



ARTICLE

Federated Learning Blockchain (FLBC): Bidirectionally Accountable Federated Learning via On-Chain Pre-Commitment

Neil Kwabena Afum Ohene¹, Frederick Edem Junior Broni², Sultan Noman Qasem^{3,*},
Tawfik Al-Hadhrani⁴, Kwame Assa-Agyei⁴ and Abdullah Albarrak³

¹Engineering Department, Bronsoft Technology, Tema, Ghana

²Computer Science and Engineering Department, University of Mines and Technology, Tarkwa, Ghana

³Computer Science Department, College of Computer and Information Sciences, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, Saudi Arabia

⁴Department of Computer Science, Nottingham Trent University, Nottingham, UK

*Corresponding Author: Sultan Noman Qasem. Email: snmohammed@imamu.edu.sa

Received: 02 June 2026; Accepted: 20 August 2026; Published: 21 October 2026

ABSTRACT: Federated learning (FL) allows institutions to train shared models without exchanging raw data, but standard deployments provide little auditable evidence of which contributions were submitted, received, and included. This is a practical problem in cross-silo settings, where accountability and technical-documentation obligations extend beyond data locality. This paper presents FLBC, a permissioned-ledger protocol for bidirectional accountability in FL. FLBC records client pre-commit hashes, coordinator acknowledgments, round receipts, and coordinator-announced phase deadlines recorded on chain, and applies a pure-function verifier that judges record timeliness by ledger confirmation times and classifies detectable inclusion failures into four evidence categories: silent drop, fake error, partial exclusion, and bilateral mismatch. A reference implementation on Flower and MultiChain was evaluated on EMNIST under a Dirichlet- $\alpha = 0.5$ label-skew partition in an emulated five-institution cross-silo consortium, deployed on a single Linux host via Docker Compose with logically separate MultiChain nodes, containers, and per-hospital wallets, and additionally under emulated wide-area latency, across two physical hosts of differing processor architecture, and on a second dataset (CIFAR-10). In both shared-RPC simulation and per-hospital-wallet enforce-mode deployment, FLBC surfaced the expected omission evidence for every scripted fault, clean runs produced no false violations, and verdict parity held across trust modes, under injected WAN latency, and across the two-host deployment. The verifier detects inconsistencies among on-chain records rather than adjudicating intent: it does not decide whether a stated rejection reason is truthful, and it cannot by itself separate a withheld update from one a client never sent or the network dropped, so its output is evidence for dispute resolution rather than a verdict of malice. FLBC adds measurable wall-clock overhead relative to vanilla Flower, but in return produces a regulator-readable audit artifact and source-pinned run provenance. The reported deployments remain controlled lab environments for reproducibility and do not reproduce asymmetric link failures, clock drift, or operationally independent datacenters. The protocol is complementary to privacy-preserving and Byzantine-robust aggregation methods and is intended to support, rather than determine, compliance workflows.

KEYWORDS: Federated learning; permissioned blockchain; non-repudiation; auditability; regulatory compliance; cross-silo; tamper-evident audit; MultiChain; EMNIST

1 Introduction

Federated learning (FL) allows multiple institutions to train a shared model without exchanging raw data, which makes it appealing in domains where data cannot leave the institutional boundary [1]. In the common *Federated Averaging* (FedAvg) setting, clients compute local updates and a central coordinator aggregates them round by round [1,2]. This architecture is particularly relevant in cross-silo deployments, such as the multi-hospital collaborations surveyed in [3], where legal and operational requirements constrain both data movement and technical documentation [4–6].

That privacy advantage does not, by itself, solve the provenance problem. In regulated settings, institutions and auditors may need to establish which participant submitted a model update, whether that update was received, whether it was included in the aggregate, and, if it was excluded, on what basis. The EU AI Act and GDPR frame these concerns in terms of transparency, documentation, and accountability [4,5]. In FL, they become questions about contribution-level evidence.

Standard FL protocols offer little visibility into the coordinator’s behavior once a round begins. A coordinator can omit an update, record an unjustified rejection, or alter the record of what it received, while participants outside the coordinator have no independent way to verify the event from the protocol surface alone. The problem is not one-sided. A client may pre-commit to one update and transmit another, leaving an honest coordinator with limited means to prove what happened. An auditable cross-silo protocol therefore needs bidirectional non-repudiation, so that both coordinator and clients are bound to records that other parties can verify independently.

Production cross-silo federated learning motivates a geographically distributed consortium, whereas the empirical study emulates membership with five logical hospitals on one documented evaluation host. Separate Docker containers, MultiChain node processes, and wallets preserve institutional identities for reproducibility, but CPU, storage, and the kernel scheduler remain shared, so resource usage across containers is correlated.

Prior work on FL and blockchain has addressed adjacent problems rather than this one directly. BAFFLE removes the coordinator through smart-contract aggregation [7]. BlockFLow studies contribution attribution under collusion [8]. Biscotti focuses on peer-to-peer learning, poisoning resistance, and privacy [9]. VeriFL verifies the arithmetic correctness of aggregation through cryptographic commitments [10]. Two systems come closer to the auditing question. In VFChain, a committee selected through the blockchain aggregates models and records verifiable proofs, and an authenticated data structure keeps those proofs searchable while the committee rotates [11]. Kalapaaking et al. remove the central authority altogether: each participant runs a smart-contract monitoring system, blockchain nodes validate one another’s local models, and the aggregation steps are recorded in a contract before the global model is multisigned [12]. Both establish that a training procedure ran as claimed. Neither produces a per-client record of what a designated coordinator chose to include. Surveys of blockchain-based FL organize this design space by ledger role, aggregation topology, and threat focus [13–15], and per-client accountability for a *designated* coordinator’s inclusion decisions does not appear as a solved problem in those taxonomies.

This paper presents FLBC (Federated Learning Blockchain), a phase-windowed protocol built on a permissioned MultiChain ledger for accountable FL. Clients publish SHA-256 pre-commit hashes of their local updates, the coordinator publishes signed acknowledgments over the update tensors Flower delivers to its aggregation step, and a RoundStart record announces explicit per-phase deadlines that are recorded on chain, while record timeliness is later judged against ledger confirmation times. After aggregation, a CoordinatorReceipt records the contributors, received update hashes, aggregate model hash, and any explicitly rejected clients. A verifier compares these

records against the round deadlines and classifies every detectable inclusion failure into four evidence categories: `silent_drop`, `fake_error`, `partial_exclusion`, and `bilateral_mismatch`. Because the classifier is a pure function over chain state, independent readers of the ledger can derive the same result from the same record.

The paper contributes a phase-windowed protocol for bidirectional non-repudiation in cross-silo FL; a four-evidence violation taxonomy with a pure-function classifier under a documented threat model; an implementation that emits a regulator-readable JSON audit with source-state pinning; and an evaluation on EMNIST under a Dirichlet- $\alpha = 0.5$ label-skew partition, under both shared-RPC simulation and per-hospital-wallet enforcement, extended in revision with emulated wide-area latency, a two-host deployment across heterogeneous architectures, a ledger-cost decomposition, a timing-parameter sensitivity sweep, and a second dataset (CIFAR-10).

The remainder of the paper is organized as follows. [Section 2](#) formalizes the threat model, the FLBC round and stream design (Algorithms 1–4), the Python reference implementation, and the EMNIST evaluation harness. [Figs. 1 and 2](#) summarize phased coordination with verifier timing and the evaluation deployment topology. [Section 3](#) reports omission detection, simulation-vs-identity parity, and training-time overhead from a frozen artifact bundle, together with randomized attack placement, ledger cost metrics, emulated wide-area latency, parameter sensitivity, and a second-dataset accuracy check. [Section 4](#) contrasts FLBC with adjacent federated learning and blockchain systems and restates evaluation limitations. [Section 5](#) recaps the main takeaways. [Appendices A and B](#) collect reproduction commands and default phase parameters.

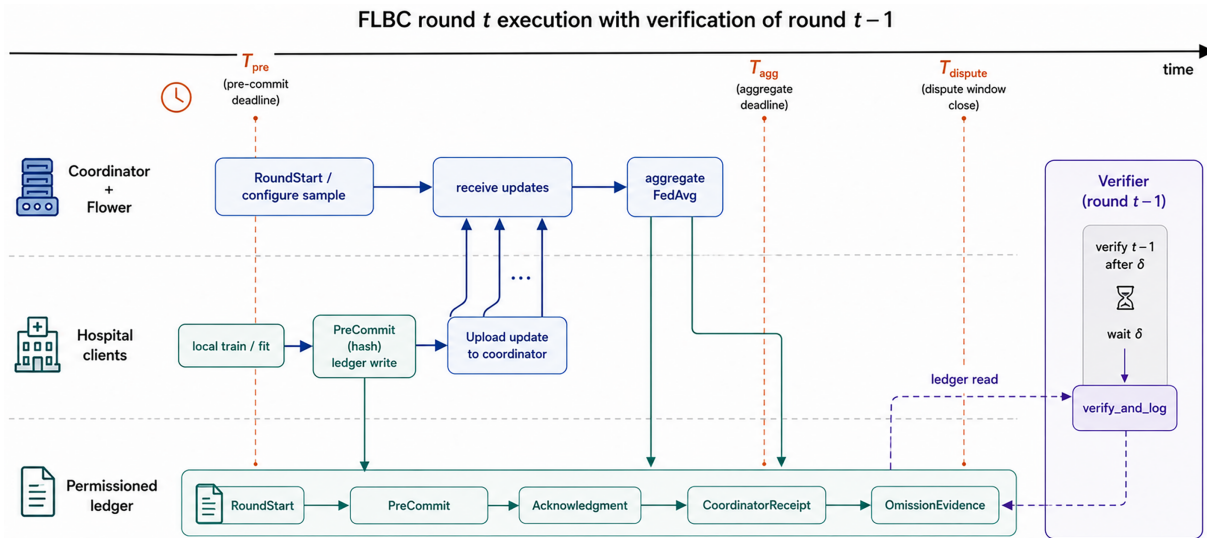


Figure 1: FLBC round- t timeline. Coordinator, clients, and ledger; round $t-1$ is verified after delay δ .

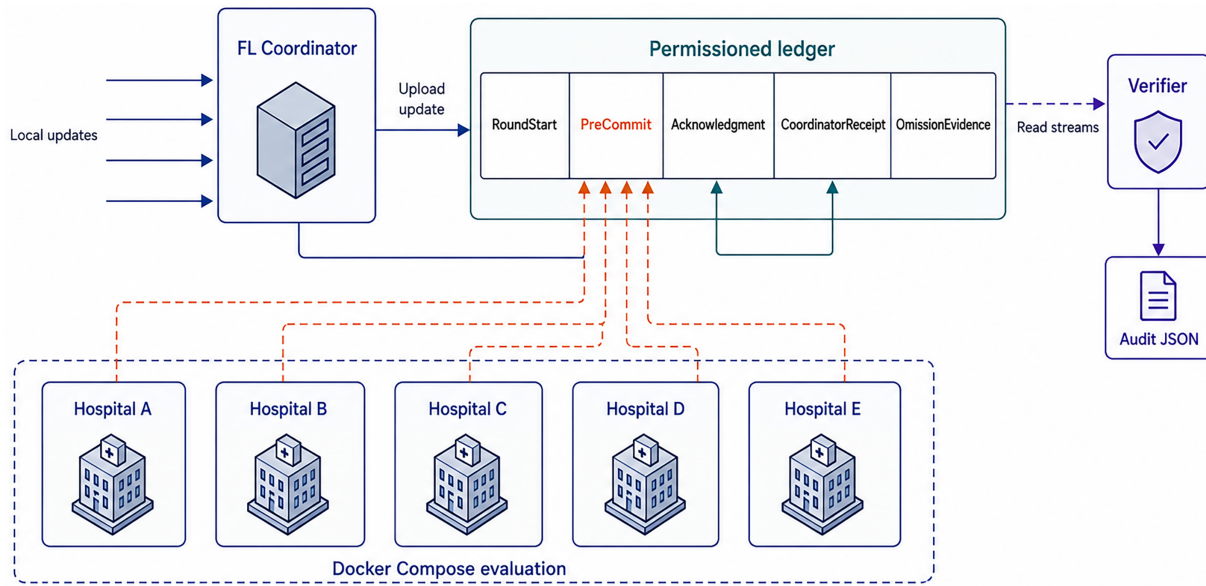


Figure 2: FLBC evaluation architecture. Five hospitals, coordinator, ledger, and verifier on a single-host Docker Compose stack.

2 Materials and Methods

2.1 Threat Model

The setting is a permissioned consortium of N institutions training a shared model under federated learning. One institution acts as the coordinator for a given round and the remaining institutions act as clients. Each institution operates a MultiChain node and can read the protocol streams written to the ledger [16].

The threat model distinguishes two adversarial roles. The first is a corrupted coordinator. This party may receive a client update and omit it from the final record, assert an unjustified rejection reason, or publish a received-hash record that differs from the value it acknowledged earlier. The second is a corrupted client. This party may publish a pre-commit for one update and then transmit different bytes to the coordinator. In both cases, the adversary is assumed unable to publish records on behalf of an honest participant because publisher identity is tied to wallet ownership and enforced against the consortium registry.

The protocol depends on four assumptions. First, the underlying permissioned chain satisfies an honest-majority consensus assumption. Second, each institution operates its own node rather than writing through a single shared operator. Third, the coordinator's wallet is known and registered at consortium setup. Fourth, private keys remain secret. FLBC does not attempt to address poisoning, label manipulation, model backdoors, or other update-content attacks. Its inclusion guarantees apply to updates for which a client published a timely, wallet-authenticated pre-commit; it does not audit participant selection or invitation, and so cannot determine whether the coordinator fairly chose or invited which clients to sample in a given round. It is designed to record what was submitted and what was included, not whether a submitted update was beneficial.

2.2 Protocol Design

FLBC uses a phase-windowed round structure on a permissioned MultiChain ledger. Each round begins with a `RoundStart` record that identifies the coordinator, lists the expected participants, and defines three absolute deadlines: `precommit_deadline`, `aggregate_deadline`, and `dispute_window_close`. These deadlines are compared against the ledger's own confirmation timestamps, so the verifier does not depend on synchronized participant clocks.

The protocol writes to eight streams. [Table 1](#) summarizes their roles.

Table 1: FLBC streams and their roles.

Stream	Record Type	Purpose
<code>client_registry</code>	enrollment record	Consortium membership and <code>node_id</code> -to-wallet binding
<code>fl_rounds</code>	<code>RoundStart</code>	Round announcement and phase deadlines
<code>fl_precommits</code>	<code>PreCommit</code>	Client pre-commit hash before transmission
<code>fl_acknowledgments</code>	<code>Acknowledgment</code>	Coordinator receipt of bytes received
<code>fl_verification</code>	<code>CoordinatorReceipt</code>	Contributors, received hashes, aggregate hash
<code>fl_disputes</code>	<code>Dispute</code>	Client claims of missing or mismatched acks
<code>fl_omissions</code>	<code>OmissionEvidence</code>	Verifier-published violation findings
<code>model_registry</code>	aggregate hash record	Per-round aggregate model identity

Parameter commitment hash

Let $\theta_c^{(r)} = (\mathbf{W}_1, \dots, \mathbf{W}_L)$ denote hospital c 's local model parameters at round r , represented as an ordered list of NumPy arrays in Flower's wire format. FLBC defines a map H into $\{0,1\}^{256}$ (written in hexadecimal in the ledger) by SHA-256 over a canonical byte sequence: each layer j is prepended with its index, cast to IEEE-754 float64 without intermediate rounding, concatenated with its shape string, and the layers are hashed in order. Empty tensors contribute a fixed sentinel. This construction binds the commitment to exact tensor values, order, and geometry; any change in the bytes the coordinator receives after a pre-commit alters $H(\theta)$ with overwhelming probability under standard collision resistance of SHA-256.

Each client publishes a `PreCommit` before transmitting its update to the coordinator. The coordinator then publishes an `Acknowledgment` containing the hash of the bytes actually received. After aggregation, the coordinator writes a `CoordinatorReceipt` containing the contributor list, per-contributor received hashes, the aggregate model hash, and any explicit rejections. In the next round, the verifier reads the previous round's records and publishes one or more `OmissionEvidence` records if a detectable failure occurred. Clients may also publish `Dispute` records before the dispute window closes, although the four-category verifier does not require disputes to reach a classification.

Algorithms 1 and 2 mirror the Flower strategy split `configure_fit/aggregate_fit` and the client `fit` path in the reference code: deadlines are wall-clock seconds on the coordinator before publication on-chain, acknowledgments use the same digest $H(\cdot)$ as pre-commits, aggregation delegates to Flower FedAvg, and omission verification (`VERIFYANDLOG`) always runs for round $t - 1$ beginning at $t = 2$, after a bounded wall-clock wait for confirmation, even when FedAvg yields no aggregated parameters that round (so `CoordinatorReceipt` may be absent while verification still executes).

Algorithm 1: BLOCKCHAINVERIFIEDFEDAVG coordinator hooks for training round t . Deadlines $T^{\text{pre}}, T^{\text{agg}}, T^{\text{dispute}}$ are UNIX-epoch cutoffs published in RoundStart; $\Delta_{\text{pc}}, \Delta_{\text{agg}}, \Delta_{\text{disp}}$ are policy offsets; $\delta > 0$ is the verifier propagation delay (implementation default ties δ to ledger block cadence)

```

1: procedure CONFIGUREFIT( $t$ , global parameters  $\Theta_{t-1}^{\text{glob}}$ )
2:    $s \leftarrow \text{UNIXSECONDS}()$ ;  $T^{\text{pre}} \leftarrow s + \Delta_{\text{pc}}$ ;  $T^{\text{agg}} \leftarrow s + \Delta_{\text{agg}}$ ;  $T^{\text{dispute}} \leftarrow s + \Delta_{\text{disp}}$ 
3:    $\mathcal{C}^{\text{exp}} \leftarrow \text{REGISTRYEXPECTEDCLIENTS}()$   $\triangleright$  empty on bootstrap failure; non-fatal
4:   chain-write RoundStart( $t$ , coordinator,  $\mathcal{C}^{\text{exp}}, T^{\text{pre}}, T^{\text{agg}}, T^{\text{dispute}}$ )
5:    $\mathcal{I} \leftarrow \text{FEDAVG.configure\_fit}(t, \Theta_{t-1}^{\text{glob}})$   $\triangleright$  samples clients under fraction_fit/quorum floors
6:   for each tuple  $(w, \text{FitIns}) \in \mathcal{I}$  do
7:     attach round_id  $\leftarrow t$  to fit configuration
8:     if wrapper  $w$  maps to known hospital id  $h$  then
9:        $B \leftarrow \text{POPENDINGACKS}(Q, h)$   $\triangleright$  remove all pending acks for  $h$  from store  $Q$ 
10:      if  $B \neq \emptyset$  then
11:        embed JSON list of  $\{\text{round} : r, \text{received\_hash} : d\}$  records for  $(r, d) \in B$  (round-sorted)
        into config key acks_for_previous_rounds
12:      end if
13:    end if
14:  end for
15:  return patched instructions  $\mathcal{I}$ 
16: end procedure
17: procedure AGGREGATEFIT( $t$ , successful results  $\mathcal{R}$ )
18:  Initialize ordered contributor list  $\mathcal{C}_t \leftarrow []$  and map  $\mathcal{D}_t: h \mapsto d_h$ 
19:  for each  $(w, \text{FitRes}) \in \mathcal{R}$  whose client metrics define node_id =  $h$  (otherwise skip this result) do
20:     $\Theta_h \leftarrow \text{NDARRAYS}(\text{FitRes.parameters})$ ;  $d_h \leftarrow H(\Theta_h)$ 
21:    chain-write Acknowledgment( $t, h, d_h$ ); append  $h$  to  $\mathcal{C}_t$ ;  $\mathcal{D}_t(h) \leftarrow d_h$ 
22:     $Q(h) \leftarrow Q(h) \cup \{(t, d_h)\}$   $\triangleright$  queued until  $h$  is next sampled, then drained by POPENDINGACKS
23:  end for
24:   $(\Theta_t^{\text{glob}}, \text{metrics}_{\text{fed}}) \leftarrow \text{FEDAVG.aggregate\_fit}(t, \mathcal{R})$ 
25:  if  $\Theta_t^{\text{glob}} \neq \perp$  then
26:    chain-write CoordinatorReceipt( $t$ , coordinator,  $\mathcal{C}_t, \mathcal{D}_t, H(\Theta_t^{\text{glob}}), \text{metrics}_{\text{fed}}$ )
27:  end if
28:  if  $t > 1$  then
29:    if  $\delta > 0$  then SLEEP( $\delta$ )
30:    end if  $\triangleright$  so round  $t-1$  items are confirmable before reads
31:    VERIFYANDLOG( $t-1$ )  $\triangleright$  pure ledger read + optional OmissionEvidence publish
32:  end if
33:  return  $(\Theta_t^{\text{glob}}, \text{metrics}_{\text{fed}})$ 
34: end procedure

```

Algorithm 2: Hospital NumPyClient.fit for hospital identity h enrolled in the consortium registry: inbound acknowledgments, local training, then ledger pre-commit on the exact tensors returned to Flower (hash H as in Algorithm 1)

```

1: procedure HOSPITALFIT(hospital identity  $h$ , configuration  $\Gamma$ , global parameters  $\Theta_{t-1}^{\text{glob}}$ )
2:   parse JSON list  $A$  from  $\Gamma$  at key acks_for_previous_rounds; if missing, set  $A \leftarrow \emptyset$ 
3:   for each record  $(r', d) \in A$  do  $\triangleright r'$  ranges over earlier rounds;  $t$  is the current round
4:     if  $r'$  exists in local pre-commit table  $P$  and  $d \neq P(r')$  then
5:       chain-write Dispute with type ACK_HASH_MISMATCH for round  $r'$   $\triangleright$  suspicious coordinator
        ack vs. prior commitment
6:     end if
7:   end for
8:   load  $\Theta_{t-1}^{\text{glob}}$  into local model; run local optimization (fixed epoch/batch regime from  $\Gamma$ )
9:    $\theta^h \leftarrow \text{FLATTENEDWEIGHTS}(\text{model})$   $\triangleright$  same tensors transmitted to Flower
10:  if round_id is present in  $\Gamma$  then
11:     $t \leftarrow \Gamma[\text{round\_id}]$ ;  $p \leftarrow H(\theta^h)$ ;  $P(t) \leftarrow p$ ; chain-write PreCommit( $t, h, p$ )
12:  end if
13:  return  $\theta^h$ , sample count, metrics including node_id  $\leftarrow h$ 
14: end procedure

```

2.3 Verification Procedure

Fig. 1 aligns one coordinator training round (indexed t in the implementation hooks) with the ledger cutoffs issued in RoundStart and with verification of omission evidence for round $t-1$, matching the control flow sketched in Algorithm 1.

Fix a round r and let T_r^{pre} and T_r^{agg} denote the absolute ledger deadlines announced in that round's RoundStart. Write $\tau(\cdot)$ for the ledger confirmation time of a record. The active pre-commit set is

$$\mathcal{P}_r = \left\{ c : \text{PreCommit}_c \text{ exists, } \tau \leq T_r^{\text{pre}} \text{ or } \tau = 0 \right\}, \quad (1)$$

where $\tau = 0$ is the sentinel for a record whose confirmation time is unset, which occurs when the classifier is replayed over an archived audit artifact rather than a live chain. Let $\mathcal{A}_r$ furnish optional acknowledgment digests a_c . Let $\tau(R)$ be the ledger confirmation time of receipt R ; declare R_r absent when no CoordinatorReceipt survives with $\tau(R_r) \leq T_r^{\text{agg}}$. For credited contributors define u_c as the receipt digest attached to hospital c in R_r ; denote by p_c each hospital's promised digest from PreCommit. Classification is deterministic in $(\mathcal{P}_r, \mathcal{A}_r, R_r)$.

If aggregates are unpublished, classify every client in $\mathcal{P}_r$ as **silent_drop**. When R_r is timely, each $c \in \mathcal{P}_r$ is handled as follows: if c is neither credited nor rejected explicitly, classify **silent_drop**; if c is rejected yet excluded from the contributor list, classify **fake_error**; if c is credited with pre-commit digest p_c , optional acknowledgment digest a_c , and receipt digest u_c , apply

$$\text{viol}(c) = \begin{cases} \text{bilateral_mismatch}, & a_c \neq \perp, a_c \neq p_c, a_c = u_c, \\ \text{partial_exclusion}, & a_c \neq \perp, p_c = a_c, u_c \neq p_c, \\ \text{partial_exclusion}, & a_c \neq \perp, a_c \neq p_c, a_c \neq u_c, \\ \text{partial_exclusion}, & a_c = \perp, u_c \neq p_c, \\ \text{nominal}, & \text{otherwise.} \end{cases} \quad (2)$$

A credited contributor therefore reaches one of three dispositions. It is labelled `bilateral_mismatch` when the acknowledgment and the receipt agree with each other but not with the pre-commit, `partial_exclusion` in the three remaining inconsistent configurations of Eq. (2), and `nominal` when every chain-visible digest for that hospital coincides. Only the `nominal` case produces no omission label. Algorithm 4 uses the same three outcomes under the same names.

Algorithms 3 and 4 specialize the omission classifier applied *after* the coordinator emits prior-round streams in Algorithm 1: Algorithm 3 audits an entire round from ledger state, while Algorithm 4 isolates the hash logic for a credited contributor.

Algorithm 3: Round- r omission classification (chain reader interface)

Input: Ledger reader Φ (round start, pre-commit digests, acknowledgments, receipts); round r ; verifying coordinator v .

Output: One omission record per nonempty violation list.

```

1:  $S \leftarrow \Phi.\text{ROUNDSTART}(r)$ 
2: if  $S = \perp$  then return  $\emptyset$ 
3: end if
4: Initialize  $\mathcal{P}$  as the active pre-commit set of Eq. (1)
5: if  $\mathcal{P} = \emptyset$  then return  $\emptyset$ 
6: end if
7:  $R \leftarrow \Phi.\text{COORDINATORRECEIPT}(r)$ 
8: if  $R \neq \perp$  and  $\tau(R) > T_r^{\text{agg}}$  then                                ▷ late receipt behaves as absent aggregate
9:    $R \leftarrow \perp$ 
10: end if
11: Initialize  $L_{\text{SD}}, L_{\text{FE}}, L_{\text{PE}}, L_{\text{BD}} \leftarrow \emptyset$ 
12: if  $R = \perp$  then
13:    $L_{\text{SD}} \leftarrow \text{sorted}(\mathcal{P})$ 
14: else
15:   for  $c \in \mathcal{P}$  do
16:     Read committed digest  $p_c$  via  $\Phi$  for hospital  $c$ 
17:     if  $c \in \text{contrib}(R)$  then                                ▷ priority: credited list overrides explicit reject map
18:        $u_c \leftarrow \text{rcvhash}(c, R)$ 
19:        $a_c \leftarrow$  acknowledgment digest for  $c$  if published, otherwise  $\perp$ 
20:        $\kappa \leftarrow \text{CLASSIFYHASHES}(p_c, u_c, a_c)$ 
21:       if  $\kappa = \text{bilateral\_mismatch}$  then
22:         append  $c \rightarrow L_{\text{BD}}$ 
23:       else if  $\kappa = \text{partial\_exclusion}$  then
24:         append  $c \rightarrow L_{\text{PE}}$ 
25:       end if

```

(Continued)

Algorithm 3 (continued)

```

26:     else if  $c \in \text{reject}(R)$  then
27:         append  $c \rightarrow L_{FE}$ 
28:     else
29:         append  $c \rightarrow L_{SD}$ 
30:     end if
31: end for
32: end if
33: Package each nonempty  $L_{SD}, L_{FE}, L_{PE}, L_{BD}$  into one omission record per type, signed under
    coordinator  $v$ 
34: return the packaged omission records

```

Algorithm 4: Hash-only decision for a credited contributor (CLASSIFYHASHES in Algorithm 3)

```

1: procedure CLASSIFYHASHES( $p, u, a$ )  $\triangleright$  digests  $(p_c, u_c, a_c)$ ;  $a = \perp$  iff no acknowledgment
2:   if  $a \neq \perp$  then
3:     if  $a \neq p$  and  $a = u$  then return bilateral_mismatch
4:   end if
5:   if  $p = a$  and  $u \neq p$  then return partial_exclusion
6:   end if
7:   if  $a \neq p$  and  $a \neq u$  then return partial_exclusion
8:   end if
9:   else
10:    if  $u \neq p$  then return partial_exclusion
11:  end if
12: end if
13: return nominal
14: end procedure

```

Each nonempty aggregate list in Algorithm 3 becomes one OmissionEvidence row for round r . Under the scripted attack model, omission or mismatched hashes authored by coordinator-side bookkeeping appear as `silent_drop`, `fake_error`, or `partial_exclusion`, whereas `bilateral_mismatch` indicates coherent coordinator records that disagree with the client's published commitment, that is, client-side misalignment absent further coordinator tampering evidence.

2.4 Detection Scope and Guarantees

The verifier's guarantees follow from the threat model in [Section 2.1](#) and from two properties of the record layer: publisher identity is bound to enrolled wallets in `enforce` mode, and record confirmation times are assigned by the ledger rather than by participant clocks. [Table 2](#) states, for each evidence category, the chain-state condition that triggers it, the guarantee that the trigger provides, and the assumptions that the guarantee requires beyond ledger integrity and collision resistance of H .

Three boundaries delimit this scope. First, the classifier is a total function over the finite comparison space of chain-visible digests: every client in the active pre-commit set $\mathcal{P}_r$ receives exactly one disposition per round (one of the four categories or nominal). Any violation that leaves an inconsistency among PreCommit, Acknowledgment, and CoordinatorReceipt records is therefore classified,

whether or not it was produced by one of the scripted injectors used in [Section 3](#); the injectors instantiate the four reachable inconsistency classes rather than defining them. Second, some events are inferable but not decidable. A client that pre-commits and never transmits is indistinguishable, on chain, from a client whose transmission was lost in transit, and the truthfulness of an explicit rejection reason cannot be established from digests alone; the dispute stream exists so that the affected party can attach a counter-claim within the dispute window. Third, attacks that leave chain-visible records mutually consistent are outside the threat model by construction: coordinator–client collusion in which both parties publish matching false records, content-level attacks such as poisoning or backdoors, consensus failures below the honest-majority assumption, and private-key compromise defeat the record layer rather than the classifier, and are delegated to orthogonal mechanisms as discussed in [Section 2.1](#).

Table 2: Per-category detection conditions and guarantees.

Category	Trigger Condition on Chain State	Guarantee	Additional Assumptions
<code>silent_drop</code>	In-window <code>PreCommit</code> for c ; receipt absent by T_r^{agg} , or c neither credited nor rejected in a timely receipt	Sound detection of unexplained exclusion of an in-window pre-committed update: an honest coordinator that included c 's update would have credited c	Client c 's pre-commit confirmed within the pre-commit window
<code>fake_error</code>	c committed in-window and appears in the explicit reject map without credit	Non-repudiable proof that a rejection was <i>asserted</i> against a committed client	Truthfulness of the stated reason is not decidable from digests; adjudication uses the dispute window
<code>partial_exclusion</code>	Credited digest u_c inconsistent with the committed digest p_c (and with a_c per Eq. (2))	Sound detection that the bytes credited differ from the bytes committed	None beyond the baseline assumptions
<code>bilateral_mismatch</code>	Acknowledgment and receipt agree with each other ($a_c = u_c$) but differ from the pre-commit ($a_c \neq p_c$)	Evidence of client-side deviation: coherent coordinator records against the client's own published commitment	Attribution to the client is an inference; the pre-commit remains the client's non-repudiable statement in either reading

Verification of round r executes during round $r + 1$ after the propagation delay δ , so omission evidence is available no later than one federated round plus δ after the violating round closes. A corollary is that a violation in the final training round is only audited if one further verification pass runs after training ends; deployments should therefore schedule a closing verification round (the evaluation harness enforces

the equivalent constraint that the attack round precede the final round). FLBC consequently targets post-hoc accountability rather than online prevention: the evidence identifies the violating party and round, and corrective action (for example, suspending a misbehaving coordinator or client) is a consortium-governance step layered above the protocol.

2.5 Implementation

The reference implementation is written in Python 3.10+ and integrates with the Flower FL framework and MultiChain Community Edition over JSON-RPC [2,16]. The implementation is organized into three modules. The `flbc.core` module contains the FL client and server classes. The client writes a `PreCommit` immediately before returning fit results. The server builds a blockchain-aware variant of FedAvg that announces rounds, records acknowledgments over the results Flower delivers to its aggregation hook, publishes round receipts, and triggers verification of the previous round.

The `flbc.blockchain.multichain` module implements the RPC connector. It serializes typed protocol records to JSON, publishes them to the appropriate stream, and decodes queried records back into typed objects. The connector supports three trust modes: `disabled` for shared-RPC simulation, a permissive bootstrap mode for development, and `enforce` for identity-bound deployments. In `enforce` mode, the connector rejects records whose publisher wallet does not match the wallet enrolled for the claimed participant identity.

The `flbc.protocol` module contains the schema dataclasses, parameter hashing utilities, and the pure-function verifier. The verifier depends only on a connector-shaped read interface, which makes it testable against both a real chain and in-memory test doubles. The implementation is accompanied by an extensive unit test suite covering hashing, schema round trips, verification behavior, connector trust modes, and the attack harness. Running `uv run pytest on tests/` collects the full suite at the frozen snapshot bundled with these experiments; exercising that suite before long sweeps helps catch regressions in connector trust switches, ledger record round trips, the inclusion verifier, and attack stubs. Those unit tests do not themselves emit the omission verdict aggregates in Tables 3 and 4: those aggregates come from scripted integrate-and-audit workflows described under the experimental setup, and the tests are aligned with the audited JSON artifacts they produce.

2.6 Experimental Setup

This subsection fixes the *evaluation contract* used with the FLBC protocol and implementation above: what is held constant across runs, which deployment shape exercises publisher identity, where outputs land on disk, and how headline tables relate to scripted sweeps. Two configurations are reported side by side: **simulation mode**, in which the MultiChain connector uses shared-RPC writing with trust `disabled` for detection sweeps, and **identity mode**, in which the same scenarios run inside the Docker Compose stack with per-hospital wallets and `enforce` so that stream publishers align with the consortium registry. Unless stated otherwise, Flower sampling, the EMNIST partition, the compact CNN, the hospital count $N = 5$, the coordinator designation `hospital_b`, and round budgets match between vanilla-Flower baselines and FLBC runs so differences isolate ledger coupling rather than task choice. Section 3 cites two frozen bundles checked in beside the manuscript, `audit_round0_bundle/` and `audit_round1_bundle/`; `audit_runs/` is the working directory the sweep scripts write to, not an archived bundle. Appendices A and B list shell entry points and default phase parameters.

Table 3: Detection summary for frozen bundle `audit_round0_bundle/`.

Scenario	Expected	Sim Fired	Sim Ok	Sim Viol.	Id Fired	Id Ok	Id Viol.
clean	—	—	✓	0	—	✓	0
silent_drop	silent_drop	2	✓	1	2	✓	1
fake_error	fake_error	2	✓	1	2	✓	1
partial_exclusion	partial_exclusion	2	✓	1	2	✓	1
bilateral_mismatch	bilateral_mismatch	2	✓	1	2	✓	1

Note: “Ok” means the audit tooling left `evaluation_passed` true: the expected omission fires once when armed, and no stray categories appear. “Fired” is the federated round to which the evidence pertains (verification of round $t - 1$ executes during round t , per Algorithm 1); “Viol.” counts classified omissions.

Table 4: Per-round omission evidence (`attack_round = 2`, victim `hospital_a`).

Scenario	Source	Round	Evidence	Victim
silent_drop	sim	2	silent_drop	hospital_a
silent_drop	identity	2	silent_drop	hospital_a
fake_error	sim	2	fake_error	hospital_a
fake_error	identity	2	fake_error	hospital_a
partial_exclusion	sim	2	partial_exclusion	hospital_a
partial_exclusion	identity	2	partial_exclusion	hospital_a
bilateral_mismatch	sim	2	bilateral_mismatch	hospital_a
bilateral_mismatch	identity	2	bilateral_mismatch	hospital_a

2.6.1 Deployment Topology

Fig. 2 summarizes logical federated-learning traffic, ledger records, and verifier readout. Except for the wide-area and two-host deployments of Section 3.5, identity-mode experiments run on a single Linux host (specifications are recorded with each bundle, e.g., in `audit_round0_bundle/env.txt`) using Docker Compose. The stack comprises five MultiChain nodes (one per emulated institution), a one-shot bootstrap service, a coordinator service, and one Flower client container per hospital. Each emulated institution therefore has a distinct container, node process, and wallet in enforce mode; this yields logical separation and publisher-bound identity without implying five physically separate operators or a wide-area deployment. Because every service shares one host, CPU, memory, disk I/O, and scheduling delays are correlated across containers and are not equivalent to cross-datacenter contention or inter-site latency.

2.6.2 Dataset, Model, and Round Configuration

The experiments use EMNIST-balanced [17]. For each class k , independently draw $\mathbf{p}^{(k)} \sim \text{Dirichlet}(\alpha, \dots, \alpha) \in \mathbb{R}^N$ with seeded randomness, shuffle the class- k training indices, split that list at cumulative proportion breakpoints given by $\mathbf{p}^{(k)}$, and assign the N shards to distinct hospitals [18]. Setting $\alpha = 0.5$ yields strong institutional label skew. This partition differs from the writer-keyed FEMNIST release distributed with LEAF, the benchmark suite for federated settings [19], and is referred to throughout as Dirichlet-partitioned EMNIST.

Flower instantiates synchronous FedAvg [1]: each participating hospital contributes a numerical update; the coordinator aggregates sample-weighted means. Connected clients obey `fraction_fit =  $\phi$` , matching `fraction_evaluate`, with quorum floors $m_{\min}$ enforced by Flower’s `min_*` client arguments; the reference artifacts use $\phi = 0.7$, $m_{\min} = 2$, and $N = 5$, so benign `clean` simulations train roughly three

hospitals per federated round, whereas scripted attack wrappers in the artifacts force deterministic victims into each attack round before sampling returns.

The learning model is a compact convolutional neural network that matches the EMNIST input shape and 47-class output space: two 5×5 convolutional layers of 16 and 32 channels, each followed by 2×2 max pooling, then fully connected layers of width 128 and 47, for 220,143 trainable parameters. The same model is used in both the baseline Flower runs and the FLBC runs. The evaluation topology contains five hospitals, with `hospital_b` designated as coordinator. Each emulated hospital runs its own MultiChain node process in an isolated container, and a bootstrap container provisions streams, funds joiner wallets, and enrolls each hospital's wallet in the registry.

Four scripted attack injectors implement the four violation categories. `silent_drop` removes the victim from the aggregate path. `fake_error` removes the victim and inserts a rejection reason. `partial_exclusion` tampers with the victim's receipt hash after a correct acknowledgment has been recorded. `bilateral_mismatch` mutates the victim's parameters after pre-commit and before transmission. The harness forces deterministic victim sampling on the configured attack round so that the expected evidence type is exercised reliably.

The empirical study uses two complementary sweeps. The detection sweep runs the clean scenario and the four attack scenarios under shared-RPC simulation and writes `audit_runs/audit_<scenario>.json` by default. The identity sweep runs the same scenarios in the Docker deployment with per-hospital wallets and registry enforcement, producing `audit_runs/identity_<scenario>.json`. The numerical results in [Section 3](#) were consolidated from two frozen bundles checked in alongside this submission: `audit_round0_bundle/` holds the original-submission measurements of [Tables 3–6](#) (JSON, `tables.md`, `overhead.csv`, and `env.txt`), and `audit_round1_bundle/` holds the revision measurements of [Tables 7–13](#). For overhead measurement, `experiments/measure_overhead.py` compares vanilla Flower and FLBC across repeated runs with the same model, partition, hospital count, round count, and compute device. Every generated audit JSON includes a provenance block containing the source commit hash, dirty-tree marker, and generation timestamp, and the full workflow is documented in the artifact reproduction instructions. [Appendix A](#) lists representative artifact paths ([Table A1](#)) and canonical shell entry points; [Appendix B](#) records default coordinator phase offsets, verifier delay settings, and Flower quorum parameters used by the artifact harness.

Table 5: Scenario-level tooling verdict parity.

Scenario	Sim Passed	Identity Passed	Parity
clean	✓	✓	✓
silent_drop	✓	✓	✓
fake_error	✓	✓	✓
partial_exclusion	✓	✓	✓
bilateral_mismatch	✓	✓	✓

Table 6: End-to-end wall time (seconds) for vanilla Flower vs. FLBC.

Scenario	Runs	Mean (s)	Per-Round (s)	Std (s)	Overhead
baseline	10	197.472	9.874	6.170	+0.00%
FLBC clean	10	271.502	13.575	7.149	+37.49%
FLBC silent_drop	10	275.634	13.782	5.058	+39.58%
FLBC fake_error [†]	10	1146.199	57.310	2700.167	+480.44%
FLBC partial_exclusion	10	279.165	13.958	5.259	+41.37%
FLBC bilateral_mismatch	10	283.766	14.188	6.299	+43.70%

Note: Per-round column divides the mean by 20. [†]fake_error: the sample standard deviation exceeds the mean, indicating a heavy-tailed run-time distribution on this host; see the discussion following this table.

Table 7: Re-measured end-to-end wall time with robust statistics (ten runs per scenario, 20 rounds, $N = 5$).

Scenario	Mean (s)	Median (s)	IQR (s)	Std (s)	Overhead (mean)	Overhead (median)
baseline	687.7	688.1	7.3	6.9	+0.00%	+0.00%
FLBC clean	797.7	796.9	14.1	10.2	+16.00%	+15.81%
FLBC silent_drop	943.0	945.2	11.1	8.5	+37.13%	+37.37%
FLBC fake_error	941.6	940.9	4.2	10.3	+36.92%	+36.74%
FLBC partial_exclusion	943.7	942.4	17.8	9.5	+37.23%	+36.96%
FLBC bilateral_mismatch	987.6	989.8	6.3	6.2	+43.61%	+43.85%

Note: Ten runs per scenario, interleaved across scenarios (one run of each per iteration) so environmental drift spreads evenly. Same physical machine as the original submission; the software environment was rebuilt for the revision (updated Python toolchain), so absolute times are not comparable with [Table 6](#); overhead percentages are the cross-study quantity. Raw per-run durations are archived in `overhead_runs.csv`, so these statistics can be recomputed from the artifact.

Table 8: Randomized-placement detection ($R = 20$, three seeded repetitions per scenario).

Scenario	Rep.	Drawn Round	Drawn Victim	Fired Round	Category Match
silent_drop	1	10	hospital_d	10	✓
silent_drop	2	10	hospital_e	10	✓
silent_drop	3	16	hospital_e	16	✓
fake_error	1	17	hospital_d	17	✓
fake_error	2	4	hospital_c	4	✓
fake_error	3	10	hospital_e	10	✓
partial_exclusion	1	11	hospital_e	11	✓
partial_exclusion	2	14	hospital_a	14	✓
partial_exclusion	3	15	hospital_a	15	✓
bilateral_mismatch	1	4	hospital_a	4	✓
bilateral_mismatch	2	19	hospital_e	19	✓
bilateral_mismatch	3	10	hospital_c	10	✓

Table 9: Ledger cost metrics per scenario ($R = 20$, $N = 5$).

Scenario	Coord. Writes/Round	Write Latency, Median (ms)	Coord. Payload (B/round)	Chain Growth (B/round)	Verifier (s/round)
clean	5.0	2.8	1593	2150	0.039
silent_drop	5.9	3.1	1908	2622	0.043
fake_error	6.6	2.6	2155	2992	0.043
partial_exclusion	6.0	2.7	1959	2691	0.045
bilateral_mismatch	5.3	2.8	1715	2324	0.042

Note: Coordinator write counts, latency, and verifier time are in-process (coordinator scope); the write latency is the per-publish JSON-RPC call to the coordinator's own node and excludes block-confirmation time. Chain growth is read back from the ledger across all publishers and counts FLBC record payloads only, excluding MultiChain per-item framing, inter-node replication traffic, and the Flower model-weight transmission.

Table 10: Identity-mode detection under emulated WAN latency ($R = 10$, $\delta = 3$ s).

Scenario	Unshaped Verdict	WAN Verdict	Unshaped Wall Clock (s)	WAN Wall Clock (s)
clean	✓ (0 violations)	✓ (0 violations)	42	48
silent_drop	✓ (round 2)	✓ (round 2)	43	51
fake_error	✓ (round 2)	✓ (round 2)	42	47
partial_exclusion	✓ (round 2)	✓ (round 2)	40	48
bilateral_mismatch	✓ (round 2)	✓ (round 2)	45	47

Table 11: Enforce-mode detection across two physical hosts (five institutions, heterogeneous architectures, $R = 3$, $\delta = 3$ s).

Scenario	Verdict	Evidence Round	Coordinator Wall Clock (s)
clean	✓ (0 violations)	—	19
silent_drop	✓	2	21
fake_error	✓	2	21
partial_exclusion	✓	2	23
bilateral_mismatch	✓	2	22

Table 12: Detection verdict and wall clock across timing and population configurations (silent_drop).

Config	δ (s)	Windows (s)	N	R	Detected	Wall Clock (s)
default	3	60/120/240	5	10	✓	176
$\delta = 0$	0	60/120/240	5	10	✓	136
$\delta = 1$	1	60/120/240	5	10	✓	157
$\delta = 5$	5	60/120/240	5	10	✓	176
windows 0.5×	3	30/60/120	5	10	✓	156
windows 2×	3	120/240/480	5	10	✓	156
$N = 3$	3	60/120/240	3	10	✓	216
$N = 8$	3	60/120/240	8	10	✓	136
$R = 5$	3	60/120/240	5	5	✓	105
$R = 20$	3	60/120/240	5	20	✓	276

Table 13: Final federated evaluation accuracy from clean runs, vanilla Flower vs. FLBC ($R = 20$, $N = 5$). The detection column refers to separate armed runs, not to the clean runs that produced the accuracy figures.

Dataset	Baseline Accuracy	FLBC Accuracy	Detection in a Separate Armed Run
EMNIST-balanced	0.8518	0.8690	(Tables 3 and 8)
CIFAR-10	0.7103	0.7401	silent_drop detected (round 2)

Note: Both accuracy columns come from clean runs, which produced zero violations on either dataset. The final column records a different run of the same dataset with an injector armed, and is listed here only to show that the detection path also works on CIFAR-10; for EMNIST the corresponding armed runs are those of Tables 3 and 8.

3 Results

This section pulls numbers from checked-in artifact bundles: paired JSON audits, timing CSVs, and per-host `env.txt` fingerprints. The original submission's measurements (Tables 3–6) were gathered on an Ubuntu QEMU/KVM machine; hardware and Docker revisions appear with Table 6. Except where marked below, scripted attacks used `attack_round = 2` and victim `hospital_a`. Sections 3.3–3.7 report the measurements added in revision: randomized attack placement, ledger cost decomposition, emulated wide-area latency, a two-host deployment, timing-parameter sensitivity, and dataset generality with a second-dataset accuracy check. Verdict-valued revision measurements (detection outcomes, evidence categories, accuracy) were collected on a second documented host (Apple-silicon macOS; MultiChain containers under x86-64 emulation, training native; fingerprint archived with the bundle); verdicts are deterministic functions of chain state and do not depend on the host. Table 7 reports overhead re-measured on the same Ubuntu machine as the original submission, since timing comparability requires a fixed host. Every audit JSON pins its source commit with a clean-tree marker.

3.1 Detection across Scenarios

Tables 3 and 4 place shared-RPC simulation (Sim) next to wallets resolved through the consortium registry (Id). Every armed scenario emits one omission verdict at federation round 2; the taxonomy entry matches what the injector was written to provoke, without duplicate evidence objects or unexplained side-lines. Each verdict marks an inconsistency among on-chain records classified into its category; attribution of intent, and the truthfulness of a stated rejection, are left to adjudication (Section 2.4).

Table 4 lists the omission label recorded per scenario and trust mode at round 2.

Across Tables 3 and 4, each scripted fault settles into exactly one verifier category instead of dissipating into undocumented coordinator behavior; Table 5 shows publisher-bound `enforce` mode recovering the classifications seen under shared-RPC simulation (Sim). Together, the artifacts document which omission label fired and tie contributors to ledger-visible aggregate hashes rather than leaderboard accuracy metrics alone.

3.2 Wall-Clock Overhead

Table 6 prints means from `audit_round0_bundle/overhead.csv`. The guest was Ubuntu 6.8.0-111-generic on four QEMU/KVM vCPUs with ≈ 7.7 GiB RAM; Docker 29.4.3 and Compose v5.1.3 match the stamp in `env.txt` (UTC 2026-05-10T17:54:50+00:00). Every reported mean uses ten full training jobs at 20 rounds and $N = 5$ hospitals. The overhead driver sets `attack_round = 8` via `min(8, 20)` inside `measure_overhead.py`, whereas the detection tables used round 2 above.

Across `clean`, `silent_drop`, `partial_exclusion`, and `bilateral_mismatch`, mean wall-clock time sits about 37%–44% above the Flower baseline mean under these settings, so tying training to ledger writes under this harness carries a measurable wall-clock premium (Table 6) beside the omission evidence surfaced above. For the `fake_error` scenario, the sample standard deviation (2700.167 s) exceeds the mean (1146.199 s), which for ten non-negative durations implies that a small number of runs lasted an order of magnitude longer than the remainder. The mean and the derived overhead percentage are therefore not robust statistics for this scenario, and no conclusion is drawn from this row. The measurement harness now records every per-run duration and reports medians and inter-quartile ranges beside means; Table 7 reports the re-measured sweep.

The re-measurement resolves the `fake_error` anomaly: across ten raw-logged runs its distribution is indistinguishable from the other attack scenarios (median 940.9 s, IQR 4.2 s, standard deviation 10.3 s), and the heavy tail of the original sweep did not recur in any of the sixty runs. Because the original harness retained only summary statistics, the specific cause of the original outlier is unrecoverable; the per-run log now makes any recurrence directly attributable. The re-measured sweep also separates two costs the original conflated: the `clean` row (+16% mean) is the protocol's overhead (pre-commits, acknowledgments, receipts, and verification attached to every round), whereas the attack rows (+37%–44%) additionally include the evaluation harness forcing full-cohort sampling during warm-up rounds so the scripted victim is deterministically resolvable, a harness cost rather than a protocol one.

3.3 Detection under Randomized Attack Placement

The detection results above use a fixed placement (attack round 2, victim `hospital_a`), which leaves open whether the verifier merely recognizes that configuration. To probe whether detection depends on the specific placement, the sweep re-runs each attack scenario three times at $R = 20$ with the attack round drawn uniformly from $[2, R - 1]$ and the victim drawn uniformly from the non-coordinator hospitals, both from a seeded generator so the draws are reproducible. The final round is excluded by the protocol's own verification schedule: round t is audited during round $t + 1$ (Section 2.4), so a violation in the last round of a run without a closing verification pass is structurally unauditably rather than undetected. Table 8 lists every draw and its verdict: all twelve draws, spanning rounds 4 through 19 and all four non-coordinator victims, were detected at exactly the drawn round with the expected evidence category, no duplicate evidence, and no unexpected categories; the clean control produced zero violations. This is twelve successful sampled placements, not a proof of invariance to every possible placement.

3.4 Ledger Cost Metrics

Wall-clock overhead alone does not decompose where the ledger coupling spends its budget. The instrumented harness reports three cost surfaces. First, coordinator-side write latency: the connector times every stream publish and records its payload size. This latency is the JSON-RPC publish call to the coordinator's own MultiChain node; it excludes block-confirmation time, which is decoupled from the write path and absorbed by the verifier propagation delay δ (Section 2.4). Second, chain storage growth: every audit JSON carries a `ledger_footprint` block with per-stream record counts and payload bytes read back from the chain across all publishers, so per-round growth follows directly. Third, verifier compute: the end-to-end duration of each `verify_and_log` call (chain reads, pure classification, and evidence publishes). Table 9 summarizes the three surfaces for $R = 20$ detection runs. The absolute costs are small: the coordinator publishes five to seven records per round at a median publish latency under 4 ms, total ledger growth is 2.1–3.0 KB per round across all publishers (attack rounds add omission evidence, hence the spread), and verification of a prior round costs about 40–45 ms, three orders of magnitude below the round's

training time. Because records are fixed-size digests and per-round record count is linear in the number of participants, storage growth is independent of model size and scales linearly in N and R . These byte figures count FLBC record payloads only: they exclude MultiChain's inter-node consensus and replication traffic and JSON-RPC transport framing, and they exclude the Flower model-weight transmission, which is the vanilla-Flower communication cost that FLBC leaves unchanged and never places on chain.

3.5 Operation under Emulated WAN Latency

To probe the gap between the single-host deployment and a geographically distributed consortium, a network-emulation overlay assigns each institution a one-way `netem` delay (10, 25, 40, 60, and 80 ms for `hospital_a-e`, with $\sim 20\%$ jitter, normal distribution) applied to chain peer-to-peer traffic and Flower traffic, while each hospital's LAN-local RPC to its own node remains unshaped; pairwise round-trip times span roughly 35–160 ms. The sweep verifies that every shaper attached before training progressed and aborts a scenario rather than let it run unshaped, a safeguard added after an early harness iteration silently under-shaped short runs. The identity-mode sweep then runs unchanged under this profile at $R = 10$. [Table 10](#) compares verdicts and coordinator wall clock against unshaped identity runs at the same round budget.

Verdicts are identical with and without injected latency, and repeating the shaped sweep with the verifier delay raised to $\delta = 10$ s reproduced every verdict, so detection is insensitive to δ in this regime for the same reason as in [Section 3.6](#): at ~ 4 – 5 s per round, prior-round records confirm well before verification reads them. Wall clock grows by 4%–20% under shaping; training dominates the round, so injected latency shows up mainly in coordination edges. One incidental observation from harness development is worth recording: in an early iteration whose shaper attachment could restart the coordinator container mid-round, the verifier flagged the disrupted rounds with omission evidence (pre-commits without matching credit), which shows that a coordinator failure mid-round is chain-visible and lands in the same evidence categories as deliberate omission, with attribution again left to adjudication ([Section 2.4](#)).

Beyond emulation, the artifact includes a per-institution deployment profile that runs the same consortium across physically separate machines: each institution's MultiChain node, wallet, and FLBC process run on its host, chain peer-to-peer and RPC traffic cross a WireGuard-based private mesh, and identity enforcement operates unchanged. The five institutions ran across two hosts with heterogeneous processor architectures: an Apple-silicon machine hosts `hospital_a` (chain creator) and the coordinator `hospital_b`, and an x86-64 machine hosts `hospital_c-e`. Unlike the Compose replica, the institutions no longer share a kernel, scheduler, or physical resources. Bringing up each scenario is a manual per-host procedure rather than a scripted sweep, so this deployment uses a shorter round budget of $R = 3$ at the default verifier delay $\delta = 3$ s. [Table 11](#) reports enforce-mode detection under this topology.

The emulation injects latency and jitter only, and the two-host deployment separates physical resources without separating administrative domains; together they narrow rather than close the deployment gap discussed in [Section 4.1](#): asymmetric partitions, link loss, clock drift, and operationally independent operators remain future work.

3.6 Parameter Sensitivity

Detection correctness is deterministic in the confirmed chain records ([Section 2.4](#)); the timing parameters govern whether honest records confirm inside their windows. The sensitivity sweep varies one factor at a time from the defaults of [Appendix B](#): the verifier delay $\delta \in \{0, 1, 3, 5\}$ s, the phase windows at $0.5\times$ and $2\times$, the hospital count $N \in \{3, 5, 8\}$, and the round budget $R \in \{5, 10, 20\}$, running the `silent_drop` scenario per configuration. [Table 12](#) reports the verdict and wall clock per configuration.

Every configuration detected the attack, including $\delta = 0$, which had been expected to race block confirmation. The explanation is instructive: with local training taking several seconds per round, the previous round's writes confirm during the next round's training regardless of δ , so the sweep's round durations already exceed the 2 s block interval by a wide margin. δ matters only when a round completes faster than the ledger block interval or when confirmation itself slows, and neither condition arises in the configurations tested here. Within this envelope, detection is insensitive to all four factors, and wall clock responds to the round budget and population as expected.

3.7 Dataset Generality and Observed Accuracy

FLBC's record layer operates on parameter digests and is agnostic to the learning task, but two empirical checks make that concrete. First, the detection path re-runs on CIFAR-10 under the same Dirichlet- $\alpha = 0.5$ partition protocol with a larger convolutional network: three 3×3 convolutional layers of 32, 64, and 128 channels, each followed by 2×2 max pooling, then fully connected layers of width 256 and 10. That network holds 620,362 trainable parameters against the EMNIST model's 220,143, a factor of 2.8, so the hash and commitment path runs on a correspondingly larger update payload. That CIFAR-10 detection run arms the `silent_drop` injector and is separate from the clean runs used for accuracy below. Second, because FLBC wraps rather than replaces FedAvg, coupling training to the ledger should not change learning outcomes by construction; as a functional sanity check on a second dataset, both runners report the final sample-weighted federated evaluation accuracy from clean runs, compared in [Table 13](#).

On both datasets the FLBC run's final accuracy is close to the vanilla baseline, marginally higher in these particular draws. A single paired run per dataset does not support a quantitative accuracy comparison. FLBC's record layer hashes the exact tensors FedAvg returns and leaves client selection and aggregation unchanged, so parity is expected by construction rather than established empirically here, and the small residual gap reflects the baseline's own run-to-run variance under stochastic client sampling and a non-IID partition, which was not separately quantified.

4 Discussion

Compared with neighboring federated-learning and ledger designs, BAFFLE removes the aggregator via on-contract averaging, so omission by a standalone coordinator disappears by construction [7]; BlockFlow scores contributions under differential privacy-aware incentives [8]; Biscotti stresses decentralized averaging with poisoning resistance and hidden update payloads [9]; VeriFL targets arithmetic fidelity of aggregation through cryptographic commitments [10]; VFChain records committee-signed proofs of aggregation in a searchable authenticated data structure [11]; and the decentralized framework of Kalapaaking et al. replaces the central authority with per-participant monitoring contracts and a multisigned global model [12]. Surveys of blockchain-based FL organize this space by ledger role, aggregation topology, and threat focus [13–15], and applied systems in healthcare have used blockchain-enabled FL for malicious-node detection in decentralized clinical networks [20]. [Table 14](#) situates FLBC along the axes most relevant to accountability. FLBC targets mechanically checkable coordinator inclusion decisions anchored to ledger-readable phase timestamps; the stacks address different failure modes and can be complementary. A quantitative head-to-head benchmark is not meaningful across this set, because the systems answer different questions (aggregator removal, contribution scoring, poisoning resistance, aggregation arithmetic) with no shared correctness metric, and not all publish runnable artifacts; the comparison is therefore structural.

Table 14: Structural comparison with adjacent blockchain-FL systems.

System	Aggregator/Ledger Role	Primary Failure Mode Addressed	Inclusion-Decision Audit Output
BAFFLE [7]	No coordinator; aggregation on contract	Single-aggregator trust, removed by construction	Not applicable (no designated coordinator)
BlockFlow [8]	Decentralized evaluation; ledger scores contributions	Unfair attribution, collusion	Contribution scores, not per-client inclusion evidence
Biscotti [9]	Peer-to-peer committees; ledger coordinates rounds	Poisoning, privacy of updates	None (focus on update content)
VeriFL [10]	Central untrusted aggregator; cryptographic commitments	Incorrect aggregation arithmetic	Proof of correct sum, not per-client inclusion
VFChain [11]	Rotating blockchain committee aggregates; ledger stores verifiable proofs	Incorrect execution of the training procedure	Searchable proofs that aggregation ran as claimed
Kalapaaking et al. [12]	No central authority; per-participant monitoring contracts on chain	Single point of failure at a central authority	Monitoring records and a multisigned global model
FLBC (this work)	Designated coordinator; ledger as evidence layer	Omission and misattribution of contributions	Per-client omission evidence in four categories; regulator-readable JSON

Recording coordination metadata on a shared ledger raises privacy considerations of its own. Contributor lists, per-round timing, and explicit rejection reasons can reveal operational information (which institutions participated in which rounds, how often an institution was rejected, and when its infrastructure was active) even though no model parameters or raw data appear on chain; FLBC publishes only fixed-size digests. In cross-silo settings among commercially competing institutions, this metadata may itself be sensitive. Permissioned deployments can mitigate the exposure: MultiChain stream read permissions can restrict audit visibility to consortium members and designated auditors, participant identifiers can be pseudonymized against an off-chain enrollment registry, and rejection reasons can be published as commitments that are opened only inside a dispute. Quantifying residual metadata leakage, and combining the evidence layer with differential privacy or secure aggregation on the update path, remains open.

Verifier output rests on tamper-evident stream items that can bolster documentation narratives under the EU AI Act and GDPR without substituting compliance judgment [4,5]. The primary deployment is deliberately a single-documented-host Docker Compose replica with logically separate wallets, chosen so that empirical claims stay reproducible rather than because a single host reflects production practice.

4.1 Limitations and Future Work

The gap between the tested setups and a production consortium is narrower after this revision but has not closed. Headline numbers come from a single-host Docker Compose replica where five institutions share a kernel and scheduler; the emulated wide-area profile (Table 10) and the two-host deployment (Table 11) separate injected latency and physical resources, yet neither reproduces asymmetric link failures, packet loss, clock drift, or administratively independent operators. `hospital_b` coordinates every round here, so coordinator rotation is untested, as are production-scale cohorts and the sub-second-round regime in which the verifier delay δ starts to matter. Detection is post-hoc rather than preventive: evidence appears one federated round plus δ after the violating round closes, and a violation in the final training round is audited only if a closing verification pass runs after training ends (Section 2.4), so deployments must schedule that pass or treat last-round evidence as unavailable.

The verifier's correctness rests on the informal argument of Section 2.4 under the threat model of Section 2.1. A machine-checked proof would sharpen it. The evidence categories classify a failure but say nothing about its severity, how often it recurs over a training run, or what it did to the resulting model, and an auditor ranking incidents would want all three. Composition is also unexplored: differential privacy, secure aggregation, and Byzantine-robust aggregators each sit on the update path, and any of them could disturb on-chain timing, hash consistency, or how readable the omission evidence turns out to be. The questions that will take longest to answer, though, are institutional rather than technical. Machine-verifiable output and evidence admissible in a real dispute are not the same thing, so whether these records satisfy regulators under the EU AI Act, GDPR, or FDA guidance remains an open question [4–6]. A live consortium still has to settle who coordinates, who is liable when something goes wrong, and how keys and membership are handled among institutions that compete commercially. A clinical pilot would then test whether compliance officers can read what the verifier emits at all.

5 Conclusions

Accountability in cross-silo federated learning does not reduce to data locality. Even when raw data never leaves an institution, a coordinator can drop a client's update without saying so, record an exclusion reason that was never true, or aggregate weights other than the ones it acknowledged, and a standard FL protocol preserves no trace that another party can check. FLBC addresses that gap by binding each federated round to a permissioned ledger. Clients publish a hash of their update before transmitting it, the coordinator publishes signed acknowledgments of the tensors it received and a receipt after aggregation, and every phase deadline is announced on chain so that timeliness is judged by ledger confirmation times rather than by participant clocks. A verifier reads only that chain state and assigns each pre-committed client exactly one disposition per round, drawn from four categories of detectable inclusion failure: `silent_drop`, `fake_error`, `partial_exclusion`, and `bilateral_mismatch`. Because the verifier is a pure function of the record, any party holding the same ledger derives the same verdict.

A reference implementation on Flower and MultiChain ran on EMNIST under a Dirichlet- $\alpha = 0.5$ label-skew partition across five emulated institutions, first in shared-RPC simulation and then with per-hospital wallets under identity enforcement. Every scripted fault produced the evidence category it was built

to provoke, clean runs produced none, and the verdicts agreed across trust modes. The same verdicts survived randomized attack placement, emulated wide-area latency, a two-host deployment across differing processor architectures, and a second dataset. Re-measurement with per-run logging resolved a timing anomaly in the original sweep and puts the clean protocol path at roughly +16% wall clock over vanilla Flower; the larger figures for attack scenarios include an evaluation harness that forces full-cohort sampling during warm-up rounds, so they are a cost of the harness rather than of the protocol. Ledger growth stays near three kilobytes per round and verifying a prior round takes tens of milliseconds, three orders of magnitude below the round's training time.

What FLBC contributes is a record, not a judgment. The evidence marks an inconsistency among on-chain records; it does not certify that a stated rejection was truthful, and it cannot separate an update the coordinator withheld from one a client never sent or the network lost. Adjudication stays with the consortium, informed by the dispute stream rather than settled by the verifier. The audit output is machine-readable JSON naming which institution submitted what, what the coordinator acknowledged, and what reached the aggregate, and each audit file is pinned to the code commit that produced it, so a reader can regenerate the result from the archived artifact. FLBC is built to sit beside privacy-preserving and Byzantine-robust aggregation rather than to replace either: it records what was submitted and what was included, and leaves the question of whether a contribution was correct or beneficial to methods designed to answer it.

Acknowledgement: The authors extend their appreciation to the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University (IMSIU) for funding this work through grant number IMSIU-DDRSP2604. The authors used generative AI tools selectively, for exploratory software scaffolding outside the verifier core and for drafting code documentation in the research artifact repository, such as the README and reproduction instructions. These tools were used as assistive aids only and are not credited as authors or contributors. All artifacts used in the reported experiments were reviewed, tested, and validated by the authors, who take responsibility for the final content of the manuscript and the associated research artifacts.

Funding Statement: This work was supported and funded by the Deanship of Scientific Research at Imam Mohammad Ibn Saud Islamic University (IMSIU) (grant number IMSIU-DDRSP2604).

Author Contributions: The authors confirm contribution to the paper as follows: conceptualization, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Sultan Noman Qasem, Abdullah Albarrak, Tawfik Al-Hadhrami and Kwame Assa-Agyei; methodology, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Sultan Noman Qasem, Abdullah Albarrak, Tawfik Al-Hadhrami and Kwame Assa-Agyei; software, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Sultan Noman Qasem, Abdullah Albarrak, Tawfik Al-Hadhrami and Kwame Assa-Agyei; validation, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Sultan Noman Qasem, Abdullah Albarrak, Tawfik Al-Hadhrami and Kwame Assa-Agyei; formal analysis, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Sultan Noman Qasem, Abdullah Albarrak, Tawfik Al-Hadhrami and Kwame Assa-Agyei; investigation, Neil Kwabena Afum Ohene and Frederick Edem Junior Broni; resources, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Sultan Noman Qasem, Abdullah Albarrak, Tawfik Al-Hadhrami and Kwame Assa-Agyei; data curation, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Sultan Noman Qasem, Abdullah Albarrak, Tawfik Al-Hadhrami and Kwame Assa-Agyei; writing—original draft preparation, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Tawfik Al-Hadhrami and Kwame Assa-Agyei; writing—review and editing, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Sultan Noman Qasem, Abdullah Albarrak, Tawfik Al-Hadhrami and Kwame Assa-Agyei; visualization, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Sultan Noman Qasem, Abdullah Albarrak, Tawfik Al-Hadhrami and Kwame Assa-Agyei; supervision, Neil Kwabena Afum Ohene and Frederick Edem Junior Broni; project administration, Neil Kwabena Afum Ohene, Frederick Edem Junior Broni, Sultan Noman Qasem, Abdullah Albarrak, Tawfik Al-Hadhrami and Kwame Assa-Agyei; funding acquisition, Sultan Noman Qasem, Abdullah Albarrak and Tawfik Al-Hadhrami. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the Corresponding Author upon reasonable request. The EMNIST-balanced dataset is obtained from the NIST EMNIST distribution (https://biometrics.nist.gov/cs_links/EMNIST/gzip.zip) via the reference implementation's torchvision loader. Research artifacts are available at <https://github.com/0xDVC/flbc>.

Ethics Approval: Not applicable. This study did not involve human participants or animals.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial intelligence
CNN	Convolutional neural network
CPU	Central processing unit
DOI	Digital object identifier
EMNIST	Extended MNIST (Modified National Institute of Standards and Technology)
EU AI Act	European Union Artificial Intelligence Act
FDA	U.S. Food and Drug Administration
FedAvg	Federated averaging
FEMNIST	Federated Extended MNIST
FL	Federated learning
FLBC	Federated Learning Blockchain
GDPR	General Data Protection Regulation
IEEE	Institute of Electrical and Electronics Engineers
JSON	JavaScript Object Notation
JSON-RPC	JSON Remote Procedure Call
KVM	Kernel-based Virtual Machine
LTS	Long-term support
NIST	National Institute of Standards and Technology
POSIX	Portable Operating System Interface
QEMU	Quick Emulator
RAM	Random-access memory
RPC	Remote procedure call
SHA-256	Secure Hash Algorithm 256-bit
UTC	Coordinated Universal Time
WAN	Wide-area network

Appendix A Reproduction Checklist

This appendix summarizes artifact filenames and representative commands omitted from the experimental setup subsection for brevity. [Table A1](#) lists the representative artifacts and their role in the empirical study. Sweep scripts write their output under `audit_runs/`; the archived bundles pinned for [Section 3](#) (`audit_round0_bundle/` for the original submission and `audit_round1_bundle/` for the revision) mirror these filenames. The authoritative step-by-step script is bundled with the public research artifact (see Availability of Data and Materials). All commands assume POSIX shell from the checkout root after `uv sync --frozen --no-dev`.

Appendix A.1 Suggested Host

Runs used for submission-grade numbers should use a documented Linux x86_64 guest (Ubuntu Server LTS recommended) with Docker Engine, the Compose plugin, and uv; capture `uname`, CPU model, RAM, and Docker versions once into an `env.txt` stored beside the artifacts being cited. Sweeps write `audit_runs/env.txt` by default; the original-submission bundle keeps `audit_round0_bundle/env.txt`, and the revision bundle records per-host fingerprints in `env_mac.txt` and `env_vm.txt`. macOS is acceptable for smoke tests but introduces extra virtualisation variance.

Table A1: Representative artifacts.

Path/Glob	Role in the Empirical Study
<code>audit_runs/env.txt</code>	Host and toolchain fingerprint
<code>audit_runs/audit_<scenario>.json</code>	Detection verdicts (shared-RPC simulation)
<code>audit_runs/audit_<scenario>_rep<K>.json</code>	Verdicts under randomized attack placement
<code>audit_runs/identity_<scenario>.json</code>	Verdicts with per-hospital wallets (enforce)
<code>audit_runs/identity_wan_<scenario>.json</code>	Enforce-mode verdicts under WAN emulation
<code>audit_runs/timing_<scenario>.csv</code>	Flower-side timings alongside audits
<code>audit_runs/overhead.csv</code>	Wall-clock summary from <code>measure_overhead</code>
<code>audit_runs/overhead_runs.csv</code>	Raw per-run durations (crash-safe append)
<code>audit_runs/ledger_metrics_<scenario>.json</code>	Coordinator write latency, bytes, verifier time
<code>audit_runs/sensitivity.csv</code>	Detection and timing across parameter configs

Note: Default output directory is `audit_runs/`; the archived bundles are `audit_round0_bundle/` (original submission, [Tables 3–6](#)) and `audit_round1_bundle/` (revision, [Tables 7–13](#)).

Appendix A.2 Minimal Validation Ladder

1. `uv run pytest` exercises hashing, verifier logic, schemas, connector trust modes, ledger-metric instrumentation, and attack stubs.
2. `bash experiments/run_all_scenarios.sh` reproduces simulation-mode detection artifacts under Flower with a shared RPC publisher (`clean`, `silent_drop`, `fake_error`, `partial_exclusion`, `bilateral_mismatch`). Setting `RANDOM_PLACEMENT=1` draws the attack round and victim from a seeded generator per repetition instead of the fixed `round-2/hospital_a` placement.
3. `bash experiments/run_identity_sweep.sh` tears down Compose volumes between scenarios and reproduces publisher-bound identities on the five-node stack. Setting `WAN_PROFILE=1` layers a network-emulation overlay that injects per-site `netem` delays (10–80 ms one-way, ~20% jitter) on chain peer-to-peer and Flower traffic while leaving each institution’s LAN-local RPC unshaped.
4. `bash experiments/run_sensitivity_sweep.sh` varies the verifier delay δ , the phase-window offsets, the hospital count, and the round budget one factor at a time and records detection verdicts and wall clock per configuration.
5. `uv run python -m experiments.measure_overhead` uses the flags recorded in `REPRODUCE.md`; it appends every individual run to a raw per-run CSV as it completes and reports medians and inter-quartile ranges beside means. Paper-ready markdown tables consolidate `audit_runs/` via `uv run python -m experiments.build_tables`.

Both training entry points additionally accept `-dataset cifar10` (identical Dirichlet partition protocol, larger CNN) and emit the final federated evaluation accuracy, enabling an observed-accuracy comparison between vanilla Flower and FLBC. Each scenario run records coordinator-side ledger metrics

(per-write publish latency, payload bytes, verifier durations) to `ledger_metrics_<scenario>.json`, and each audit JSON carries a `ledger_footprint` block with per-stream record counts and payload bytes read back from the chain across all publishers.

Appendix B Default Timing and Sampling Parameters

Table A2 aligns manuscript symbols Δ_{pc} , Δ_{agg} , Δ_{disp} and propagation delay δ with the reference implementation constants. Ledger writes use UNIX-epoch cutoff fields published inside each `RoundStart`; the verifier inserts a wall-clock sleep before querying round $t - 1$ so MultiChain confirmations (target block interval set to 2 s in the Compose profile) settle before omission reads, which mirrors Algorithms 1–3.

Table A2: Default timing and sampling parameters.

Setting	Symbol	Value	Remark
Pre-commit window offset	Δ_{pc}	60 s	Added to UNIX time at <code>RoundStart</code>
Aggregate window offset	Δ_{agg}	120 s	Same origin as Δ_{pc}
Dispute-window close offset	Δ_{disp}	240 s	Bounds client <code>Dispute</code> issuance
Verifier propagation delay	δ	3 s	Idle before <code>verify_and_log(t - 1)</code>
Flower <code>fraction_fit/evaluate</code>	ϕ	0.7	Sampling quorum numerator
Flower <code>min_* clients</code>	m_{min}	2	Quorum floor on fit, eval, availability
Target block interval (Compose)	—	2 s	<code>start_multichain.sh</code> rewrites block time

Note: $N = 5$ hospitals; `hospital_b` is coordinator. Epoch and batch sizes follow `on_fit_config_fn` on the Flower server.

References

- McMahan HB, Moore E, Ramage D, Hampson S, Y Arcas BA. Communication-efficient learning of deep networks from decentralized data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS); 2017 Apr 20–22; Fort Lauderdale, FL, USA. p. 1273–82.
- Beutel DJ, Topal T, Mathur A, Qiu X, Parcollet T, Gao Y, et al. Flower: a friendly federated learning research framework. arXiv:2007.14390. 2020.
- Upreti D, Yang E, Kim H, Seo C. A comprehensive survey on federated learning in the healthcare area: concept and applications. Comput Model Eng Sci. 2024;140(3):2239–74. doi:10.32604/cmesci.2024.048932.
- European Parliament and Council of the European Union. Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act) [Internet]. 2024 [cited 2026 May 10]. Available from: <https://eur-lex.europa.eu/eli/reg/2024/1689/oj>.
- European Parliament and Council of the European Union. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 (General Data Protection Regulation) [Internet]. 2016 [cited 2026 May 10]. Available from: <https://eur-lex.europa.eu/eli/reg/2016/679/oj>.
- U.S. Food and Drug Administration. Artificial intelligence/machine learning (AI/ML)-based software as a medical device (SaMD) action plan [Internet]. 2021 [cited 2026 May 10]. Available from: <https://www.fda.gov/media/145022/download>.
- Ramanan P, Nakayama K. BAFFLE: blockchain based aggregator free federated learning. arXiv:1909.07452. 2019.
- Mugunthan V, Rahman R, Kagal L. BlockFlow: an accountable and privacy-preserving solution for federated learning. arXiv:2007.03856. 2020.
- Shayan M, Fung C, Yoon CJM, Beschastnikh I. Biscotti: a blockchain system for private and secure federated learning. IEEE Trans Parallel Distrib Syst. 2021;32(7):1513–25. doi:10.1109/TPDS.2020.3044223.
- Guo X, Liu Z, Li J, Gao J, Hou B, Dong C, et al. VeriFL: communication-efficient and fast verifiable aggregation for federated learning. IEEE Trans Inf Forensics Secur. 2021;16:1736–51. doi:10.1109/TIFS.2020.3043139.

11. Peng Z, Xu J, Chu X, Gao S, Yao Y, Gu R, et al. VFChain: enabling verifiable and auditable federated learning via blockchain systems. *IEEE Trans Netw Sci Eng.* 2022;9(1):173–86. doi:10.1109/TNSE.2021.3050781.
12. Kalapaaking AP, Khalil I, Yi X, Lam KY, Huang GB, Wang N. Auditable and verifiable federated learning based on blockchain-enabled decentralization. *IEEE Trans Neural Netw Learn Syst.* 2025;36(1):102–15. doi:10.1109/TNNLS.2024.3407670.
13. Zhu J, Cao J, Saxena D, Jiang S, Ferradi H. Blockchain-empowered federated learning: challenges, solutions, and future directions. *ACM Comput Surv.* 2023;55(11):240. doi:10.1145/3570953.
14. Tang Y, Zhang Y, Niu T, Li Z, Zhang Z, Chen H, et al. A survey on blockchain-based federated learning: categorization, application and analysis. *Comput Model Eng Sci.* 2024;139(3):2451–77. doi:10.32604/cmesci.2024.030084.
15. Jiang Y, Ma B, Wang X, Yu G, Yu P, Wang Z, et al. Blockchain federated learning for Internet of Things: a comprehensive survey. *ACM Comput Surv.* 2024;56(10):1–37. doi:10.1145/3659099.
16. Greenspan G. MultiChain private blockchain—white paper [Internet]. 2015 [cited 2026 May 10]. Available from: <https://www.multichain.com/download/MultiChain-White-Paper.pdf>.
17. Cohen G, Afshar S, Tapson J, van Schaik A. EMNIST: extending MNIST to handwritten letters. In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*; 2017 May 14–19; Anchorage, AK, USA. p. 2921–6.
18. Hsu TMH, Qi H, Brown M. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv:1909.06335*. 2019.
19. Caldas S, Duodu SMK, Wu P, Li T, Konečný J, McMahan HB, et al. LEAF: a benchmark for federated settings. *arXiv:1812.01097*. 2018.
20. Sonani R, Alhejaili R, Chatterjee P, Alnafisah KH, Ali J. Secure malicious node detection in decentralized healthcare networks using cloud and edge computing with blockchain-enabled federated learning. *Comput Model Eng Sci.* 2025;144(3):3169–89. doi:10.32604/cmesci.2025.070225.