an Exclave Data Record (EDR).
Each client C_i:

(1) Receives the global model $\theta_{global,t-1}$ from the model provider.
(2) Executes training inside an exclave, obtaining $\theta_{i,t}$, while reading its integrity-protected dataset D_i from secure storage. An EDR $v^{(T)}_{i,t}$ for the training task is produced.
(3) Executes a DP task inside another exclave to obtain a DP-protected update $\hat{\theta}_{i,t}$, producing an additional EDR $v^{(D)}_{i,t}$.
(4) Endorses each EDR by signing it with a long-term issuer key and uploads the endorsed EDRs to an EDR storage service.
(5) Sends $\hat{\theta}_{i,t}$ to the model provider via the FL framework.

On the model provider:

(1) Executes aggregation inside an exclave over the received client contributions, producing an EDR $v^{(A)}_t$ for the aggregation task.
(2) Executes the model update task inside an exclave, obtaining the final model $\theta_{global,t}$ and producing an EDR $v^{(M)}_t$.
(3) Endorses and uploads the two EDRs to EDR storage, and distributes $\theta_{global,t}$ to clients for round t+1.

In an offline auditing phase, a third-party auditor collects all EDRs and verifies their hardware-backed attestation signatures, discarding any invalid records. The auditor then constructs the Exclave Dataflow Graph (EDG), where each vertex corresponds to an EDR and directed edges link records whose output hashes match subsequent input hashes. Finally, the auditor evaluates the verifiable claims by checking (i) that each task is associated with an allowed code measurement and (ii) that the resulting EDG is structurally and dataflow-consistent across rounds, e.g., ensuring that DP tasks are not bypassed, client contributions are not dropped, and dataset commitments remain stable.

B.13 Kalapaaking et al. [51]

Algorithm B.13

Setup: each client C_i establishes secure channels with the SGX enclaves of the p blockchain nodes, obtaining a set of symmetric keys $K_i = \{k_{i,1}, \ldots, k_{i,p}\}$.
Per round t, each client C_i:

(1) Trains locally obtaining a model update $\theta_{i,t}$.
(2) Encrypts the update for each blockchain node enclave: $c_{i,j} = Encrypt(k_{i,j}, \theta_{i,t}), \quad \forall k_{i,j} \in K_i$
(3) Sends each ciphertext $c_{i,j}$ to the corresponding blockchain node j.

Each blockchain node j:

(1) Collects encrypted updates from all clients: $\{c_{1,j}, \ldots, c_{n,j}\}$.
(2) Inside its SGX enclave, decrypts all updates and aggregates them to obtain a local global model $\theta^j_{global,t}$.
(3) Generates a report R_j for the aggregation, which is signed by the local quoting enclave to produce a quote q_j.
(4) Encrypts the quote as $c_{q_j} = Encrypt(pk_{IAS}, q_j)$ and broadcasts it to the other blockchain nodes.

A consensus mechanism is executed among the blockchain nodes to agree on the global model, which is then redistributed to the clients for the next round.

Verifiability in this paper relies on SGX-backed aggregation, further reinforced by a custom consensus mechanism:

(1) Each blockchain node submits the received encrypted quotes to the Intel Attestation Service (IAS) to validate the quotes (i.e., enclave identity/measurement) and extract the corresponding attested global models.
(2) Nodes compute an hash of each attested global model.
(3) Consensus is reached if all hashes match, otherwise the global model whose hash obtains a majority is selected.

C Adjacent lines of research

We now acknowledge adjacent lines of research that lie outside the strict scope of federated learning yet supply critical conceptual foundations and cryptographic tools for verifiability.

co-SNARKs in verifiable FL

In collaborative zk-SNARKs, mutually distrustful parties $P_1, \ldots, P_N$, each holding a private witness share w_i, jointly produce a single succinct proof π attesting that a public relation $R(x; w_1, \ldots, w_N)$ holds, without revealing any w_i or re-concentrating the witness at a single party. Ozdemir and Boneh provided the first empirical evidence that pairing-based proof systems (Groth16, Plonk, Marlin) can be efficiently transformed into multi-party computation protocols [76]. Follow-up work develops malicious-secure templates that emulate zk-SNARK provers via MPC, clarifying which subroutines (MSM, FFT, product checks) admit secure distribution [39]. Co-SNARKs have also been applied to multi-party smart-contract execution and collaborative analytics, preserving input privacy while yielding a single succinct proof for on-chain or audit use [40]. Interestingly, the co-SNARK abstraction aligns naturally with FL: clients i hold private data D_i and local updates θ_i, while the aggregation rule (e.g., $\theta_{global} = \frac{1}{n}\sum \theta_i$ with admissibility checks) defines a public relation $R(x; \theta_1, \ldots, \theta_n)$. A co-SNARK could let clients jointly prove that aggregation was performed correctly without a trusted aggregator and without exposing θ_i.

Decentralized model-parallel training

While conventional FL can be referred to as *data-parallel*, 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 [65]. Although very recent, this line of research is promising for handling asynchrony and constrained network links, but partitioning the models introduces also new challenges (e.g. staleness from asynchronous queues). Unlike a FL's periodic aggregation, communication here occurs at every step and is highly sensitive to latency and bandwidth. To compensate for stale updates, delay-aware optimizers have been proposed [3]. Also, transmitting large activations can be addressed with communication-efficient designs enabling billion-parameter models to be trained over ordinary internet links [86]. Notably, at partition boundaries (i.e. where one model partition hands activations or optimizer metadata to the next) the required guarantees could be cast as a form of proof of aggregation which we describe in Algorithm 2.