Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

FL-Compliance-Guard: Machine-Checkable Compliance Evidence for Federated Learning Pipelines

Ahsan Umar

232552@icp.edu.pk

Islamia College University

Research Article

Keywords:

Posted Date: November 11th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10856041/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

FL-Compliance-Guard: Machine-Checkable Compliance Evidence for Federated Learning Pipelines

Ahsan Umar
Islamia College University, Peshawar, Pakistan
Corresponding author: Ahsan Umar
Email: 232552@icp.edu.pk

Abstract Federated learning (FL) is considered for healthcare, finance, and other settings in which training data cannot be pooled. Distributed training, however, does not by itself show that an operator's regulatory and governance controls remained in force. We present FL-Compliance-Guard, a proof-of-concept framework that encodes a deployment-specific set of technical controls as *machine-checkable predicates*. Each predicate is linked to provisions of the GDPR, the EU AI Act, or the HIPAA Security Rule that it is intended to support; this linkage provides traceability, not an automated legal verdict. A static checker evaluates the declared configuration before deployment, and a runtime monitor checks per-round events against the admitted configuration. The resulting evidence is stored in a hash-chained ledger, and a conservative privacy accountant tracks the recorded differential-privacy parameters. The monitor reads bookkeeping state, consumes no random draws, and does not alter model updates. We evaluate the design with 20 simulated clients on the digits classification task using FedAvg and DP-FedAvg. The static checker detects all 18 seeded misconfigurations in the stated catalog, with a mean check time of 0.011 ms and no alerts on 500 benign perturbations. The runtime monitor detects all six injected operational faults at the next applicable check and raises no alerts in the clean runs. Monitored and unmonitored runs have bit-identical accuracy trajectories. Full checking takes monitoring time equal to 4.4–15.2% of measured training time; ledger-only recording takes 1.0%. Component ablations show which checks account for each detection. These results establish coverage for the evaluated fault model, rather than legal compliance in general.

Keywords: federated learning; regulatory compliance; runtime verification; differential privacy; auditability; EU AI Act; accountability

1 Introduction

Federated learning (FL) allows a population of clients to train a shared model without exporting their raw data, replacing data centralization with the exchange of model updates [1, 2]. This architecture is attractive when data pooling is restricted, as in multi-site medical imaging [3, 4] and some cross-institutional or on-device applications. It does not, however, remove the operator's duties concerning security, accountability, documentation, and oversight. Those duties are usually assessed through policies and audit documents, whereas the training pipeline is controlled through configuration and code. The resulting mismatch makes it difficult to determine whether a declared control was active throughout a distributed training run.

Several legal regimes motivate such evidence. The EU Artificial Intelligence Act entered into force on 1 August 2024 [5]. Following the 2026 amendment, the requirements for high-risk systems in Articles 9–15 are scheduled to apply from 2 December 2027 for Annex III systems and from 2 August 2028 for systems embedded in regulated products [6]. The GDPR addresses data protection by design (Art. 25), security of processing (Art. 32), and international transfers (Ch. V) [7]. For regulated health data, the

HIPAA Security Rule includes audit-control and transmission- security standards [8]. The applicability of these provisions depends on the system, the data, and the organization's legal role. Legal analysis therefore remains a human responsibility [9]; software can test only the technical predicates chosen to represent part of that analysis.

This paper focuses on controls that can be evaluated from a pipeline's configuration or event stream. Examples include transport encryption, privacy-mechanism parameters, an operator-defined privacy cap, log retention, participant admission and revocation, and reconciliation between update receipts and recorded participation. These checks do not interpret the law. They evaluate an explicit, reviewable policy that records how an operator has chosen to implement selected legal and governance requirements. The approach follows the broader compliance-by-design principle [10] and addresses a specific FL problem: declared settings can diverge from the state of a long-running distributed job after admission.

FL-Compliance-Guard implements this approach in three parts. A catalog defines 18 machine-checkable control predicates and links each predicate to its policy or regulatory rationale. A static checker evaluates the declared pipeline configuration before training, while a runtime monitor evaluates event-based predicates and binds each round to the admitted runtime configuration. A hash-chained ledger records the evidence [11, 12], and a privacy accountant derives an (ε, δ) bound from the mechanism parameters recorded for each round. An auditor can then reproduce the profile checks, subject to the framework's trust assumptions.

The monitor is also designed to be observational. It consumes no randomness, reads no raw data, and inspects only bookkeeping artifacts such as participation records, configuration hashes, mechanism parameters, and checkpoint commitments. With identical seeds, monitored and unmonitored runs produce bit-identical accuracy trajectories in the reference implementation. This check separates the cost of instrumentation from changes in the learning procedure.

Contributions. This paper makes the following contributions:

1. **Traceable control catalog.** A reviewable mapping from 18 technical control predicates to the regulatory provisions and deployment policies that motivate them, packaged as a declarative profile (Section 4).
2. **Two-checkpoint enforcement architecture.** A static configuration checker and a per-round runtime monitor with config-hash binding, backed by a hash-chained audit ledger and an executable conservative Gaussian-RDP accountant, with an empirical non-interference check in the reference implementation (Section 4).
3. **Fault-injection and fuzzing evaluation.** 18 seeded static misconfigurations and 6 injected operational faults, all detected, with no alerts on 500 benign fuzzed configurations, detection at the next applicable checkpoint, and a component ablation over six degraded system variants (Section 6).

Paper organization. Section 2 fixes notation and states the verification problem. Section 3 reviews related work. Section 4 presents the framework. Section 5 describes the experimental design and Section 6 reports outcomes and ablations. Sections 7–9 discuss scope, limitations, and conclusions.

2 Background and Problem Formulation

2.1 Federated optimization and differential privacy

We adopt the standard cross-device FL setting. A coordinator (server) and K clients collaborate to minimize a global objective $F(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w)$, where n_k is the number of samples held by client k. At round t the server samples a set $S_t \subseteq [K]$ of $m = |S_t|$ participants, broadcasts the current model w_t, and each selected client $k \in S_t$ performs E local epochs of SGD from w_t, returning an update $\Delta_{t,k} = w_{t,k} - w_t$. The server aggregates by weighted averaging,

$$w_{t+1} = w_t + \sum_{k \in S_t} \frac{n_k}{\sum_{j \in S_t} n_j} \Delta_{t,k}. \tag{1}$$

Under DP-FedAvg [1, 13], each client update is ℓ_2-clipped to a norm bound C,

$$\tilde{\Delta}_{t,k} = \Delta_{t,k} \min\big(1,\ C/\|\Delta_{t,k}\|_2\big), \tag{2}$$

and Gaussian noise is added to the aggregated sum before normalization,

$$w_{t+1} = w_t + \tfrac{1}{m}\Big(\textstyle\sum_{k\in S_t} \tilde{\Delta}_{t,k}\Big) + \mathcal{N}\big(0,\ \sigma^2 C^2 \mathbf{I}/m^2\big), \tag{3}$$

where σ is the noise multiplier. We use client-level add/remove adjacency, treating an absent client's clipped contribution as zero. Rather than claim amplification from the simulator's fixed-size client sampling, the accountant applies the Gaussian mechanism's per-round Rényi-DP bound $\varepsilon_{\mathrm{RDP}}(\alpha) = \alpha/(2\sigma^2)$ and composes it over T rounds [14]. The standard conversion gives

$$\varepsilon(\delta) \;=\; \min_{\alpha>1} \left[\, T \cdot \tfrac{\alpha}{2\sigma^2} \;+\; \tfrac{\ln(1/\delta)}{\alpha-1} \right]. \tag{4}$$

This bound deliberately ignores privacy amplification and is therefore looser than accountants that model a specified sampling process explicitly [15]. The conservative choice avoids attaching a sampling guarantee to a simulator that selects a fixed number of clients per round.

2.2 Pipeline artifacts and the compliance question

A deployed FL pipeline produces two classes of artifact that a compliance process can legitimately inspect. The first is the *declared configuration*: a structured document (in practice, YAML or equivalent) describing the transport security model, storage encryption, DP mechanism and parameters, aggregation scheme, admission policy, retention periods, incident-handling hooks, oversight hooks, and documentation obligations. The second is the *runtime event stream*: the per-round facts of the actual execution, including which clients participated, which mechanism parameters were in effect, which artifacts were committed, and with what hashes. The declared configuration is the natural target of pre-deployment review; the event stream is the only witness of what actually happened during a long, failure-prone training run.

The verification question is: *given a policy profile* Π*, a declared configuration* c*, and a runtime event stream* $E = (e_1, \ldots, e_T)$*, decide whether the observed artifacts satisfy* Π *before and during training, and produce evidence for an external audit.* Each profile control is represented by a decidable predicate π over c, over E, or over both. Static satisfaction is the conjunction over configuration predicates,

$$\mathrm{Sat}_{\mathrm{static}}(c, \Pi) \;=\; \bigwedge_{\pi\in\Pi_c} \pi(c), \tag{5}$$

and runtime satisfaction is the conjunction of per-round predicates over the observed stream,

$$\mathrm{Sat}_{\mathrm{runtime}}(E, \Pi) \;=\; \bigwedge_{t=1}^{T} \bigwedge_{\pi\in\Pi_E} \pi(e_t). \tag{6}$$

Three engineering issues remain: (i) *binding*, so that runtime-relevant settings can be compared with those admitted; (ii) *evidence integrity*, so that later modification is detectable relative to a trusted chain anchor; and (iii) *attribution*, so that a failed predicate can be traced to its policy rationale. Section 4 describes the corresponding configuration commitment, hash chain, and control catalog.

2.3 Threat and failure model

We consider three actors whose behaviour the framework must make visible. The *negligent operator* misconfigures the pipeline inadvertently: DP is switched off to speed up a debug run and never re-enabled, a retention period is shortened to reclaim disk space, a client from a non-approved jurisdiction is added by a script. The *drifting system* silently diverges from its declared posture through crashes and partial failures: a round completes but its ledger record is lost in a restart; three ledger entries are deleted by a buggy log-rotation job; checkpoint bytes on disk are corrupted; noise injection fails silently after an update.

The *insider* deliberately violates policy in ways that look plausible: re-admitting a revoked participant, over-reporting participation counts, or deploying a model artifact that fails integrity verification. We do not consider a malicious server that forges all artifacts and their commitments (such a threat model requires cryptographic attestation or trusted hardware [16, 17]), nor attacks on the learning process itself such as poisoning, which are orthogonal and well studied [18–20]. The framework provides detection and evidence rather than prevention. Alerts are inputs to an operator's oversight process; they do not themselves satisfy the legal requirements for human oversight.

3 Related Work

3.1 FL systems and frameworks

Production and research FL frameworks—Flower [21], FedML [22], FedScale [23], and the large-scale system described by Bonawitz et al. [24]—provide abstraction layers for client sampling, aggregation, and communication, and several expose hook points that our monitor could attach to. Robustness benchmarking efforts have begun to evaluate aggregation strategies under system failures and attacks: the FLResilience benchmark systematically evaluates aggregation robustness [25], and fairness benchmarking at the client level has been proposed alongside [26]. These systems and benchmarks largely treat regulatory evidence as out of scope. They characterize statistical behaviour such as accuracy, fairness, and resilience, but they do not evaluate a deployment-specific control profile. FL-Compliance-Guard instead monitors configuration, event, and artifact records. Its fault model includes crashes, dropped records, and silent mechanism degradation, complementing statistical robustness benchmarks.

3.2 Privacy mechanisms and their auditing

Differential privacy provides the formal language in which the GDPR's "data protection by design" and the AI Act's anonymization reasoning are typically operationalized for ML [13, 27, 28]. A parallel line of work *audits* whether deployed mechanisms deliver their promised protection: Jagielski et al. estimate empirical lower bounds on the privacy of differentially private SGD [29]; Andrew et al. give a one-shot empirical privacy estimator for FL [30]; Chang et al. design efficient membership-inference-based auditing for FL [31]. This literature asks how much privacy a training run provided. Our work asks whether recorded mechanism parameters remain consistent with an admitted configuration and an operator-defined budget. Empirical privacy auditing can test the implementation; our framework records and checks deployment evidence. The accountant in Eq. (4) ignores amplification from sampling, trading tightness for a bound that does not depend on the simulator's sampling scheme.

3.3 Verifiable and auditable federated learning

A growing literature makes FL executions verifiable against malicious aggregators: blockchain-based audit frameworks with smart-contract monitoring [32], secret-sharing-based verifiable secure aggregation [33], one-shot verifiable aggregation protocols [34], and enclave-based runtime attestation that produces verifiable claims about executed training [16]. These works primarily establish that aggregation was computed correctly over submitted updates. Our ledger uses hash chaining [11, 12] to detect later changes relative to a trusted chain anchor. It addresses consistency with a declared control profile rather than correctness of aggregation. A round payload could also carry commitments from a verifiable aggregation protocol, but such a protocol is outside the present contribution.

3.4 Regulation and compliance engineering for ML

Legal scholarship has analyzed the fit between FL and data protection law, including how fine-tuning and federation redistribute obligations across model providers and deployers [9], and risk-management frameworks such as the NIST AI RMF [35] and ISO/IEC 42001 [36] define management-level controls. The privacy-by-design tradition [10] argues for building protections into systems rather than relying only on later audits. A useful engineering layer between management frameworks and FL code would make the

chosen mappings from legal provisions to system controls explicit and testable. The FL systems reviewed above do not provide both admission-time and runtime evaluation of such a profile. FL-Compliance-Guard explores that combination; the mapping is intended to be reviewed and revised rather than treated as a fixed legal interpretation.

3.5 Positioning

Table 1 contrasts the scope of the reviewed work with ours along four axes: statistical evaluation, mechanism auditing, execution verifiability, and profile checking. The final axis refers to evaluation of declared technical controls, not a legal-compliance determination.

Table 1: Scope of adjacent work relative to this paper.

Work	Statistical eval.	Mechanism audit	Execution verifiability	Profile checks
FL frameworks [21–23]	✓	–	–	–
Robustness benchmarks [25, 26]	✓	–	–	–
DP auditing [29–31]	–	✓	–	–
Verifiable FL [16, 32–34]	–	–	✓	–
Legal analysis [9]	–	–	–	prose only
This work	–	✓(accounting)	✓(ledger)	✓

4 The FL-Compliance-Guard Framework

4.1 Architecture overview

Figure 1 shows the framework's components and their interaction points with a Flower-style FL server loop. A *compliance profile* encodes the deployment's regulatory regime: which mechanisms are mandatory, which thresholds apply, and which jurisdictions are approved. Before any training begins, the *static config checker* evaluates the declared configuration against the profile and issues an admit/deny decision with a violation report. During training, the *runtime monitor* observes each round's bookkeeping facts, evaluates the runtime assertions, and appends a hash-linked record to the *audit ledger*; the server additionally commits a hash of every served checkpoint artifact. An external *verifier* (an auditor, a data protection officer, or a supervisory script) can then check the ledger against a trusted chain anchor, the admitted profile and configuration, update receipts, and retained artifacts. These inputs support checks of chain integrity, configuration binding, participation accounting, and the recorded privacy budget.

4.2 Compliance profiles

A profile is a declarative document with two types of field: values taken from an applicable rule when that rule specifies a value, and implementation choices supplied by the operator. Listing 1 shows the profile used in the evaluation. The 72-hour breach-notification limit follows GDPR Art. 33. The six-year retention setting is an operator choice that aligns with the HIPAA documentation-retention rule when the ledger is designated as required compliance documentation [8]; it is not a general six-year retention rule for every audit log. AI Act Art. 19 specifies at least six months for covered automatically generated logs, subject to other applicable law [5]. The privacy cap, approved transfer regions, aggregation threat model, and decision to require DP are deployment choices, not values prescribed by those statutes.

```
regime:
  dp_required: true
  dp_epsilon_cap: 30.0 # deployment-specific threshold
  dp_delta: 1.0e-5
```

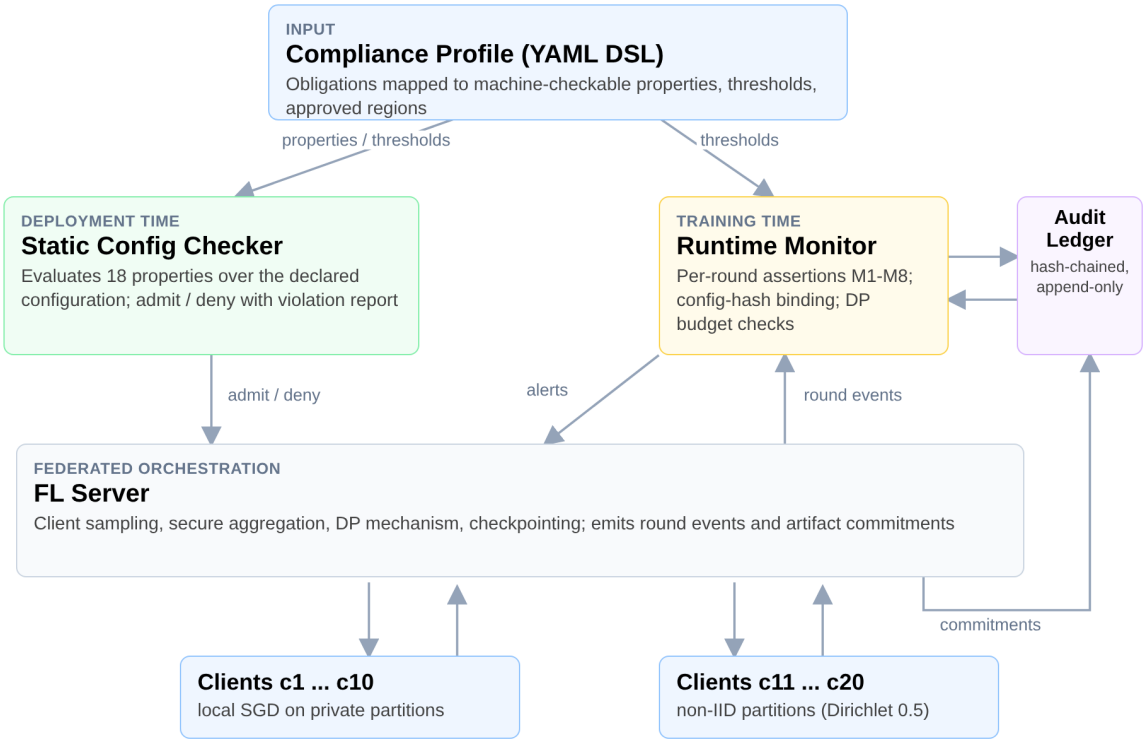


Figure 1: FL-Compliance-Guard architecture. The compliance profile drives two checkpoints (static configuration checking at deployment time; runtime assertions at training time). All evidence flows into a hash-chained audit ledger that an external verifier can inspect without access to raw training data. A trusted chain anchor and the admitted configuration remain external inputs to verification.

```
  min_retention_years: 6 # if ledger is required HIPAA documentation
  max_breach_sla_hours: 72 # GDPR 33
  approved_transfer_regions: [EU-DE, EU-FR, EU-NL] # operator policy
deployment:
  threat_model: honest_but_curious
  aggregation: secagg
  transport: {tls_enabled: true, min_version: "1.3"}
  oversight: {pause_hook: true, review_interval_days: 1} # AI Act 14
```

Listing 1: Compliance profile used in the evaluation (abridged). Comments distinguish explicit regulatory values from operator declarations.

4.3 Property catalog

Table 2 lists the 18 predicates in the current profile. Each predicate operates on the declared configuration (static), the per-round event stream (runtime), or both. The third column identifies the regulatory or policy rationale for the control; it does not assert that the cited provision mandates the implementation shown. The catalog covers security settings (P1, P2, P6), privacy-mechanism configuration and accounting (P3–P5), evidence retention and integrity (P7, P8, P17), process hooks represented by artifacts (P9, P12–P15, P18), and participant controls (P10, P11, P16). The predicates are decidable from the stated artifacts. Most static checks are dictionary comparisons, whereas ledger verification is linear in the number of records inspected. Section 8 discusses controls that cannot be represented this way.

4.4 Static configuration checking

The static checker implements each configuration predicate as a total function over the parsed document. It returns either satisfaction or a violation record $(\pi, \text{path}, \text{observed})$ identifying the predicate, failed path, and observed value. Evaluation is fail-closed: a missing key or type error is reported as a violation. The checker admits or denies the profile as a whole (Eq. (5)); in the prototype, a denial prevents training until the flagged paths are corrected. The measured mean evaluation time is 0.011 ms per configuration (Section 6).

4.5 Runtime monitoring and evidence

The runtime monitor addresses the three practical issues raised in Section 2. For *binding* (issue i), the checker hashes a canonical projection of the admitted runtime-relevant settings. Every round, the monitor derives the same projection from the effective mechanism state, including whether DP was applied, its noise multiplier and clip norm, and the aggregation scheme. A mismatch raises an alert. This detects, for example, a debug setting that disables noise after admission. For *evidence integrity* (issue ii), every round the monitor appends a payload (participants, effective configuration hash, mechanism parameters, timing) to the ledger, chaining each record to its predecessor:

$$h_t = \text{SHA-256}\big(\text{canon}(\text{idx}_t, \text{round}_t, \text{type}_t, \text{payload}_t) \,\|\, h_{t-1}\big), \tag{7}$$

with a genesis hash h_0. Deletion or modification of a historical record breaks the chain from that point onward, provided the verifier retains a trusted chain head or external anchor. Served checkpoint artifacts are committed by hash every five rounds and later compared with the stored artifact. For *attribution* (issue iii), every alert carries the control or monitor identifier that failed, allowing the reviewer to recover its stated rationale from Table 2.

The monitor evaluates eight assertion families (M1–M8 in Table 3): ledger continuity (no round executed without a record), chain integrity, effective-vs-admitted configuration binding, admission/revocation checks, participation reconciliation (the number of update receipts must equal the number the ledger claims), checkpoint artifact integrity, DP parameter recording, and privacy-budget verification through the accountant of Eq. (4). Algorithm 1 summarizes the per-round enforcement loop.

Table 2: Control catalog. The cited provisions motivate the selected engineering evidence but do not necessarily prescribe the listed mechanism.

ID	Property	Regulatory or policy rationale	Check
P1	Transport encryption	GDPR 32(1); HIPAA 164.312(e)(1)	static+rt
P2	Encryption at rest	GDPR 32(1); HIPAA 164.312(a)(2)(iv)	static
P3	DP mechanism active	Deployment policy supporting GDPR 25(1), 32(1)	static+rt
P4	Privacy accountant present	Deployment accountability policy; GDPR 5(2), 25(1)	static+rt
P5	Privacy budget cap	Operator-defined threshold; GDPR 25(1)	static+rt
P6	Secure aggregation	Deployment threat model; GDPR 25(1), 32(1)	static
P7	Ledger integrity mechanism	AI Act 12; HIPAA 164.312(b), (c)(1)	static+rt
P8	Evidence-retention period	AI Act 19; HIPAA 164.316(b)(2)(i), when applicable	static
P9	Breach-notification process	GDPR 33–34	static
P10	International-transfer allowlist	GDPR Ch. V (44–49)	static
P11	Revocation enforcement	Access policy; GDPR 32; HIPAA 164.312(a)	static+rt
P12	Human-oversight intervention hook	AI Act 14(4)(e)	static
P13	Technical-documentation artifact	AI Act 11	static
P14	Robustness-evaluation artifact	AI Act 15(1)–(4)	static
P15	Risk-assessment review interval	Profile policy; GDPR 35(11); AI Act 9	static
P16	Client access control and authentication	GDPR 32; HIPAA 164.312(a), (d)	static+rt
P17	Participation record	AI Act 12; HIPAA 164.312(b)	static+rt
P18	Model-version serving guard	Deployment policy supporting AI Act 15(4)	static

Algorithm 1: Per-round runtime enforcement.

Input: profile Π; admitted config hash h^{adm}; ledger $\mathcal{L}$; per-round receipts $\{r_{t,k}\}$; artifacts
Output: alerts $\mathcal{A}$; ledger records; budget report

$\mathcal{A} \leftarrow \emptyset$; $h_0 \leftarrow$ genesis;
for *round* $t = 1$ **to** T **do**
  $h_t^{\text{eff}} \leftarrow$ hash(effective mechanism parameters at t);
  if $h_t^{\text{eff}} \neq h^{\text{adm}}$ *(M3)* **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\textsc{ConfigBind}, t)\}$;
  end
  for *each participant* k *sampled at* t **do**
    if k *on revocation list (M4)* **then**
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\textsc{Revoked}, t, k)\}$;
    end
  end
  append $\mathcal{L}.(t, \text{participants}, n_t, h_t^{\text{eff}}, h_{t-1})$ with h_t per Eq. (7);
  if *rounds covered by* $\mathcal{L}$ *have gaps (M1)* **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\textsc{Continuity}, t)\}$;
  end
  if $|\{r_{t,k}\}| \neq n_t$ *(M5)* **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\textsc{Reconcile}, t)\}$;
  end
  if *required mechanism parameters are absent (M7)* **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\textsc{Parameters}, t)\}$;
  end
  if *re-hash of* $\mathcal{L}$ *finds a broken link (M2)* **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\textsc{Chain}, t)\}$;
  end
  if $t \bmod 5 = 0$ ***and*** *hash(served artifact)* $\neq$ *committed (M6)* **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\textsc{Integrity}, t)\}$;
  end
  if ε_t *from Eq. (4)* $> \Pi.$*cap (M8)* **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\textsc{Budget}, t)\}$;
  end
end
return $\mathcal{A}$, $\mathcal{L}$, *budget report*;

Table 3: Runtime assertion families (M1–M8) and the failure classes they target.

ID	Assertion (per round)	Failure class
M1	Ledger continuity: every executed round has a record	lost records (crash-recovery bugs)
M2	Rolling full re-hash of the ledger	deletion or rewriting of audit entries
M3	Effective vs. admitted configuration hash	silent mechanism degradation
M4	No participant on the revocation list	policy-violating admission
M5	Update receipts equal ledger participation count	participation fraud
M6	Served artifact hash equals committed hash	storage corruption / tampering
M7	Mechanism parameters recorded in the round payload	unrecorded parameters
M8	Realized ε_t under profile cap (Eq. (4))	budget overrun

4.6 Non-interference by construction

In the reference implementation, the monitor observes bookkeeping state. It consumes no random draws (client sampling, batch shuffling, and noise are drawn from streams independent of monitoring), reads no raw training data, and writes only to the ledger. Monitored and unmonitored runs should therefore be bit-identical for a fixed seed; Section 6 tests this implementation-level property. The result is empirical and does not prove non-interference for other integrations.

4.7 Privacy-budget accounting as an enforced property

The accountant applies Eq. (4) to parameters recorded in each round payload rather than copying the values declared at admission. If a record indicates that noise was absent, the corresponding privacy loss is treated as unbounded; missing parameters separately trigger M7. M3 compares the effective mechanism state with the admitted projection, while M8 compares the accumulated bound with the profile cap. The two checks therefore address different failures: configuration drift and budget overrun.

5 Experimental Setup

The evaluation addresses three questions. **Q1 (Coverage):** does the framework detect seeded violations of its control catalog, and how often does it alert on benign configurations? **Q2 (Runtime):** does the monitor detect operational faults during training, how quickly, and what monitoring cost does it add? **Q3 (Necessity):** which monitor components account for each detection?

5.1 Reference implementation

The prototype comprises approximately 500 lines of Python implementing the property catalog and static checker, the hash-chained ledger (SHA-256 over canonical JSON payloads), the runtime monitor with assertion families M1–M8, the Gaussian RDP accountant of Eq. (4), and a self-contained FL simulator (server loop, client sampling, local SGD, FedAvg/DP-FedAvg aggregation) hosting a two-layer fully connected network (64×128×10; 9,610 parameters) in NumPy. The self-contained simulator removes network nondeterminism from the timing comparison and exposes configuration, round records, and checkpoints analogous to those used by a distributed FL server. The accompanying artifact package is intended to provide the code, configuration, seeds, and raw result files used for the reported values.

5.2 Workload

We use the scikit-learn digits dataset (1,797 grayscale 8×8 images, 10 classes), split into a stratified training pool of 1,437 samples and a test set of 360. The training pool is partitioned across $K = 20$ clients with a Dirichlet(α=0.5) label-skew partition, a commonly used non-IID construction. Training runs for $T = 60$ rounds with $m = 5$ clients sampled per round ($q = 0.25$), $E = 2$ local epochs, batch size 16, and learning rate 0.1. The DP configuration was selected from a small grid over clip norm $C \in \{0.1, 0.25, 0.5, 1.0, 2.0\}$ and noise multiplier $\sigma \in \{0.5, 1.0, 2.0, 4.0\}$, keeping only combinations whose realized budget (Eq. (4), $\delta = 10^{-5}$) satisfies the illustrative profile cap $\varepsilon \leq 30$, and choosing the best final accuracy on the held-out test split. The selected mechanism ($C = 0.1$, $\sigma = 2.0$) yields $\varepsilon = 26.1$ under the no-amplification accountant. All conditions are run with five seeds $\{42, 1, 2, 3, 4\}$; we report mean ± standard deviation. We use the workload to exercise the enforcement path, not as a critical-domain benchmark. Because the DP grid was selected on the reported test split, its accuracy is descriptive and may be optimistically biased; the enforcement results do not depend on that selection.

5.3 Fault-injection protocol

Static faults. We seed 18 misconfigurations (S1–S18), one per catalog property, by mutating a single configuration path of an otherwise profile-satisfying baseline: disabling TLS (S1) or encryption at rest (S2), disabling the DP mechanism (S3) or its accountant (S4), committing $\varepsilon = 100$ against the cap (S5), replacing secure aggregation with plain averaging under an honest-but-curious threat model (S6), disabling

ledger hashing (S7), shortening retention to one year (S8), removing the breach-notification webhook (S9), adding a client from a non-approved jurisdiction (S10), disabling revocation enforcement (S11), removing the oversight pause hook (S12), disabling model-card generation (S13), deleting the robustness report (S14), aging the risk assessment beyond its currency window (S15), disabling client authentication (S16), disabling participation recording (S17), and disabling the stale-model guard (S18). A fault is *detected* if the checker raises a violation on exactly the property the fault targets.

Runtime faults. We inject six operational faults (R1–R6) into monitored training runs, chosen to span the failure classes of Table 3: a logging failure after a completed round, causing its ledger record to be dropped (R1, round 12); deletion of three ledger entries post hoc (R2, rounds 20–22); silent disabling of DP noise from round 30 onward while the admitted configuration claims DP (R3); a client revoked at round 15 that keeps participating and is illicitly re-admitted at round 25 (R4); a ledger record claiming six participants at round 18 where five update receipts exist (R5); and corruption of the served checkpoint artifact between rounds 35 and 40 (R6). A fault is detected if its targeted assertion family fires, and we report the first detection round.

Fuzzing. To estimate the false-positive rate of static checking, we generate 500 perturbed *benign* configurations by randomizing profile-irrelevant hyperparameters (learning rate, batch size, local epochs, client count, rounds, hidden width, aggregation weighting, seeds) and, in 30% of cases, DP parameters. In those cases, the committed ε is recomputed from Eq. (4) so that the configuration remains under the cap, and in 20% of cases client jurisdictions drawn from the approved list. Every fuzzed configuration must pass all 18 properties; any violation is a false positive.

5.4 Metrics and protocol

We report: detection counts and first-detection rounds for Q1/Q2; alert counts and per-configuration check latency for the static checker; final test accuracy (mean ± s.d. over five seeds), per-seed training time, monitor time, per-round monitoring cost, and ledger size for Q2; bit-exactness of accuracy trajectories between monitored and unmonitored runs as the non-interference check; and detection counts over six degraded ablation variants (static-only, runtime-only, and four single-component ablations) plus three enforcement granularities for Q3. All experiments execute on one CPU core in the containerized evaluation environment. The suite, including the DP grid, completes in under three minutes in that environment.

6 Results

6.1 Static checking: coverage, precision, and cost (Q1)

Table 4 reports the 18 seeded static misconfigurations. Each fault raises its targeted predicate and no other predicate: disabling TLS raises P1, for example, while committing ε=100 raises P5. The violation identifier links the finding to the rationale in Table 2. The mean check time is 0.011 ms per configuration in the evaluation environment.

None of the 500 benign configurations raised an alert. Mean and 95th-percentile check latencies were 0.011 ms and 0.015 ms, respectively. The generator varies profile-irrelevant hyperparameters freely. When it varies DP parameters, it also recomputes the committed budget so that the configuration remains below the profile cap. The result measures behaviour only over this generated test set; it does not establish a zero underlying false-alert rate.

6.2 Runtime fault detection (Q2)

Table 5 reports the six injected operational faults. Each fault triggers its targeted assertion family at the next applicable check. M1 first reports the missing round-12 record at round 13 and repeats the alert while the gap remains. R2 is different: the entries are deleted after training, so the post-run chain verification identifies record 20 as the first affected entry. When DP noise is disabled from round 30 (R3), the effective

mechanism projection differs from the admitted projection in that round and in the 30 rounds that follow. M4 reports the revoked client when it is next sampled at round 17 and on later prohibited participation. M5 reports the participation-count mismatch in round 18, and M6 reports checkpoint corruption at the scheduled round-40 artifact check.

Across the clean monitored runs (identical configurations and no injected faults), the monitor raises no alerts in 60 rounds. This observation is specific to the evaluated configurations and assertion families.

6.3 Utility, monitoring cost, and non-interference (Q2)

Table 6 summarizes utility and cost across the four training conditions, and Figure 2 plots the accuracy trajectories. Adding the monitor leaves accuracy bit-identical at all twelve reported checkpoints for every seed, which supports the implementation-level non-interference claim in Section 4. The monitor consumes 0.33 ms per round without DP and 1.21 ms per round with DP. Expressed as a ratio of monitor time to measured training time, these costs are 4.4% and 15.2%, respectively. The DP path additionally records mechanism parameters and queries the accountant. Because the simulator runs in less than one second, these ratios are sensitive to small timing variations and should not be read as production overhead estimates. The 60-round ledger occupies about 27 KB, or roughly 450 bytes per round, before any storage-system replication or external anchoring.

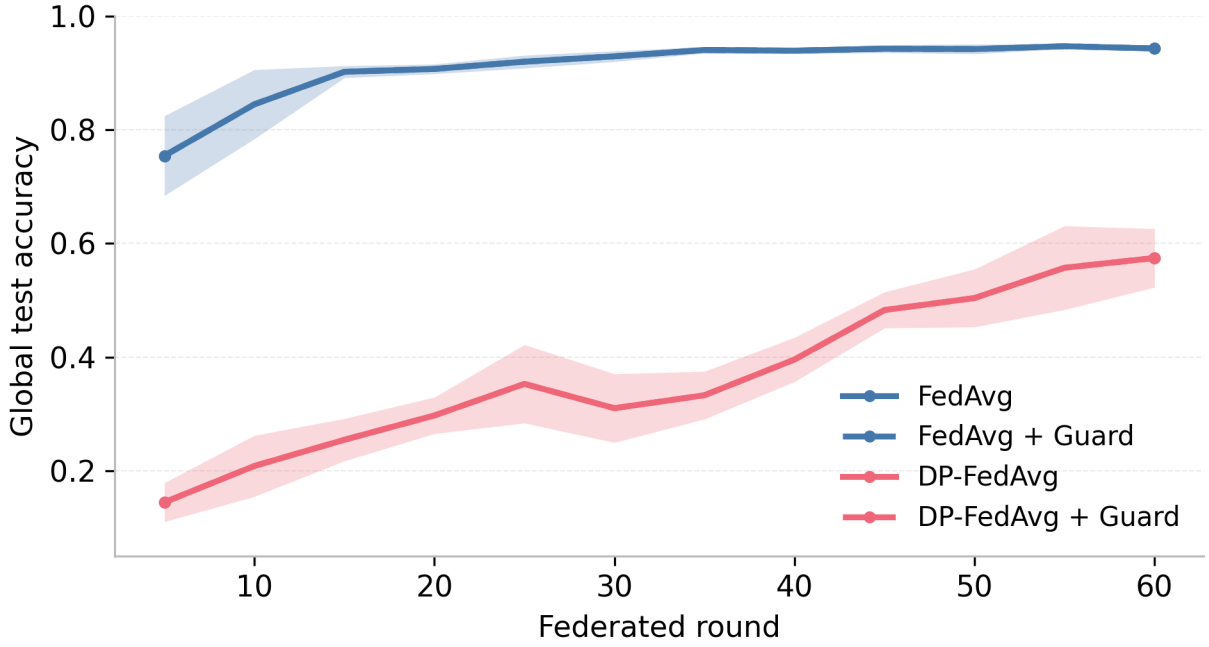


Figure 2: Global test accuracy over federated rounds (mean over five seeds; shaded bands: ± one standard deviation). Monitored and unmonitored curves coincide at the recorded precision, so the Guard curves are hidden under their counterparts.

6.4 Privacy-budget accounting in action

Table 7 illustrates the accounting layer. The selected mechanism ($C = 0.1$, $\sigma = 2.0$, $q = 0.25$, $T = 60$) realizes $\varepsilon = 26.1$ at $\delta = 10^{-5}$ under Eq. (4), below the profile's illustrative cap of 30. The participation rate q describes the workload but does not enter this no-amplification bound. If only the noise multiplier is changed to $\sigma = 0.1$, the bound becomes approximately 3,372, more than 100 times the profile cap. The static checker rejects that configuration through P5. If introduced during a run, the one-round bound already exceeds 30, so M8 alerts at the first affected round. Checking only whether a DP flag is enabled would miss this parameter-level failure.

Under the selected configuration, $\varepsilon = 26.1$ is an accounting result, not a claim that 30 is an appropriate privacy threshold for other deployments. At this proof-of-concept scale, DP-FedAvg reaches 0.574 ± 0.052

Table 4: Static fault injection: 18 seeded misconfigurations, all detected by their target property. Mean check time per configuration: 0.011 ms.

ID	Seeded misconfiguration	Property	Detected
S1	TLS transport disabled (plaintext channels)	P1	✓
S2	Encryption at rest disabled	P2	✓
S3	DP mechanism disabled	P3	✓
S4	DP enabled but accountant removed	P4	✓
S5	Committed $\varepsilon = 100$ exceeds cap 30	P5	✓
S6	Secure aggregation replaced by plain averaging	P6	✓
S7	Audit-log hash chaining disabled	P7	✓
S8	Log retention reduced to 1 year (requires 6)	P8	✓
S9	Breach-notification webhook missing	P9	✓
S10	Client outside the approved transfer-region list admitted	P10	✓
S11	Revocation-list enforcement disabled	P11	✓
S12	Human oversight pause hook removed	P12	✓
S13	Automatic model-card generation disabled	P13	✓
S14	Robustness evaluation report missing	P14	✓
S15	Risk assessment (DPIA) stale (400 days old)	P15	✓
S16	Client authentication disabled	P16	✓
S17	Per-round participation recording disabled	P17	✓
S18	Stale-model serving guard disabled	P18	✓

Table 5: Runtime fault-injection outcomes. All six faults are detected at the next applicable check; R2 is injected and detected during the post-run audit. The clean monitored runs produce no alerts.

ID	Injected fault	Primary check	First detection	Primary alerts
R1	Completed round has no ledger record	M1	round 13	48
R2	Ledger entries for rounds 20–22 deleted	M2	round 20	1
R3	DP noise silently disabled from round 30	M3	round 30	31
R4	Revoked client participates; re-admitted	M4	round 17	3
R5	Ledger claims 6 participants, 5 receipts	M5	round 18	1
R6	Served checkpoint artifact corrupted	M6	round 40	5

Table 6: Utility and cost over five seeds (mean ± s.d.; timing columns show means per run). The ratio is monitor time divided by measured training time, not end-to-end wall-clock slowdown.

Condition	Final acc.	t_{train} (s)	t_{mon} (s)	Monitor/train	Monitor (ms/round)
FedAvg	0.943 ± 0.002	0.448	n/a	n/a	n/a
FedAvg + Guard	0.943 ± 0.002	0.449	0.020	4.4%	0.33
DP-FedAvg	0.574 ± 0.052	0.457	n/a	n/a	n/a
DP-FedAvg + Guard	0.574 ± 0.052	0.478	0.073	15.2%	1.21

Table 7: Privacy-budget accounting under the illustrative profile cap $\varepsilon \leq 30$ ($\delta = 10^{-5}$, $T = 60$). The conservative accountant does not use q.

Mechanism	C	σ	ε	Verdict
DP-FedAvg (selected)	0.1	2.0	26.1	pass (M8 silent)
DP-FedAvg (degenerate)	0.1	0.1	≈ 3,372	reject: P5 static; M8 alerts from round 1

against 0.943 ± 0.002 without DP (Figure 2). Aggressive clipping and per-round noise reduce utility in this small model and dataset. The identical monitored and unmonitored trajectories attribute that gap to the selected training mechanism rather than to the monitor. Larger deployments require a separate privacy–utility study and a justified profile threshold.

6.5 Ablation study (Q3)

Figure 3 reports detection coverage over all 24 faults (18 static, 6 runtime) for the full system and six degraded variants. The full system detects 24/24. Static-only checking (A1) detects the 18 configuration violations but none of the six runtime faults. Runtime-only checking (A2) detects all six runtime faults but does not perform the admission tests. The four single-component ablations each lose one fault: removing the rolling chain audit (A3) hides post-hoc ledger deletion (R2); removing config-hash binding (A4) hides silent mechanism degradation (R3); removing participation reconciliation (A5) hides the count fraud (R5); and removing artifact integrity checking (A6) hides checkpoint corruption (R6). Within this fault suite, M2, M3, M5, and M6 cover distinct failures; the experiment does not measure redundancy for faults outside the catalog.

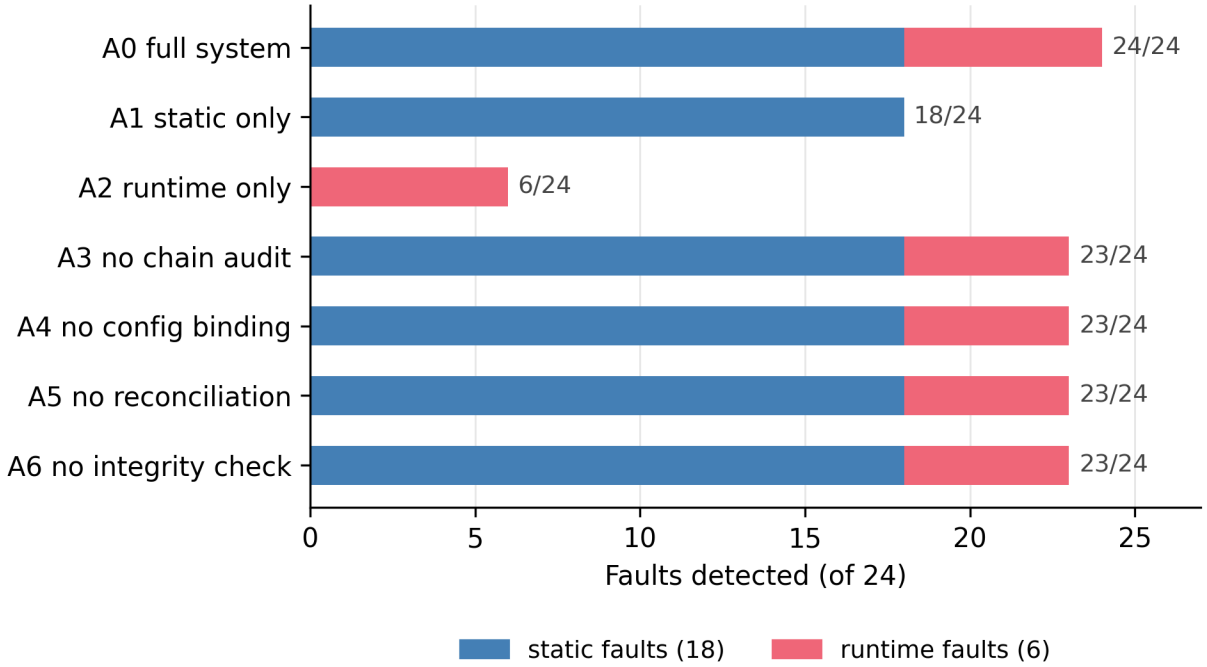


Figure 3: Ablation study: faults detected (of 24) when monitor components are disabled. The full system (A0) detects all 24; each single-component ablation (A3–A6) loses exactly one runtime fault; single-checkpoint variants (A1, A2) lose an entire fault class.

6.6 Enforcement granularity

Figure 4 compares three checking schedules in the DP condition. Ledger-only recording takes time equal to 1.0% of training time, checking the accumulated records every fourth round takes 3.3%, and checking every round takes 15.6% in this seed subset. Because events remain in the ledger, the sampled schedule retains eventual coverage in this fault suite but can delay an alert by up to the checkpoint interval. Ledger-only mode defers all alerts until the post-run audit. The profile can select a schedule according to the required detection latency.

6.7 Threats to validity

Construct validity. Detection is measured against the fault catalog the framework was designed to catch; the catalog was fixed before running the injections, but both derive from the same property analysis, so

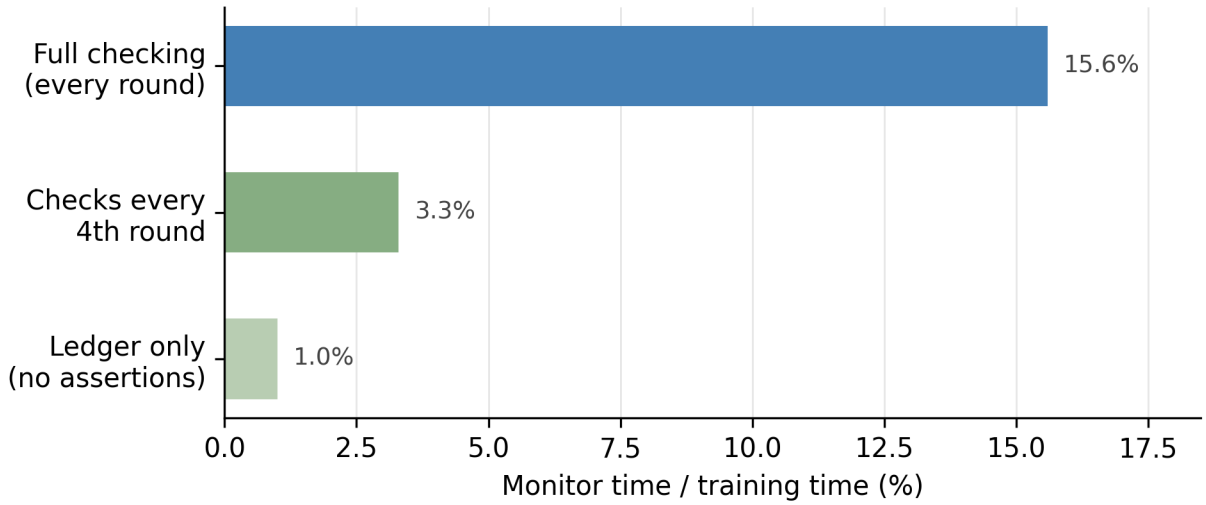


Figure 4: Monitor time divided by training time in the DP condition, by checking schedule.

coverage of *unanticipated* misconfigurations is not established here. The 500 benign perturbations probe false alerts only within the generator's configuration space. In addition, selection of the DP configuration on the reported test split can bias its accuracy upward; a separate validation split is needed for an unbiased model comparison.

Internal validity. Timing measurements on a shared, containerized CPU carry noise; we report means over five seeds for accuracy and multiple timing repetitions, and the qualitative conclusions (bit-exactness, detection completeness) are deterministic. All injected faults and their ground truth are encoded in the released scripts.

External validity. The workload is digits-scale with 20 simulated clients and a 9,610-parameter model, and the simulator abstracts communication entirely; absolute monitoring-to-training ratios will differ in a networked deployment (where monitor cost amortizes against much larger per-round wall-clock), and the accuracy cost of DP at this scale does not transfer to production regimes. The property catalog contains 18 controls motivated by a small subset of the provisions in the three regulatory regimes. These limits are discussed further in Section 8.

7 Discussion

The evaluation separates failures visible at admission from those that appear during execution. Static checking rejects a configuration that violates the selected profile, whereas runtime checking reports later divergence. In the stated fault suite, neither checkpoint alone covers both classes. The single-component ablations also associate ledger verification, configuration binding, participation reconciliation, and artifact integrity with different injected failures.

The framework is compliance-evidence infrastructure, not an automated legal decision-maker. A pass means only that the observed artifacts satisfy the selected predicates. It does not show that every applicable legal duty has been identified or fulfilled. Operators, legal specialists, data protection officers, and other accountable reviewers must determine applicability and interpret context-dependent requirements. The catalog exposes the boundary by linking each predicate to its stated rationale and leaving non-observable requirements outside the profile verdict.

The checking schedule presents a deployment trade-off. Per-round verification minimizes detection latency, sampled verification reduces monitoring time, and ledger-only operation defers detection until audit. A profile can specify the schedule appropriate to the application's risk and operational constraints.

The subsecond simulator cannot predict end-to-end cost in a networked system; that question requires production-scale measurement.

8 Limitations

The evaluation uses a small, single-process simulator and a digits classification workload. It demonstrates end-to-end enforcement behavior but does not reproduce the scale, communication failures, infrastructure diversity, or governance processes of a critical-domain deployment. The fault suite is derived from the current property catalog, so the reported coverage does not imply detection of unmodeled failures. Likewise, zero false positives over 500 benign perturbations is evidence for the tested configuration space, not a universal precision guarantee.

The trust model excludes a server that can forge the complete execution history and all external commitments. Hash chaining makes deletion and modification evident relative to a trusted chain head, but stronger adversaries require external anchoring, remote attestation, trusted execution, or a verifiable aggregation protocol. The privacy accountant enforces a conservative analytical bound from recorded parameters without claiming amplification from client sampling. It does not empirically audit the implemented mechanism or prove that sensitive information cannot leak through other pipeline components.

Finally, the catalog contains engineering controls motivated by a limited selection of provisions from three regulatory regimes. Legal applicability and interpretation depend on jurisdiction, role allocation, deployment context, and later guidance. The profile and its mappings require periodic expert review and should not be treated as legal advice.

9 Conclusion

FL-Compliance-Guard links a declared control profile to observable artifacts from an FL pipeline. In the proof-of-concept evaluation, its static checker, runtime monitor, hash-chained ledger, configuration binding, and privacy accountant detect the 18 seeded configuration faults and six runtime faults without changing the recorded accuracy trajectories. The result is evidence about the selected profile and fault model, not a general compliance determination. Future work should evaluate distributed deployments, broaden and independently review the control catalog, and integrate external anchoring, execution attestation, and verifiable aggregation.

Declarations

Acknowledgments

The author gratefully acknowledges the developers and maintainers of NumPy and scikit-learn, whose open-source software supported the reference implementation and evaluation. The author also acknowledges the contributors who made the scikit-learn digits dataset publicly available for reproducible research, and the institutions that maintain public access to the official GDPR, EU AI Act, and HIPAA materials used to construct the obligation-to-property mappings.

Conflict of interest

The author declares no competing interests.

Author contributions

Ahsan Umar: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Visualization, Writing—original draft, and Writing—review and editing. The author read and approved the final manuscript.

References

[1] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282, 2017.

[2] Peter Kairouz, H. Brendan McMahan, Brendan Avent, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021. doi: 10.1561/2200000083.

[3] Nicola Rieke, Jonny Hancox, Wenqi Li, et al. The future of digital health with federated learning. *npj Digital Medicine*, 3:119, 2020. doi: 10.1038/s41746-020-00323-1.

[4] Georgios Kaissis, Alexander Ziller, Jonathan Passerat-Palmbach, Théo Ryffel, Dmitrii Usynin, Andrew Trask, Ionésio Lima, Jason Mancuso, Friederike Jungmann, Marc-Matthias Steinborn, Andreas Saleh, Marcus Makowski, Daniel Rückert, and Rickmer Braren. End-to-end privacy preserving deep learning on multi-institutional medical imaging. *Nature Machine Intelligence*, 3(6): 473–484, 2021. doi: 10.1038/s42256-021-00337-8.

[5] European Parliament and Council of the European Union. Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence. Official Journal of the European Union, L 2024/1689, 2024. URL https://eur-lex.europa.eu/eli/reg/2024/1689/oj.

[6] European Parliament and Council of the European Union. Regulation (EU) 2026/1744 amending regulations (EU) 2024/1689, (EU) 2018/1139 and (EU) 2023/1230 as regards the simplification of the implementation of harmonised rules on artificial intelligence. Official Journal of the European Union, L 2026/1744, 2026. URL https://eur-lex.europa.eu/eli/reg/2026/1744/oj.

[7] European Parliament and Council of the European Union. Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data. Official Journal of the European Union, L 119, 2016. URL https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[8] U.S. Department of Health and Human Services. Modifications to the HIPAA privacy, security, enforcement, and breach notification rules. Federal Register, 78 FR 5566, 2013. URL https://www.federalregister.gov/d/2013-01073.

[9] Philipp Hacker and Matthias Holweg. The regulation of fine-tuning: Federated compliance for modified general-purpose AI models. *Computer Law & Security Review*, 60:106234, 2026. doi: 10.1016/j.clsr.2025.106234.

[10] Ann Cavoukian. Privacy by design: The 7 foundational principles. Technical report, Information and Privacy Commissioner of Ontario, 2009.

[11] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991. doi: 10.1007/BF00196791.

[12] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology—CRYPTO '89 Proceedings*, pages 218–238, 1990. doi: 10.1007/0-387-34805-0_21.

[13] Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016. doi: 10.1145/2976749.2978318.

[14] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium*, pages 263–275, 2017. doi: 10.1109/CSF.2017.11.

[15] Ilya Mironov, Kunal Talwar, and Li Zhang. Rényi differential privacy of the sampled gaussian mechanism. arXiv:1908.10530, 2019.

[16] Jinnan Guo, Kapil Vaswani, Andrew Paverd, and Peter Pietzuch. VerifiableFL: Providing verifiable claims to federated learning using exclaves. arXiv:2412.10537, 2024.

[17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, et al. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017. doi: 10.1145/3133956.3133982.

[18] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2938–2948, 2020.

[19] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[20] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5650–5659, 2018.

[21] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing To Li, Titouan Parcollet, Pedro P. B. de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020. doi: 10.48550/arXiv.2007.14390.

[22] Chaoyang He, Songze Li, Jinhyun So, et al. FedML: A research library and benchmark for federated machine learning. arXiv:2007.13518, 2020.

[23] Fan Lai, Yinwei Dai, Sanjay S. Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. FedScale: Benchmarking model and system performance of federated learning at scale. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 11814–11827, 2022.

[24] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, et al. Towards federated learning at scale: System design. In *Proceedings of Machine Learning and Systems*, volume 1, pages 374–388, 2019.

[25] Laura Tran-Dubois. The FLResilience benchmark: A systematic framework for evaluating federated learning aggregation strategies. OpenReview preprint, 2026. URL https://openreview.net/forum?id=rxdJjgYtNw.

[26] Xenia Heilmann, Luca Corbucci, and Mattia Cerrato. A benchmark for client-level fairness in federated learning. In *Proceedings of the Fourth European Workshop on Algorithmic Fairness*, volume 294 of *Proceedings of Machine Learning Research*, pages 434–438, 2025.

[27] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. *Journal of Privacy and Confidentiality*, 7(3):17–51, 2016. doi: 10.29012/jpc.v7i3.405. Conference version appeared at TCC 2006.

[28] Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*, volume 9 of *Foundations and Trends in Theoretical Computer Science*. Now Publishers, 2014. doi: 10.1561/0400000042.

[29] Matthew Jagielski, Jonathan Ullman, and Alina Oprea. Auditing differentially private machine learning: How private is private SGD? In *Advances in Neural Information Processing Systems*, volume 33, pages 22205–22216, 2020.

[30] Galen Andrew, Peter Kairouz, Sewoong Oh, Alina Oprea, H. Brendan McMahan, and Vinith M. Suriyakumar. One-shot empirical privacy estimation for federated learning. In *International Conference on Learning Representations*, 2024.

[31] Hongyan Chang, Brandon Edwards, Anindya S. Paul, and Reza Shokri. Efficient privacy auditing in federated learning. In *33rd USENIX Security Symposium*, pages 307–323, 2024.

[32] Aditya Pribadi Kalapaaking, Ibrahim Khalil, Xun Yi, Kwok-Yan Lam, Guang-Bin Huang, and Ning Wang. Auditable and verifiable federated learning based on blockchain-enabled decentralization. *IEEE Transactions on Neural Networks and Learning Systems*, 36(1):102–115, 2025. doi: 10.1109/TNNLS.2024.3407670.

[33] Sufang Zhou, Lin Wang, Lin Chen, et al. Group verifiable secure aggregate federated learning based on secret sharing. *Scientific Reports*, 15:9712, 2025. doi: 10.1038/s41598-025-94478-0.

[34] Yin Zhu, Junqing Gong, Kai Zhang, Shay Gueron, and Haifeng Qian. FOVA: Fast one-shot verifiable aggregation for federated learning. Cryptology ePrint Archive, Paper 2026/643, 2026. URL https://eprint.iacr.org/2026/643.

[35] National Institute of Standards and Technology. Artificial intelligence risk management framework (AI RMF 1.0). Technical Report NIST AI 100-1, U.S. Department of Commerce, 2023.

[36] International Organization for Standardization. ISO/IEC 42001:2023: Information technology—artificial intelligence—management system. International Standard, 2023.