RollupFL: An Auditable Federated Learning Framework for Byzantine Client Accountability

Md Tahmid Ashraf Chowdhury[1], Fasee Ullah[2],
Shanjida Islam Labonno[3], Shahid Kamal[4]*, Mohammad Ahsanul Islam[5]
Department of Computing, Universiti Teknologi PETRONAS, Seri Iskandar, Perak, Malaysia[1,2,3,5]
Center for Advanced Analytics, CoE for Artificial Intelligence
Multimedia University, Persiaran Multimedia 63100 Cyberjaya, Selangor, Malaysia[4]

***Abstract*—Federated learning (FL) trains a shared model without sending raw data, but some clients can be Byzantine and send harmful updates. Robust aggregation methods like Median and Krum can reduce poisoning damage, but they do not clearly show which client attacked. In this study, we propose RollupFL, an audit layer for FL that improves accountability under Byzantine attacks. RollupFL keeps aggregation and auditing separate, so it can work with FedAvg, Median, or Krum without changing how aggregation is computed. We study two audit designs: simple logging, which is fast, but assumes a trusted server, and blockchain-based audit, which gives stronger integrity and attribution, but adds more latency. We evaluate MNIST training for 20 rounds with 10%–30% Byzantine clients under sign-flip and model-replacement attacks. Results show that auditing does not meaningfully change accuracy, but it improves accountability. At 30% Byzantine, blockchain audit achieves higher attribution (0.95) and tamper detection (0.92) than logging (0.65 and 0.58). Logging adds small per-round latency, while blockchain adds larger latency mainly due to ledger writing.**

***Keywords*—Federated learning; Byzantine attacks; audit layer; accountability; attacker attribution; tamper detection; robust aggregation; FedAvg; blockchain audit; sign-flip attack; model-replacement attack**

I. Introduction

A. Background and Motivation

Federated learning lets many clients train one model without sending raw data. This is useful when the data is private or regulated. Some recent works also add auditing, so model updates can be checked later [1], [2]. Incentives and evaluator roles are also used to push clients to act honestly [3]. A main risk is the Byzantine client. A bad client can send fake updates and hurt the global model. Many papers study robust aggregation or filtering to reduce this damage [4], [5], [6], [7]. These defenses are important when clients are open or weakly controlled.

In real federated learning deployments, missing accountability can cause practical harm. In healthcare-oriented FL systems, poisoned or tampered updates can affect sensitive decision pipelines, and later investigation becomes difficult if the system cannot verify which participant submitted the suspicious update [1], [2]. Similar risks also appear in application settings such as cellular traffic prediction and large-scale IoT, where abnormal or malicious updates can degrade the shared model and affect downstream system behavior [8], [9]. Robust aggregation can reduce the influence of abnormal updates, but it does not by itself provide clear evidence about which client was responsible [4], [5], [10]. As a result, a system may remain partly robust at the model level while still lacking the traceability needed for investigation, compliance, and post-incident response.

Robustness is also needed when training is asynchronous or large-scale. Some works target serverless or low-trust settings [11]. Others study robust FL under asynchronous updates and real traffic prediction tasks [8]. Large IoT settings also need scalable defenses [9]. Another challenge is system heterogeneity. Clients can have different compute, bandwidth, and data distributions. Split and adaptive designs try to reduce this gap [12], [13]. Multi-task and clustered FL also help when clients are not similar [14], [15]. Personalization methods reduce the "one model fits all" problem [16].

Auditing and accountability are now common goals. Some designs add a third-party auditor to reduce burden on the server and clients [17]. Some works combine robustness with auditable steps during training [10]. IDS-style detection is also surveyed as a way to spot malicious clients [18]. Blockchain is often used when we want tamper-resistant logs and public verification. Some frameworks use blockchain for verifiable training and traceable updates [19]. Some focus on credibility and fairness of participants and results [20]. Related security work also studies stronger authentication and collusion or free-riding behaviors [21], [22].

B. Problem Statement

FL needs three things at the same time: privacy, robustness, and traceability. Privacy alone is not enough if poisoned updates can still pass [4], [5]. Robust defenses can be costly or hard to verify later [7], [11]. So the system can fail silently. Auditing helps, but it also adds new design constraints. Logs must be correct, complete, and easy to check. Some works use structured auditing or integrity checks [1], [2]. Some push the auditing role to a third party to reduce load [17]. If blockchain is used, we must also manage cost and latency. On-chain storage is expensive if we store everything. Verification must still be strong even with limited on-chain data [19], [20]. Collusion and free-riding also remain risks [22].

C. Contributions

1) Orthogonal audit layer: We propose RollupFL as an audit layer for FL. It works with different aggregation rules. It does not change how aggregation is computed.

*Corresponding author.

2) Two audit designs: We study simple logging audit. We also study blockchain-based audit. Logging is fast but weaker if the server is not trusted. Blockchain audit is stronger but adds more delay.

3) Evaluation under byzantine attacks: We test the clean setting and attacked settings. We use sign-flip and model-replacement attacks. We vary the malicious client fraction from 10% to 30%. We compare FedAvg, Median, and Krum with and without auditing.

4) Metrics for robustness, cost, and accountability: We measure test accuracy and convergence across rounds. We measure ASR for model-replacement. We measure attacker attribution and tamper detection. We measure per-round latency and storage growth.

II. Related Work

This section reviews prior studies on robust and auditable federated learning and highlights the main design choices that shape accountability in these systems.Table I shows that prior work in secure federated learning differs not only by technique, but also by system assumptions and accountability design. Some methods mainly improve robustness under a centralized trusted server, some add auditing through trusted logging or third-party verification, and others reduce trust by using blockchain-backed traceability or decentralized verification.

Although prior work has improved the security of federated learning, several practical gaps remain. Methods based on robust aggregation can reduce the effect of malicious updates, but they usually do not support clear attacker attribution or post-hoc traceability [4], [5], [6], [7]. Some auditable FL approaches improve verification, yet they often depend on a trusted server, coordinator, or external auditor [1], [2], [10], [17]. Blockchain-based solutions strengthen integrity and tamper resistance, but they can introduce extra latency, storage cost, and coordination overhead [19], [20]. In addition, some existing methods are harder to scale or deploy in realistic FL settings with heterogeneous clients, dynamic participation, and large communication cost [8], [9], [11].

A. Robust Aggregation in Federated Learning

Many robust FL papers focus on aggregation rules and filtering. They try to reduce the effect of poisoned or extreme updates [4], [5]. The goal is to keep accuracy stable even when some clients are malicious. Some frameworks use defense at both server and client sides. This can improve robustness but it can add overhead [6], [7]. Other work moves away from a single trusted server and still aims for Byzantine robustness [11]. Robust FL is also studied in specific networked settings. Asynchronous training needs extra care because stale updates can mix with attacks [8]. Large IoT systems also push for scalable robust learning designs [9]. This line of work also includes compression or low-dimensional update ideas. These can reduce leakage and may help detect strange updates [23].

B. Auditing, Accountability, and Trusted Logging

Auditing in FL often means we can prove that key steps happened. Some designs support auditable training flows for sensitive domains [1]. Some add data integrity auditing to detect tampering [2]. Some work combines robustness with auditable evidence. This helps when we want both defense and later accountability [10]. Another idea is to rely on a third-party auditor to reduce burden on the server [17]. Accountability can also come from incentives and access control. Evaluator-based incentives try to reward good behavior [3]. Authentication schemes help reduce fake identities in distributed learning [21]. IDS-based detection is also surveyed for spotting malicious clients [18]. Audit logs are also useful for studying collusion and free-riding. Correlation-based analysis is one approach to detect suspicious behavior patterns [22].

C. Blockchain-Based Verification (in FL/security)

Blockchain is used when we want tamper-resistant records. Some FL frameworks use blockchain to support verifiable training and traceable updates [19]. Others focus on credibility, fairness, and evidence for participation [20]. Even with blockchain, security issues remain. Collusion and free-riding can still happen and need detection methods [22]. Strong authentication is also relevant when identities matter [21].

III. System Model and Threat Model

This section defines the system model and the threat model used in this work. We describe the federated learning (FL) setting, the Byzantine client behavior, and the attack cases we evaluate. We also state the audit goals and the limits of auditing. This helps explain why we add an audit layer in Section IV.

A. Federated Learning Setup

We consider a synchronous FL system with one server and n clients. Training runs in rounds. In round t, the server holds the global model θ_t. The server selects a subset of clients $C_t \subseteq \{1, \ldots, n\}$ and sends θ_t to them. Each selected client $c \in C_t$ trains locally on its private dataset D_c. The client then computes a local update Δ_t^c and sends it to the server. The server aggregates the received updates using an aggregation function $\mathrm{Agg}(\cdot)$ and produces the next model θ_{t+1}.

For simplicity, we write the global update rule as:

$$\theta_{t+1} = \theta_t + \mathrm{Agg}\left(\{\Delta_t^c\}_{c \in C_t}\right). \tag{1}$$

This form covers common methods. For example, FedAvg uses averaging, while robust rules like Median and Krum aim to reduce the influence of abnormal updates. Clients do not communicate with each other. The system follows a star topology. This is a common FL deployment model.

We assume client data is non-IID. This means different clients may have different label proportions and feature patterns. This makes the setting realistic. It also matters for security. Honest updates can differ widely due to data skew. So simple outlier detection becomes harder. This is why robust aggregation and audit are both relevant.

TABLE I. TAXONOMY OF RELATED WORKS BY SYSTEM ASSUMPTIONS, DEFENSE DESIGN, AND ACCOUNTABILITY MODEL

Work	Topology	Trust Model	Defense Type	Audit Type	Attack Focus	Main Limitation
[4], [5]	Centralized	Trusted server	Robust aggregation and filtering	None	Byzantine poisoning	Improves robustness, but does not provide attribution or tamper-evident audit records.
[6], [7]	Centralized	Trusted server	Server-side and client-side defense	Limited or none	Poisoning and abnormal updates	Can improve robustness, but adds system overhead and still gives limited accountability.
[8], [9]	Centralized	Trusted server	Robust FL for asynchronous or large-scale settings	None	Byzantine behavior, stale updates, and large-scale update issues	Focuses on robustness under deployment constraints, but not on post-hoc traceability.
[11]	Decentralized or serverless	Low-trust	Byzantine-robust training	None	Byzantine clients	Reduces reliance on one server, but does not directly solve auditability and attribution.
[1], [2]	Centralized	Trusted server	Secure and privacy-preserving FL with integrity support	Structured integrity audit	Tampering and integrity violations	Supports checking and integrity, but has limited focus on Byzantine attacker attribution during FL.
[10]	Centralized	Trusted or semi-trusted coordinator	Robust FL with auditable evidence	Training-step audit	Byzantine behavior	Moves toward robustness and audit together, but increases design complexity and verification cost.
[17]	Centralized	Third-party auditor	External verification support	Third-party audit	Integrity and compliance	Reduces burden on the server, but depends on an additional trusted entity.
[19], [20]	Blockchain-assisted	Trustless or low-trust	Verifiable training, credibility, and fairness support	Blockchain audit	Tampering and dishonest participation	Gives stronger integrity and traceability, but adds latency, storage, and coordination cost.
[21], [22]	Distributed	Identity-aware and access-controlled	Authentication and suspicious behavior analysis	Event or behavior logging	Fake identities, collusion, and free-riding	Helps with identity and misuse detection, but does not fully link update-level evidence to aggregation outputs.

B. Byzantine Client Model

A subset of clients may be Byzantine. These clients deviate from the protocol. They are controlled by an adversary and may coordinate across rounds. We assume Byzantine clients can craft arbitrary updates. So a Byzantine client can send any vector in $\mathbb{R}^d$, not only a true gradient. This captures many poisoning behaviors, including random noise, sign changes, scaling, and targeted backdoor updates.

We use β to denote the Byzantine fraction:

$$\beta = \frac{\#\text{Byzantine clients}}{n}. \tag{2}$$

We evaluate multiple values of β in later experiments. Larger β usually makes training harder. It also weakens robust aggregation rules. In many distributed settings, one-third is an important boundary under common assumptions. In practice, model quality can degrade even below that boundary when data is non-IID and attacks are adaptive.

We assume standard cryptographic hardness. The adversary cannot break SHA-256 or ECDSA-256. This is needed for the audit mechanisms described later. We do not focus on privacy attacks such as full model inversion. Those attacks are important but they are not the main target of this study. Our focus is Byzantine manipulation and post-hoc accountability.

C. Attack Scenarios

We study two Byzantine attack scenarios. The first is a sign-flip attack. It is simple and non-adaptive. A Byzantine client flips the direction of its update by multiplying it by -1:

$$\Delta_t^{\text{Byz}} = -\Delta_t^{\text{Hon}}. \tag{3}$$

This attack pushes the global model away from the descent direction. Under FedAvg, flipped updates can cancel honest progress. This can slow convergence or cause divergence when the Byzantine fraction is high. Robust aggregation can reduce the impact because flipped updates often appear as outliers relative to honest updates.

The second scenario is a model-replacement (targeted poisoning) attack. This attack aims to insert a backdoor. The attacker crafts an update that optimizes a target loss on an attacker-chosen dataset D_{target}. The attacker also amplifies the update using a scaling factor $\alpha > 0$:

$$\Delta_t^{\text{Byz}} = \alpha \nabla \mathcal{L}_{\text{target}}(\theta_t; D_{\text{target}}). \tag{4}$$

This attack can be stealthy. Overall test accuracy may remain acceptable, while the model fails on a specific trigger or class. Under FedAvg, amplified targeted updates can dominate the mean. Under Median or Krum, these updates may be filtered, but success depends on β, the amplification strength, and the natural variance created by non-IID data.

These two attacks cover two different goals. Sign-flip aims to reduce utility directly. Model-replacement aims to keep utility high but add hidden malicious behavior. In both cases, the audit layer records update evidence regardless of whether the attack succeeds.

D. Assumptions and Audit Goals

We assume secure client–server communication using authenticated encryption. This protects updates in transit. We also assume clients have stable identities, so an update can be linked to a client. We assume timestamps are meaningful under small clock drift. We also assume auditing is offline. The auditor verifies logs after training completes. This reduces runtime overhead. It also means audit is not a real-time defense.

For blockchain-based audit, we assume client signing keys are not compromised. If a private key is stolen, an attacker can sign updates as that client. This would break non-repudiation. Key compromise is important in practice, but it is outside the scope of our evaluation.

The audit layer targets three goals. First, it should support tamper detection. If a log entry is modified after it is recorded, the auditor should detect it. Second, it should support attacker attribution. After training, the auditor should identify which clients sent malicious updates with high confidence. Third, for blockchain audit, it should provide non-repudiation. A client should not be able to deny an update that it signed.

1) Threat summary and link to proposed method: The main risk in this system is that Byzantine clients can send malicious updates that harm training or insert a backdoor. Robust aggregation helps reduce the impact of these updates during training, but it does not explain who attacked. Auditing can record evidence and support post-hoc investigation, but it does not stop poisoning by itself. This motivates an orthogonal design. In Section IV, we introduce an audit layer that can be combined with any aggregation rule. This keeps the defense logic and the accountability logic separate, while allowing them to work together in the same FL pipeline.

IV. Proposed Method: Audit Layer for Federated Learning

Byzantine clients can harm federated learning by sending wrong model updates. Robust aggregation methods, such as Median and Krum, can reduce the impact of these attacks. However, they do not provide accountability. They may block the attack, but they do not clearly tell which client acted maliciously. This is a problem in real systems where we also need traceability, compliance, and investigation support.

To address this gap, we propose *RollupFL*. RollupFL adds an audit layer to federated learning. The audit layer records client updates in a verifiable way. This record can be checked later. So the system can identify suspicious clients and detect tampering after training. The audit layer is designed to work with any aggregation rule. So defense and accountability can be selected independently. We implement two audit options. The first is a simple logging audit. It is light and fast, but it assumes the server is trusted. The second is a blockchain-based audit. It gives stronger integrity and attribution, but it costs more latency.

A. Audit Layer Overview

The main idea is that robustness and accountability are different goals. Robust aggregation improves safety during training. It reduces the effect of poisoned updates. However, it does not keep a strong record of who sent what. Audit logging improves accountability. It helps trace actions back to clients. However, it does not stop poisoning by itself. RollupFL separates these two parts. The aggregation rule still decides how the global model is updated. The audit layer only records evidence about the received updates.

1) Design Principles: RollupFL follows three design principles.

a) Orthogonality: The audit layer must work with any aggregation rule. So the same auditing logic can be used with FedAvg, Median, or Krum.

b) Minimality: The audit operations should not add heavy computation. The goal is to keep the audit cost low when possible.

c) Immutability: The audit trail must resist tampering. If someone changes a stored update, the system should detect it. Blockchain-based audit gives stronger proof. Simple logging gives weaker proof yet lower cost.

2) Audit workflow: The workflow has three main stages, as shown in Fig. 1.

a) Update collection: Each client sends a local update to the server. The audit layer captures this update immediately. It also stores the metadata and proof material needed to bind each submitted update to its round, client identity, and later verification result.

b) Aggregation: The server applies the chosen aggregation rule. The audit layer does not change how aggregation is computed. This keeps the design modular.

c) Verification and attribution: After training, an auditor can verify the audit trail. The auditor checks whether the trail was modified. The auditor can also link suspicious updates to specific clients. This supports post-hoc investigation.

B. Simple Logging Audit

Simple logging is the lightweight audit option. The server records each update in a centralized log. A hash is computed for integrity checks. This method is suitable when the server is trusted. It is also useful when the budget is limited.

1) Mechanism: For each client update, the server computes a hash. The hash binds the client identity and the update content. A typical log record stores the client ID, timestamp, update hash, and the update itself. To strengthen integrity, the server can also chain hashes across rounds. This creates a linked structure. If an entry is modified later, the chain consistency breaks.

2) Tamper detection: Tamper detection is done by recomputing hashes. The auditor recomputes the hash from stored data. If the computed value does not match the stored hash, then tampering is detected. This approach is simple. However, it depends on the log being available and trustworthy. If the server can rewrite logs, the method becomes weaker.

3) Attacker attribution: Attribution under simple logging is mostly heuristic. The log tells which client sent which update. Then the auditor checks patterns that look malicious. For example, sign-flip updates often show reversed gradient directions. Model-replacement updates may show unusual magnitude or direction. This can help identify attackers. However it is not a cryptographic proof. So the confidence may be lower than blockchain audit.

4) Overhead: The overhead of simple logging is small. Hashing and storing data is fast compared to blockchain consensus. Storage grows with the number of rounds and clients. This is expected because each update generates one record.

C. Blockchain-Based Audit

Blockchain audit is designed for stronger guarantees. It is useful when the server is not fully trusted. It also helps in multi-party environments where independent verification is needed. In this method, updates are signed and recorded in an immutable ledger.

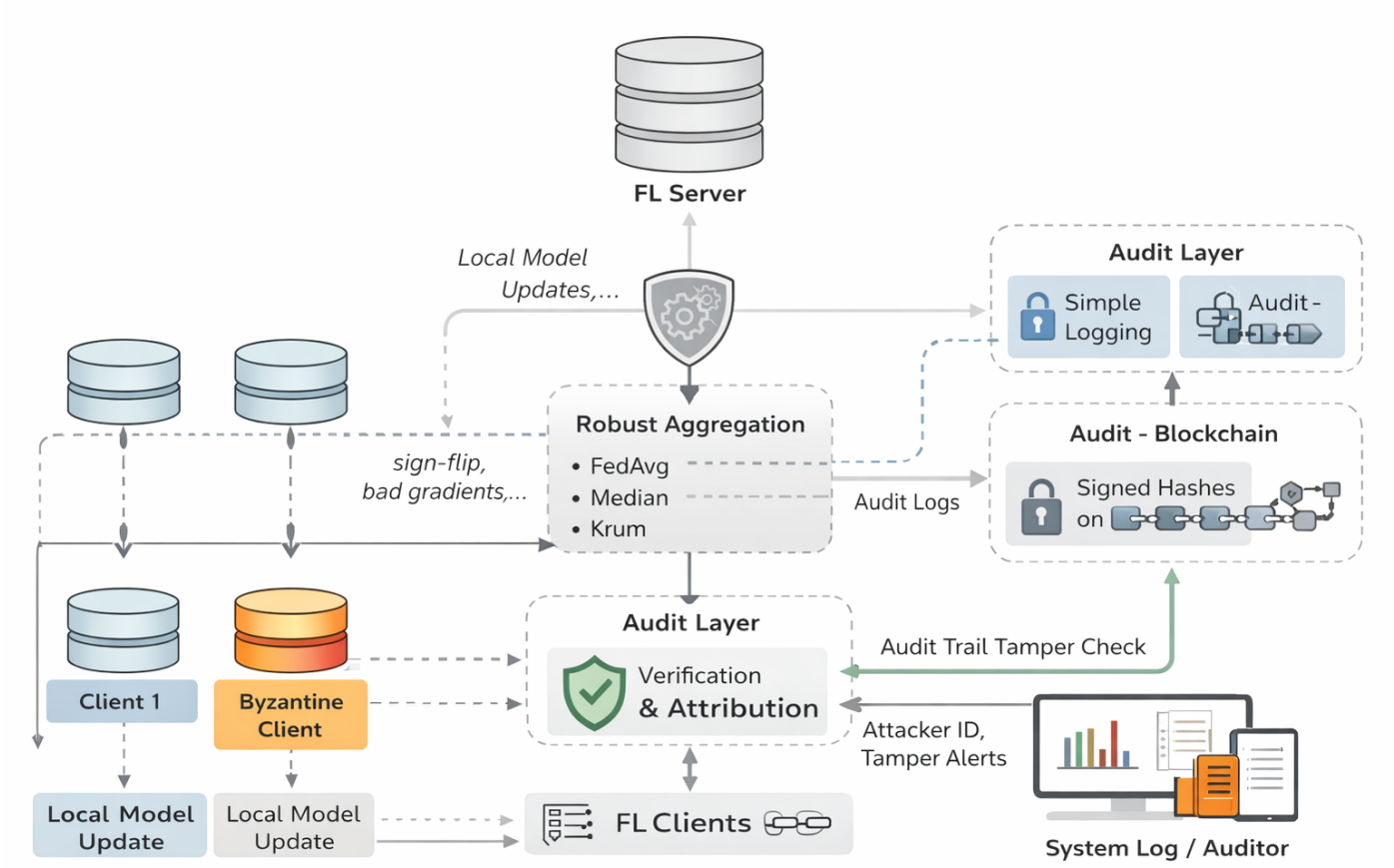

Fig. 1. RollupFL architecture with parallel auditing: Clients send updates, the server aggregates (FedAvg/Median/Krum), and logs or blockchain records enable later verification and accountability.

1) Mechanism: Each client signs its update using a private key. The server verifies the signature using the client public key. If the signature is valid, the server records the update evidence in the ledger. The ledger stores a hash of the update and the signature. This binds the update to the client identity. It also prevents denial later.

2) Tamper detection: Tamper detection becomes stronger in this setting. If someone changes an update, the signature verification fails. So integrity checks are direct and reliable. The ledger history also prevents silent rewriting of records.

3) Attacker attribution: Attribution is also stronger. The client signature provides non-repudiation. So the system can prove that a specific client signed a specific update. This is important for accountability and auditing.

4) Overhead: Blockchain audit adds more cost. Signing and verification add computation per update. Ledger writing adds network and consensus delay. So the round latency increases. This is the trade-off for stronger guarantees.

D. Integration with Aggregation (FedAvg/Median/Krum)

RollupFL is designed to integrate without changing the aggregation rule. The server receives client updates for each round. The audit layer records evidence for each update. Then the server runs the aggregation rule. This can be FedAvg, Median, or Krum. So the final output has two parts. The first part is the updated global model. The second part is the audit trail for later verification.

To make this link explicit, let client $c \in C_t$ submit update Δ_t^c in round t. For each received update, the audit layer creates a record:

$$e_t^c = \big(c,\ t,\ \tau_t^c,\ h(\Delta_t^c),\ \pi_t^c\big),$$

where, τ_t^c is the timestamp, $h(\cdot)$ is a cryptographic hash, and π_t^c is the audit proof. In simple logging, π_t^c can be empty or server-generated metadata. In blockchain audit, π_t^c is the client signature and ledger reference. The round audit set is then:

$$A_t = \{e_t^c\}_{c \in C_t}.$$

The server computes the next global model from the submitted updates as:

$$\theta_{t+1} = \theta_t + \mathrm{Agg}\big(\{\Delta_t^c\}_{c \in C_t}\big).$$

To bind the audit trail to the training result of that round, RollupFL stores a round commitment:

$$H_t = H\big(H_{t-1} \,\|\, h(A_t) \,\|\, h(\theta_{t+1})\big),$$

where, $H(\cdot)$ denotes a hash-combination function and H_{t-1} is the previous round commitment. This means the audit record is linked not only to each client update, but also to the aggregation output produced from the set of submitted updates.

1) FedAvg integration: With FedAvg, the server averages client updates. Audit does not change averaging. It only records what was received. FedAvg can still be vulnerable under strong Byzantine attacks. However, audit improves accountability after the fact.

2) Median integration: With Median, the server computes a coordinate-wise median. This reduces the effect of outliers. Audit still records the raw submitted updates. So the system can later identify which clients were outliers.

3) Krum integration: With Krum, the server selects the most central update. This method can tolerate Byzantine clients under standard assumptions. Audit again records the submitted updates and metadata. So suspicious updates can be traced later.

E. Why Audit Does Not Reduce Attack Success?

It is important to state the correct role of auditing. Auditing is not a defense method. It does not block malicious updates in real time. It records evidence and supports investigation. So metrics like ASR are mainly affected by aggregation. Audit changes accountability metrics, not defense metrics. This separation is intentional in RollupFL.

V. Experimental Setup

This section explains how we evaluate RollupFL. We test it under different Byzantine attack settings. We also use a consistent and repeatable experimental process. We describe the dataset, the model, and how data is split across clients. We then explain the training setup, the attacker settings, and the attack types. Next, we list the baselines and the audit variants we compare against. Finally, we define the metrics we report and how we compute confidence intervals.

A. Dataset and Model

We use the MNIST dataset in all experiments. MNIST is a common benchmark for federated learning studies. It is simple compared to modern vision datasets, however it is useful for controlled tests. It lets us study attack behavior and audit behavior without many extra factors. Each example is a 28×28 grayscale image. Pixel values are normalized into the range $[0, 1]$. The training set has 60,000 images and the test set has 10,000 images.

We train a small convolutional neural network (CNN). The network has two convolution layers and one fully connected layer. The first convolution uses 32 filters with a 5×5 kernel and ReLU. A 2×2 max-pooling layer follows it. The second convolution uses 64 filters with a 5×5 kernel and ReLU. It is also followed by 2×2 max-pooling. After that, we use a fully connected layer with 128 units and ReLU. The output layer has 10 units with softmax for digit classification. This model reaches about 99%+ accuracy in centralized training. The model size is about 100 KB, which is small enough for federated updates.

1) Non-IID client data: We use a non-IID split to represent realistic client behavior. We generate client splits using a Dirichlet distribution with concentration $\alpha = 0.5$. This setting creates heterogeneous clients. Each client tends to have more samples from a small set of digits. This makes training harder than IID splits and is closer to real federated settings. We split the 60,000 training samples across 50 clients. Each client has about 1,200 samples on average.

B. Training Configuration

We simulate synchronous federated learning. In each round, the server selects a subset of clients. Those clients train locally and then send updates back to the server. The server aggregates the updates and produces the next global model. We repeat this for a fixed number of rounds.

We use 50 total clients. In each round, 20 clients participate. Clients are selected uniformly at random. This gives a 40% participation rate per round. Each selected client trains for 1 local epoch using SGD. We use a constant learning rate of 0.01. We use batch size 32. We run 20 communication rounds in total. These values keep the setup simple and stable. They also allow attacks to show their effect within a short training horizon.

These hyperparameter choices are intended for a controlled MNIST benchmark rather than aggressive optimization. A learning rate of 0.01 with one local epoch gives stable convergence for the small CNN and helps avoid large client-side drift under the non-IID split. A 40% participation rate provides enough diversity in each round while still reflecting partial client availability in federated learning. We use 20 communication rounds to keep the study short and repeatable, while still allowing Byzantine effects and audit behavior to become visible. Overall, these settings balance stability, comparability, and attack observability in a simple experimental setup.

1) Byzantine clients: We choose Byzantine clients at the start of the training. The same clients remain Byzantine for all 20 rounds. This keeps the attack condition consistent. We evaluate three attacker ratios. They are 10%, 20%, and 30%. With 50 clients, this equals 5, 10, and 15 Byzantine clients. The 10% case is challenging however often survivable. The 30% case is more extreme and stresses the aggregation rules.

In the current evaluation, Byzantine clients are selected at the beginning of training and remain Byzantine for all rounds. This static adversary setting is used to provide a controlled and repeatable comparison across aggregation and audit configurations. However, real federated learning deployments may also include dynamic adversaries, client churn, and time-varying malicious behavior. These cases are important in practice, but they are not covered in the present experiments and should be studied in the future work.

2) Attack strategies: We implement two types of Byzantine behavior.

The first attack is sign-flip. A Byzantine client multiplies its gradient (or update) by -1 before sending it. This is a simple and non-adaptive attack. It does not require knowledge of other client updates. It tests whether the aggregation method can tolerate inverted signals.

The second attack is model-replacement. This is stronger and more targeted. The attacker crafts an update that pushes the global model toward an adversarial objective. In our experiments, the attacker targets a chosen class (digit "0").

We represent the crafted update using a scaled gradient of an adversarial loss:

$$\delta = \alpha \cdot \nabla \mathcal{L}(\theta_t; D_{\text{target}}),$$

where, D_{target} is attacker-controlled data and $\alpha > 0$ amplifies the effect. This attack aims to create a backdoor-like behavior. It tries to force misclassification for the target class. This is closer to practical poisoning and trojan-style attacks.

C. Baselines and Compared Methods

We compare standard federated learning baselines and audit-enabled variants. The goal is to separate two roles. The first role is robust learning under attack. The second role is accountability after the fact. Robust aggregation mainly affects learning robustness. Audit mainly affects traceability and evidence.

1) FL baselines (no audit): We include FedAvg as a standard baseline. FedAvg performs simple averaging of client updates. It is widely used, however it is known to be vulnerable to Byzantine behavior.We also include Median. Median computes the coordinate-wise median of client updates. This can reduce the effect of strong outliers. It is a common robust aggregation choice. We also include Krum. Krum selects one update that is closest to others in terms of distance.It tries to filter outliers by geometry.It is also a known Byzantine-robust method.

2) Audit variants (orthogonal layer): We add auditing on top of the aggregation rules. We test two audit types: 1) simple logging and 2) blockchain audit. Audit+FedAvg adds blockchain audit to FedAvg. This does not make FedAvg robust by itself. It provides stronger evidence and traceability. Audit+Median adds blockchain audit to Median.This combines robustness from Median with stronger audit guarantees.Log+FedAvg adds simple logging to FedAvg. It is cheaper than blockchain even so depends more on server trust. Log+Median adds simple logging to Median. It combines robust aggregation with low-cost audit. These variants are designed to be modular. The audit layer can be applied to different aggregators. This makes the design easy to extend.

D. Evaluation Metrics

We report metrics for robustness, accountability, and cost. We group them into three sets.

1) Defense metrics: We measure final test accuracy after 20 rounds. We compute accuracy on the MNIST test set (10,000 images). Lower accuracy usually means the attack harmed training. We also measure attack success rate (ASR) for model-replacement. ASR measures how often the attack succeeds at triggering the target behavior. In our case, the target relates to digit "0". Higher ASR means the attacker is more successful. Lower ASR means stronger resistance.

2) Audit metrics: We measure attacker attribution rate. This is the fraction of Byzantine clients that can be identified from audit records. For logging, this depends on stored hashes and server-side verification. For blockchain, this uses signatures and immutable proofs. We also measure tamper detection rate. This is the fraction of malicious updates detected during verification.It measures whether the audit trail can reveal manipulation or suspicious updates. Higher values are better.

3) Overhead metrics: We measure round latency in milliseconds. This includes local computation, communication, aggregation, and audit work. We report both the raw time and multipliers relative to the no-audit baseline.We also track storage growth for blockchain audit.Storage grows roughly linearly with the number of rounds because each round adds audit data.

4) Statistical methodology: We repeat each experiment using $N = 10$ random seeds. Each seed changes model initialization and data shuffling. For all metrics, we report the mean and a 95% confidence interval (CI). We compute CI as $1.96 \times \text{SEM}$, where $\text{SEM} = \text{SD}/\sqrt{N}$. Here SD is the standard deviation across seeds. We use CI overlap as a simple significance indicator. If 95% CIs do not overlap, we treat the difference as significant at $\alpha = 0.05$. If they overlap, we report the difference however we avoid strong claims.

VI. Results and Discussion

A. Clean Baseline Results (0% Byzantine)

We first evaluate the clean setting where no malicious clients are present. This test checks normal training behavior and helps confirm that the audit layer does not reduce model utility when the system is benign. FedAvg, Median, and Krum reach similar accuracy in this setting, and the audit-enabled variants remain close to the baseline. The small error bars indicate stable results across seeds. Overall, the clean baseline suggests that auditing does not harm accuracy under normal conditions (Fig. 2).

B. Robustness Under Byzantine Attacks

We then evaluate robustness under Byzantine clients using sign-flip and model-replacement attacks. We vary the Byzantine fraction from 10% to 30%. As the attacker fraction increases, accuracy decreases for all methods. FedAvg degrades the most and becomes unstable at higher Byzantine levels. Median and Krum remain more stable and preserve higher accuracy. This shows that robust aggregation reduces the impact of malicious updates (Fig. 3).

C. Convergence Behavior Across Rounds

We also analyze convergence by tracking test accuracy across communication rounds. In the clean setting, all methods

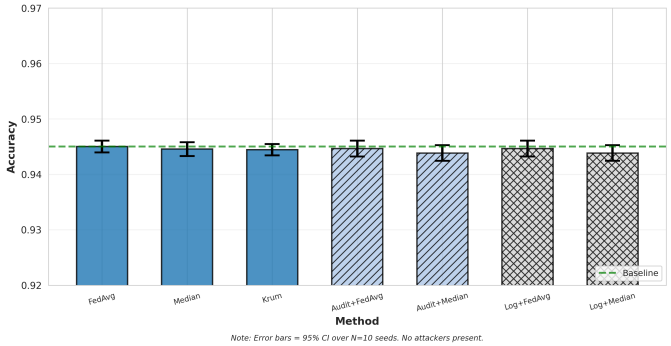

Fig. 2. Sanity check in the clean setting (0% Byzantine). FedAvg, Median, and Krum show similar baseline accuracy. Audit-enabled variants remain close to the baseline. Error bars show 95% confidence intervals over seeds.

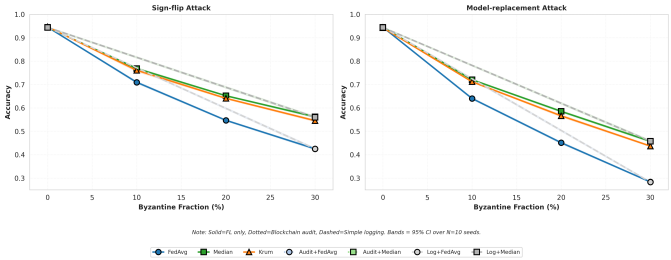

Fig. 3. Accuracy under Byzantine attacks as the malicious client fraction increases. Robust methods (Median, Krum) degrade more slowly than FedAvg under both sign-flip and model-replacement. Error bars show 95% confidence intervals over seeds.

converge smoothly. Under model-replacement attack at high Byzantine fraction, FedAvg fails to converge and collapses early. Median and Krum remain stable and preserve learning progress. These trends match the final accuracy results and show that robust aggregation improves stability under attack (Fig. 4).

D. Attack Success Rate (ASR)

Accuracy does not fully describe security impact, so we also measure attack success rate (ASR) for model-replacement. Higher ASR means the attacker succeeds more often. FedAvg exhibits high ASR under attack. Median and Krum reduce ASR compared to FedAvg. The audit layer does not reduce ASR by itself, which is expected because auditing provides accountability rather than attack prevention (Fig. 5).

E. Audit Equivalence and Measured Overhead

We report two practical questions: 1) does auditing change accuracy, and 2) what is the runtime overhead. Fig. 7 (left) summarizes accuracy differences using a paired comparison and an equivalence margin. The accuracy impact remains close to zero. Fig. 7 (right) reports measured per-round latency. Logging adds a small overhead, while blockchain audit adds a larger overhead due to verification and integrity operations. Table II lists measured latency values.

F. Audit Layer Comparison (Blockchain vs. Logging)

We compare blockchain audit against simple logging. We evaluate accuracy at 10% and 30% Byzantine fractions for

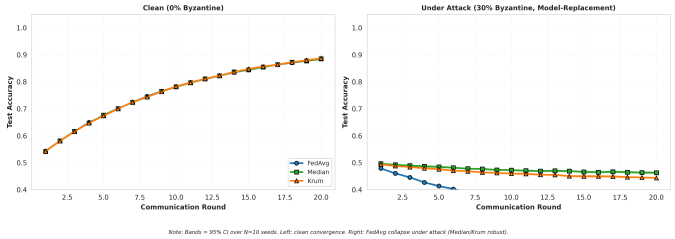

Fig. 4. Convergence across rounds in clean and attacked settings. Under strong attack, FedAvg collapses, while Median and Krum remain stable.

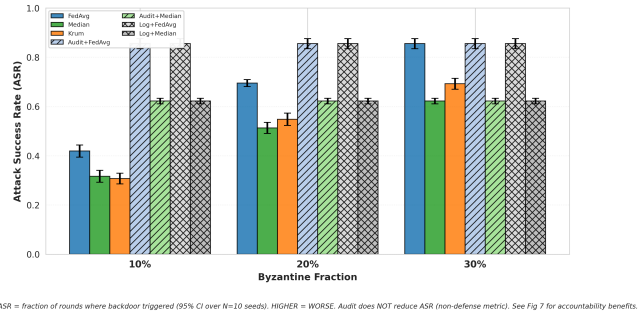

Fig. 5. Attack success rate (ASR) under model-replacement. Robust aggregation reduces ASR compared to FedAvg. Audit variants track their underlying aggregation method.

TABLE II. PER-ROUND LATENCY OVERHEAD (MS)

Aggregation	Audit	Mean	Std
FedAvg	None	44.42	5.50
	Logging	49.99	0.86
	Blockchain	136.68	7.84
Median	None	44.44	2.54
	Logging	50.08	0.99
	Blockchain	134.43	7.26
Krum	None	46.98	7.90
	Logging	50.51	0.83
	Blockchain	135.11	6.44

FedAvg, Median, and Krum. The results show that both audit options preserve accuracy close to the FL-only baseline. Differences are small and remain within the confidence intervals. This suggests that audit instrumentation is largely orthogonal to model training (Fig. 6).

G. Accountability Metrics

The main benefit of auditing is accountability. We measure attacker attribution and tamper detection. Blockchain audit provides stronger attribution and stronger tamper detection than simple logging. This is expected because blockchain offers integrity proofs and non-repudiation properties. Logging can be useful in trusted environments due to lower overhead, except it provides weaker guarantees. These results are shown in Fig. 8 and summarized in Table III.

H. Summary

Across all experiments, robust aggregation (Median, Krum) improves resilience against Byzantine attacks compared to

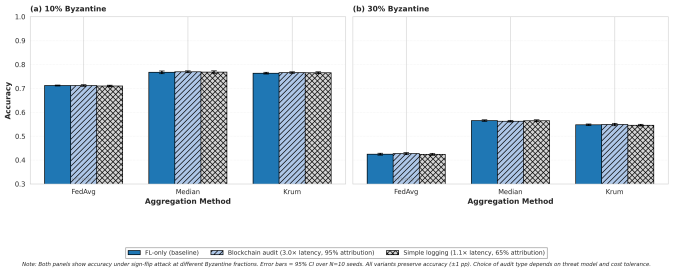

Fig. 6. Accuracy comparison between FL-only, logging audit, and blockchain audit at 10% and 30% Byzantine fractions. Results are shown for FedAvg, Median, and Krum with 95% confidence intervals over seeds.

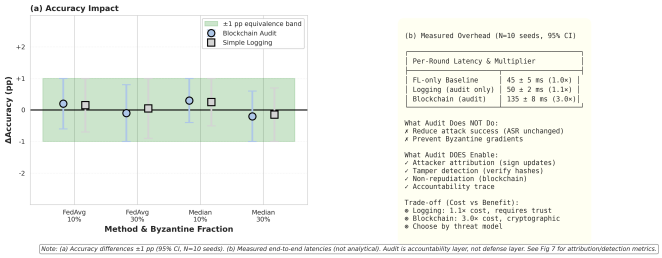

Fig. 7. Audit accuracy impact and measured overhead. Left: paired accuracy difference with an equivalence band. Right: measured per-round latency for baseline, logging, and blockchain audit. Error bars show 95% confidence intervals.

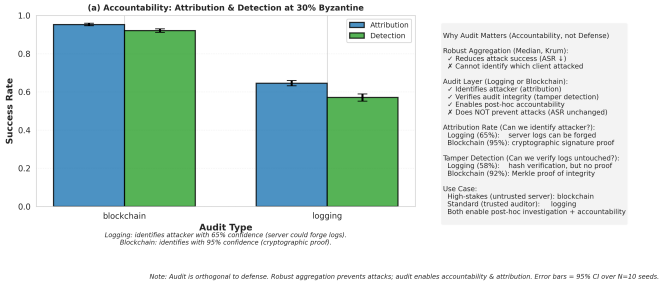

Fig. 8. Accountability benefits at 30% Byzantine. Blockchain audit improves attacker attribution and tamper detection compared to simple logging.

FedAvg (Fig. 3 to Fig. 4). Auditing does not act as a defense mechanism and therefore does not reduce ASR by itself (Fig. 5). Instead, auditing improves accountability. Logging offers low overhead, while blockchain provides stronger attribution and tamper detection at higher runtime cost (Fig. 7, Fig. 8, Table II, Table III).

VII. Conclusion and Future Work

This study presented *RollupFL*, an audit layer that can be added to federated learning (FL) systems to improve accountability under Byzantine attacks. In many FL deployments, robust aggregation methods such as Median or Krum are used to reduce poisoning damage. These methods help protect the global model, yet they do not clearly answer a key question: *which client attacked?* In practice, this matters because real systems often need traceability, investigation, and compliance. RollupFL addresses this gap by separating two concerns. The aggregation rule focuses on defense during training. The audit

TABLE III. Accountability Metrics at 30% Byzantine Fraction

Audit Type	Attribution	Tamper Detection
Logging	0.65	0.58
Blockchain	0.95	0.92

layer focuses on evidence after training. RollupFL records client updates in a way that can be verified later. We study two audit designs. The first design is simple logging, which is lightweight and fast, however it assumes the server is trusted to keep correct logs. The second design is blockchain-based audit, which provides stronger integrity and non-repudiation using signatures and an immutable ledger, however it adds more delay. Our experiments on MNIST under sign-flip and model-replacement attacks (10–30% Byzantine clients) show that the audit layer does not meaningfully change model accuracy, and accuracy changes stay within a small range across seeds. We also find clear differences in accountability. Simple logging provides moderate attacker attribution and tamper detection, while blockchain audit provides much stronger attribution and tamper detection. We further observe that the main cost of blockchain audit comes from ledger writing latency rather than signature computation. Overall, these results support a practical message: robust aggregation helps reduce attack impact, and auditing helps explain and prove what happened. RollupFL makes it easy to combine these two benefits without changing the aggregation logic. Another limitation is that the experiments use a static Byzantine setting, where malicious clients remain fixed across rounds, while dynamic adversaries and client churn are left for future study.

RollupFL also has several limitations under adversarial system assumptions. First, its accountability guarantees depend on stable client identities and secure client–server communication. If identity management is weak, attribution becomes less reliable. Second, in blockchain mode, non-repudiation depends on correct key management. If a client signing key is compromised, an attacker can submit updates under that identity. Third, the current design assumes the audit trail is checked after training, so it supports post-hoc accountability rather than real-time prevention. Finally, the present evaluation is limited to MNIST, 50 clients, 20 communication rounds, and a static Byzantine setting in which malicious clients remain fixed across rounds. These choices support a controlled study, but they do not yet capture the full scale and dynamics of real federated learning deployments.

There are several useful directions for future work. First, RollupFL should be tested at larger scale. Real FL systems can involve thousands or more clients. In that setting, blockchain writing and storage can become the main bottleneck. Future work can reduce this cost using batching, sharding, or off-chain aggregation with periodic on-chain commitments. Second, stronger attacker models should be studied. Some attackers can be adaptive. They may change behavior over time and try to avoid detection. It would be useful to evaluate RollupFL under adaptive and coordinated strategies and to study whether audit signals can help trigger defenses, such as switching aggregation rules or isolating suspicious clients. Third, privacy should be explored together with accountability. In this study, the audit goal is integrity and traceability, not privacy protection. Future work can combine RollupFL with

privacy tools such as differential privacy or secure aggregation, and then study how much accountability can still be preserved under stronger privacy constraints. Fourth, evaluation on more complex datasets and models is important. MNIST is useful for controlled tests, yet modern systems use datasets such as CIFAR-10 and ImageNet, and they may use language models. These settings also have different compute and network costs, so the audit overhead should be measured under realistic conditions. Fifth, audit outputs can support incentives. For example, audit evidence could help reward honest clients and penalize malicious ones in decentralized settings. Another important direction is to evaluate RollupFL under stronger and more diverse Byzantine behaviors. The current study uses representative attack settings, but future work should vary the strength of sign-flip attacks, the scaling factor α in model-replacement attacks, and additional attack types such as random-noise or label-flip poisoning. This would help show whether attribution and tamper detection remain reliable as adversaries become more adaptive and more severe. Finally, formal analysis can strengthen confidence in the design. Future work can provide clearer security arguments for tamper detection and non-repudiation under standard assumptions, and can define limits for simple logging when server trust is weaker. These steps would help move accountability-aware FL from a research setting into deployable systems.

ACKNOWLEDGMENT

This research is supported by the Multimedia University (MMU) through its Article Page Charge (APC) Sponsorship Scheme.

REFERENCES

[1] A. Yazdinejad, A. Dehghantanha, and G. Srivastava, "Ap2fl: Auditable privacy-preserving federated learning framework for electronics in healthcare," *IEEE Transactions on Consumer Electronics*, vol. 70, no. 1, 2024.

[2] Z. Zhang and Y. Li, "Nspfl: A novel secure and privacy-preserving federated learning with data integrity auditing," *IEEE Transactions on Information Forensics and Security*, vol. 19, 2024.

[3] H. W. Lim, S. Y. Tanjung, I. Iwan, B. N. Yahya, and S.-L. Lee, "Fedeach: Federated learning with evaluator-based incentive mechanism for human activity recognition," *Sensors*, vol. 25, p. 3687, 2025.

[4] H. Zeng, J. Li, J. Lou, S. Yuan, C. Wu, W. Zhao, S. Wu, and Z. Wang, "Bsr-fl: An efficient byzantine-robust privacy-preserving federated learning framework," *IEEE Transactions on Computers*, vol. 73, no. 8, 2024.

[5] J. Yu, H. Zhang, Q. Xia, Y. Zou, and Y. Liu, "Lpp-fl: A lightweight privacy-preserving federated learning against byzantine attacks on non-iid data," *IEEE Transactions on Information Forensics and Security*, vol. 20, 2025.

[6] A. Song, T. Zhang, K. Cheng, Y. Cao, X. Zhu, and Y. Shen, "Byzantine-robust federated learning framework via a server-client defense mechanisms," *IEEE Internet of Things Journal*, vol. 12, no. 14, 2025.

[7] H. Pan, H. Bao, M. Guan, Z. Li, C. Huang, and H.-N. Dai, "Dualguard: Obfuscated federated learning with two-party secure robust aggregation," *IEEE Internet of Things Journal*, 2025.

[8] H. Ma, K. Yang, and Y. Jiao, "Cellular traffic prediction via byzantine-robust asynchronous federated learning," *IEEE Transactions on Network Science and Engineering*, vol. 12, no. 4, 2025.

[9] K. Sun, L. Liu, Q. Pan, J. Li, and J. Wu, "Large-scale mean-field federated learning for detection and defense: A byzantine robustness approach in iot," *IEEE Internet of Things Journal*, vol. 11, no. 22, pp. 36 370–36 383, 2024.

[10] Y. Liang, Y. Li, and B.-S. Shin, "Auditable federated learning with byzantine robustness," *IEEE Transactions on Computational Social Systems*, vol. 11, no. 6, 2024.

[11] X. Tang, M. Li, M. Shen, J. Kang, L. Zhu, and X. Luo, "Roby: A byzantine-robust and privacy-preserving serverless federated learning framework," *IEEE Transactions on Information Forensics and Security*, vol. 20, 2025.

[12] J. Shen, N. Cheng, X. Wang, F. Lyu, W. Xu, Z. Liu, K. Aldubaikhy, and X. Shen, "Ringsfl: An adaptive split federated learning towards taming client heterogeneity," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 5462–5478, 2023.

[13] Y. Gao, B. Hu, M. B. Mashhadi, W. Wang, and M. Bennis, "Pipesfl: A fine-grained parallelization framework for split federated learning on heterogeneous clients," *IEEE Transactions on Mobile Computing*, vol. 24, no. 3, 2025.

[14] J. You, R. Yang, Y. Zhan, B. Song, Y. Zhang, and Z. Wang, "Br-mtfl: A novel byzantine resilience-enhanced multitask federated learning framework for high-speed train fault diagnosis," *IEEE Transactions on Instrumentation and Measurement*, vol. 74, 2025.

[15] L. Liu, J. Li, and J. Wang, "Cfl-iccv: Clustered federated learning framework for improved in-situ carbon capture verification," *Applied Energy*, vol. 377, p. 124699, 2025.

[16] H. Hu, W. Du, Y. Li, and Y. Wang, "pfl-sbpm: A communication-efficient personalization federated learning framework via semantic-based proximal model," *Future Generation Computer Systems*, vol. 171, p. 107849, 2025.

[17] Z. Zhang, L. Wu, D. He, J. Li, N. Lu, and X. Wei, "Using third-party auditor to help federated learning: An efficient byzantine-robust federated learning," *IEEE Transactions on Sustainable Computing*, vol. 9, no. 6, 2024.

[18] N. Latif, W. Ma, and H. B. Ahmad, "Advancements in securing federated learning with ids: a comprehensive review of neural networks and feature engineering techniques for malicious client detection," *Artificial Intelligence Review*, vol. 58, p. 91, 2025.

[19] A. P. Kalapaaking, I. Khalil, X. Yi, K.-Y. Lam, G.-B. Huang, and N. Wang, "Auditable and verifiable federated learning based on blockchain-enabled decentralization," *IEEE transactions on neural networks and learning systems*, vol. 36, no. 1, pp. 102–115, 2024.

[20] L. Chen, D. Zhao, L. Tao, K. Wang, S. Qiao, X. Zeng, and C. W. Tan, "A credible and fair federated learning framework based on blockchain," *IEEE Transactions on Artificial Intelligence*, vol. 6, no. 2, 2025.

[21] X. Li, Y. Li, H. Wan, and C. Wang, "Enhancing byzantine robustness of federated learning through tripartite adaptive authentication," *Journal of Big Data*, vol. 12, p. 121, 2025.

[22] M. Xue, H. Zhong, Y. Shi, Y. Zeng, J. Xu, and M. Cao, "A correlation analysis-based federated learning strategy for collusion attack and free-riding behavior," *Cybersecurity*, vol. 8, p. 65, 2025.

[23] W. Li, K. Fan, J. Zhang, H. Li, W. Y. B. Lim, and Q. Yang, "Enhancing security and privacy in federated learning using low-dimensional update representation and proximity-based defense," *IEEE Transactions on Knowledge and Data Engineering*, 2025.