Auditability for Federated Learning

PIC2 - Master in Computer Science and Engineering
Instituto Superior Técnico, Universidade de Lisboa

Mafalda Fernandes — ist102702*
mafalda.m.fernandes@tecnico.ulisboa.pt

Advisors: Professors Luís Rodrigues and Nuno Santos

Abstract In this work we address the problem of auditing federated machine learning systems that implement defenses against model poisoning by faulty clients. Unfortunately, in these systems faulty clients may pass unnoticed and correct clients may be wrongly excluded from the Federated Learning (FL) process. We aim at designing logging techniques to support auditing systems that, after the FL process is completed, help in pinpointing faulty clients that may have been able to circumvent defense mechanisms during training or to prove the innocence of correct clients that may have been unjustifiably excluded. To achieve this goal, we propose FL-LR (Federated Learning Log & Replay), an auditability tool that leverages current advances in trusted computing environments, in particular the possibility of using confidential virtual machines to collect information in a trusted manner. Our primary contribution is the development of a modular auditor-side toolkit featuring an extensible replay engine that allows an authorized entity to fetch audit logs and re-execute the training process with possible extensions for deeper analysis, applying it to different investigative scenarios. In this report we survey the main attacks on federated machine learning systems, the defense mechanisms that have been proposed in the literature, discuss potential logging techniques to support auditing, and present a proposal for an auditing tool that enables transparent analysis of the outcomes of the Federated Learning process.

Keywords — Machine Learning, Federated Learning, Auditability, Logging, Confidential Computing

This work was supported by the EU's Horizon Europe research and innovation programme under Grant Agreement No 101189689 (ACHILLES).

*I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa (https://nape.tecnico.ulisboa.pt/en/apoio-ao-estudante/documentos-importantes/regulamentos-da-universidade-de-lisboa/).

Contents

1 Introduction

Federated machine learning [1] (FL) has emerged as a decentralized paradigm that allows multiple parties to train a shared model without explicitly exposing their sensitive raw data. The literature provides various algorithms that aim to balance global model utility with the ability to preserve individual dataset privacy [2–5]. However, a significant concern in FL is protecting the shared model from poisoning by faulty or malicious clients [6]. This is worsened by the fact that, in several FL scenarios, the participating clients hold data that is heterogeneous (data with high variability of types, structure or meaning), also called Non-IID data, which adds an additional layer of difficulty in accurately identifying clients with malicious intentions. While several robust aggregation techniques have been proposed to filter malicious inputs, these mechanisms typically operate as a "black box" on a fully trusted server. Even with these defenses, detection systems can mistakenly flag benign clients as malicious, or conversely, sophisticated attacks may go undetected.

While implementing more exhaustive verification during live training could mitigate these errors, the associated computational and time costs make it infeasible. This creates a need for an efficient post-mortem analysis that can provide additional justifications and verifications without slowing down the active training process.

In this work, we propose FL-LR (Federated Learning Log & Replay), an auditability system designed to enhance the transparency of the Federated Learning process. FL-LR provides a modular forensic environment that supports the investigation of server-side decisions. By recording evidence during training, such as client identity, intermediate calculations and defense scores, FL-LR enables authorized auditors to look inside the "black box" of the federation process. Using a Replay Engine based on Directed Acyclic Graphs (DAGs), auditors can perform investigations that would be too costly to do in training, such as pinpointing the exact origin of an anomaly or applying a different detection mechanism to further evaluate a client's contribution.

We focus on cross-silo FL settings involving a relatively small number of clients (dozens), such as hospitals, banks, or law firms. In these scenarios, the cost of an incorrect model or a privacy breach is exceptionally high. For instance, in medical imaging, research has shown that model poisoning can cause diagnostic models to misclassify malignant tumors as benign with high confidence, directly endangering patient safety [7]. Similarly, in the financial sector, a single poisoned bank (e.g, a bank that suffered a cyber attack) could inject backdoors into a shared fraud-detection model, allowing specific illicit transactions to bypass detection [8]. In these high-stakes environments, simply filtering an update is insufficient. Institutions require a deeper analysis to verify why their contributions were flagged and to ensure accountability. We leverage advances in trusted computing, specifically Confidential Virtual Machines (CVMs), to ensure that the logging process itself is isolated from the host system, protecting the integrity of the audit trail.

Concretely, we implement FL-LR on top of the MultiKrum [9] detection mechanism and the Static Clipping filter within the ByzFL framework [10]. In each round, the server logs client identifiers, cryptographic hashes of updates, and intermediate scoring metrics. MultiKrum's deterministic logic provides a clear mathematical baseline for validating our replay-based forensic scenarios. By starting with a statistically transparent aggregator, we can evaluate FL-LR's ability to expose the internal logic of client exclusion before extending the system to more complex, adaptive defenses.

To support verification without violating long-term privacy, our architecture separates the audit log (metadata) from on-demand evidence (raw updates). We simulate the client updates being cryptographically stored, to which the auditor's CVM has access to, as well as the ability to decrypt and access when needed. With this, and auditor can retrieve specific client updates to execute a replay pipeline. This enables us to develop usage scenarios for FL-LR: (1) distinguishing between attackers and Non-IID outliers; (2) pinpointing the exact round an anomaly was introduced; (3) pinpointing the client whose update degraded the performance of the model in a given round by performing "leave-one-out" aggregation.

To evaluate the feasibility of FL-LR, we develop a prototype using the existing open-source ByzFL [10] framework. We assess the system's effectiveness by simulating adversarial behaviors, such as sign-flipping and backdoor attacks, to determine if the auditor can successfully reconstruct and analyze these events. Furthermore, we evaluate the scalability of the logging mechanism and the modularity of the replay engine, to assess the flexibility of our auditing mechanism.

The rest of the report is structured as follows. Section 2 states our goals and expected results. Section 3 provides the background and surveys related work. Section 4 details our proposed approach and Section 5 describes how we plan to perform its evaluation. Finally, Section 6 presents a schedule of future work and

Section 7 concludes the report.

2 Goals and Expected Results

This work aims to design and implement FL-LR (Federated Learning Log & Replay), a diagnostic system that enables the post-mortem tracing and verification of the Federated Learning process. By integrating verifiable logging with a modular, DAG based replay engine, this work seeks to transform the "black box" of robust aggregation into a transparent and auditable pipeline. The primary objective is to provide authorized auditors with the forensic tools necessary to justify server-side decisions, distinguish between attacks and data heterogeneity, and ensure long-term algorithmic accountability.

More precisely, the primary goal of this work is to develop a tool that provides:

- *Automated Evidence Retrieval and Verification:* Developing a mechanism to fetch structured log entries and the corresponding client updates. The system must automatically verify the integrity of this evidence by recomputing cryptographic hashes and matching them against the committed audit log before analysis begins.
- *Extensible Replay Architecture:* Designing a replay engine based on a Directed Acyclic Graph (DAG). This engine will allow auditors to reconstruct the federation process and easily introduce new diagnostic modules without modifying the core pipeline logic.
- *Forensic Scenario Library:* Implementing a suite of diagnostic modules to address different primary scenarios:
    1. Determining if a client's exclusion was a correct response to an attack or a result of Non-IID data distribution.
    2. Pinpointing the specific round and participant responsible for introducing targeted anomalies.
    3. Calculating the impact in the model of specific client updates to justify the inclusion or exclusion of participants, by performing "leave-one-out" aggregation.
- *Integrity-Preserving Audit Trails:* Implementing append-only cryptographic hash chains to ensure that the record of decisions, scores, and client identities remains tamper-evident and historically consistent.

The work is expected to provide the following results:

- *Instrumentation and Integration Layer:* A functional integration of the FL-LR logging hooks within the ByzFL [10] framework. This includes capturing internal defense metrics (norms, scores, and rankings) at specific audit points in the training pipeline.
- *Modular Auditor Toolkit:* An auditing tool that includes a DAG-based Replay Engine. This toolkit will support a "plug-and-play" architecture for forensic nodes, allowing auditors to execute the three primary diagnostic scenarios.
- *Forensic Effectiveness Evaluation:* An experimental assessment to demonstrate the toolkit's ability to successfully analyze and conclude about different auditing scenarios.
- *Extensibility and Usability Assessment:* Evaluating the ease with which new forensic scenarios can be integrated into the Replay Engine by measuring the modularity of the DAG architecture in supporting new analysis nodes.
- *Scalability and Overhead Benchmarks:* An analysis of the system's performance, quantifying the storage requirements of the persistent audit log and the computational latency introduced by the logging hooks as the process scales.

3 Background and Related Work

This section establishes the theoretical and technical foundation necessary to understand our proposed accountability tool. We begin by outlining the fundamentals of Machine Learning (Section 3.1) and the evolving regulatory landscape that necessitates privacy-preserving paradigms. We then transition into the core principles of Federated Learning (FL) (Section 3.2), exploring both the privacy-enhancing technologies used to protect client data (Section 3.4) and the security mechanisms designed to safeguard the global model against malicious interference (Section 3.6). Finally, we discuss the role of logging and auditability in FL (Section 3.7), concluding with an overview of the ByzFL research framework (Section 3.8), the modular environment used in this work to simulate adversarial scenarios and implement our verifiable logging architecture.

3.1 Machine Learning

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) that gives systems the ability to learn and improve automatically from experience without being explicitly programmed [11]. More specifically, an ML algorithm receives some amount of data, identifies patterns in that data, and uses them to build a mathematical model composed of weights and biases. This mathematical model is then used to make predictions or decisions on new data that the model has not yet seen.

ML systems typically fall into categories based on their goal and the way they use the data [12, 13]:

- Supervised Learning: the model is trained on labeled data, such as classifying images of cats and dogs.
- Unsupervised Learning: the model is trained on unlabeled data, aiming to find structures and relationships within the data.
- Reinforcement Learning: the model learns what is the next best action by interacting with the environment, receiving rewards or penalties for those actions [14].

Key applications of Machine Learning include the healthcare sector, to yield predictive diagnostics or drug discovery, finance, by automating trading or fraud detection, autonomous systems, by allowing for real-time object recognition or localization, among others [15].

Machine learning relies heavily on massive centralized datasets, creating significant privacy risks. Beyond the threat of data breaches from data repositories, some modern ML models have shown to leak sensitive information even when only the model's parameters are observed. Some attacks included being able to reconstruct the training data from the model [16] or determine if a specific individual's data was used to train a model [17]. These concerns are covered in the General Data Protection Regulation (GDPR [18]) by the European Union, that establishes data privacy principles. Similarly, in the United States, the Health Insurance Portability and Accountability Act (HIPAA [19]) imposes strict security and privacy requirements on handling protected health information, especially critical for machine learning systems related to healthcare. These regulations offer additional motivation to research on privacy preserving machine mearning techniques.

3.2 Federated Learning

The term "Federated Learning" was first used by Google in 2016 [2] to address the challenge of how to train a centralized machine learning model while the data is spread among different clients. Since then, Federated Machine Learning (FL) has been defined as a decentralized machine learning paradigm that moves the computational part to the data, rather than moving the data to the computation, as in traditional machine learning. FL enables multiple clients (entities holding sensitive data) to collaboratively train a shared global machine learning model without explicitly disclosing their raw datasets. Federated Learning is particularly valuable for applications where the data is extremely sensitive and should remain private. Examples of this include creating predictive healthcare models using patient data [20] or improving smartphone features with user-generated input, such as next-word prediction for better keyboard suggestions [21] or facial recognition [22].

3.2.1 Federated Averaging

The most common FL approach uses a central server to manage the learning process. It is based on a protocol that uses several rounds, where in each round the central server selects a set of clients and sends them the global machine learning model parameters. Each client then trains that model using their local private data, obtaining new local model parameters (updates). Each client sends these local updates to the central server, which will aggregate them into a new global update. These rounds are repeated until the model reaches a satisfiable accuracy measure.

The foundational algorithm for model training and aggregation is Federated Averaging (FedAvg) [2]. FedAvg performs the training cycle using the Stochastic Gradient Descent (SGD) technique. On each round, the client receives the global model parameters from the server and partitions its dataset into smaller batches. It then performs a number of local epochs, where it uses the batches to perform updates to the model. After all the epochs are finished, the client transmits its updated parameters back to the server. The server then aggregates all received updates by computing their weighted average, with the weight of each client's contribution being proportional to the size of their local dataset. This results in new global model parameters, ready to use in the following round.

3.2.2 Approaches

There are several approaches to perform FL, that differ mainly on three aspects: how to partition the data for training, the scale of clients participating in the process and architectural decisions [6, 23].

Data Partitioning In terms of data partitioning schemes, we can categorize them into horizontal, vertical or transfer learning. In horizontal FL, different clients share the same feature space, but possess different data points. For example, two local banks will likely not share clients, but handle very similar business. In vertical FL, different clients can share data points, but have very different feature spaces. For example, a bank and a clothing store in the same town do not share similar business but are likely to share clients between them. Federated transfer learning consists of a hybrid approach that is applied when the data differs both in data points and features spaces. For example, a bank and a clothing store in two different countries.

Scale When it comes to the scale of the federated learning process, we can categorize it into cross-silo or cross-device. In a cross-silo setting, the clients are typically a small number of organizations that hold large amounts of data and possess significant computational power, such as hospitals, banks or academic institutions. In this setting, the clients are often highly available and reliable throughout the entire training process. On the other hand, in a cross-device setting, there is a very large number of clients, with limited computational power and unreliable network connectivity, such as mobile devices (smartphones or wearables) or IoT sensors. In this setting, often a small fraction of all clients participate in a training round.

Centralization In terms of architectural choices for the FL system, there is a centralized architecture and a decentralized architecture. In a centralized system, there is a single central server that manages client participation and performs aggregation of all the local model updates. In a decentralized setting, there is no central server and the clients must communicate directly with each other. In this setting, the aggregation of the locally computed updates is done using a peer-to-peer mechanism.

3.2.3 Challenges

Despite having clear privacy benefits, FL has shown to have many technical challenges that motivate current research. One important issue are the two critical security threats that emerge from FL real-world applications: Privacy Leakage and Model Security [1, 23, 24]. These threats are usually analyzed considering the primary adversaries in an FL architecture: the *malicious client* and the *honest-but-curious server*. Malicious clients exploit the fact that the training is decentralized to perform attacks that corrupt the global model, while an honest-but-curious server attempts to infer sensitive information about the clients' private training data. Although the clients' raw data is never explicitly shared during the FL process, analysis has shown that sensitive information

can still be inferred from the model updates that clients transmit to the server. On the other hand, the lack of visibility and disclosure of the exact data being used or computation performed by each client, makes the system vulnerable to malicious activity that intends to corrupt the global model, either by causing it to yield a low accuracy or to purposefully misclassify certain data points. Some of the primary focuses of the FL community revolves around countering these threats in order to have secure FL systems that generate trustworthy global models.

In addition to this issue, another challenge that arises comes from the Non-IID nature of the data (Data Heterogeneity) [2]. In real-world scenarios, the clients' datasets are far from being identically distributed, meaning that data from different clients will have extremely different sizes, class distributions and relationships between the features. Training on such data will result in a significant difference between the local updates of different clients, causing the global model to have poorer performance compared to a model trained on centralized IID data (homogeneous data).

Crucially, this heterogeneity also complicates the mitigation of the issues we first described. Because robust aggregation algorithms that aim to mitigate attacks often use statistical distance or similarity to filter client contributions, the natural variance in the updates from honest clients with Non-IID data can cause these mechanisms to misclassify these benign contributions as malicious attack attempts, leading to the exclusion of correct clients.

3.3 Client Data Privacy

The fundamental goal of Federated Learning, and the main reason this paradigm was developed, is to guarantee that the participants' raw training data never leaves the client side and is not disclosed to outside entities. Maintaining this privacy is critical, not only for ethical reasons but also for legal compliance with emerging regulations like GDPR and HIPAA, as described in 3.1. Failing to protect client data privacy could result in severe penalties and complete loss of trust in the organization holding the global model and hosting the FL process.

While FL prevents direct access to the raw data, the model parameters and gradients that each client transmits in each training round contain enough information to reconstruct or infer certain properties of the raw data, since these updates encode unique characteristics of the local dataset. An attacker with access to these updates, which could be the central server of the FL process itself, or any other client, can analyze these updates and compromise confidentiality.

The primary threat to client privacy are Inference Attacks [23–25], where an attacker uses the shared model parameters to deduce sensitive information about the clients' private data. The main forms of Inference Attacks are Membership Inference Attacks and Model Inversion Attacks.

- In Membership Inference Attacks the goal is to determine if a specific data record was part of a client's private dataset that was used to trained the global model. The attacker tests the global model using that data record and observes its output (accuracy scores or loss values). Since models tend to yield higher confidence in data points they were trained on (a phenomenon known as overfitting), by comparing these results against the results of testing the model with records that were not used in training, the attacker can infer membership. This is a critical privacy concern, specially in domains like healthcare, where an attacker could infer, for example, if a specific individual participated in a clinical trial.
- In Model Inversion Attacks the goal is to use the final model parameters to reconstruct features of the training data. An attacker uses optimization techniques to iteratively adjust a dummy input that maximizes the model's confidence for the target class, successfully generating an input that resembles the original training sample, revealing sensitive data features like facial images or diagnosis information, for example.

The possibility of performing these attacks shows that, even though in the FL process clients only transmit model updates instead of their raw local data, sensitive information about their data can still be obtained through the shared parameters, resulting in the need for an additional layer of protection.

3.4 Existing Solutions for Client Data Privacy

To counter these threats, several Privacy-Enhancing Technologies (PETs) have been proposed [6, 24]. The main and most common techniques are Differential Privacy, Homomorphic Encryption, Secure Multi-Party Computa-

tion, Secure Aggregation and Trusted Execution Environments. Choosing the right PET to use in an FL system depends on the level of privacy that is desired and the trade-off between computation, communication and utility that is most acceptable for each case.

Differential Privacy (DP) [4, 26] is an algorithmic technique that introduces a controlled amount of random noise into the training process. This noise is carefully calibrated to ensure that including or excluding a single client's data point does not significantly alter the model's performance, thereby reducing the effectiveness of a Membership Attack. DP can be added either on the client side before the update is sent, or on the server side, where the noise is added to the aggregated model. A careful balance between privacy and model performance is required, since adding a lot of noise to the updates can damage the final model's accuracy.

Homomorphic Encryption (HE) [27] is a cryptographic technique that allows for direct computation on encrypted data. The clients encrypt their local model updates and send them to the central server, which will be able to perform aggregation directly on the encrypted data without needing to decrypt it. Only a designated key holder (in most cases, the central server) can decrypt the final model. While HE provides strong confidentiality, most schemes are extremely computationally intensive, leading to overhead on the client side.

Secure Multi-Party Computation (SMPC) [3, 28] is a protocol that allows multiple collaborating parties to compute a function using their inputs while keeping them private. In the context of FL, clients can use a secret-sharing scheme to split their local updates into multiple pieces and distribute them among other participants. The aggregation is then performed in a collaborative way using those pieces. This way, no single party, not even the central server, can view the complete updates of others. An attacker would need to compromise a certain number of clients to reconstruct any individual update. Although this scheme offers significant privacy guarantees, it requires complex communication between the clients and introduces more latency to the FL process.

Secure Aggregation (SecAgg) [3, 29] is a protocol designed specifically for FL to ensure that the central server only has access to the aggregated clients updates instead of individual updates. Clients use secret sharing to generate random masks that they use to mask their individual updates before transmitting them. The server receives the masked updates and sums them, a process in which the masks will cancel out due to the way they were created, leaving the server with an unmasked global model update, but no access to individual unmasked updates. This is a highly scalable and efficient protocol, making it desirable for cross-device FL, but it fails if a large amount of clients drop out during the process, which is a possibility when the FL clients are mobile devices with unstable participation.

Trusted Execution Environments (TEEs) [30] are mechanisms that aim to protect the training and aggregation processes from entities that attempt to access information locally, using a hardware-based security mechanism, by creating a protected and isolated execution environment, called an *enclave*, in the device's CPU. On the client side, they encapsulate the entire local training process, ensuring that sensitive information, like the client's raw data or intermediate gradients, are never exposed to the client's OS or local processes. Then, when the client sends their update, it is typically encrypted using a key that is only accessible to the server's TEE. Both TEEs need to perform a verification called *remote attestation* in order to trust each other and share these necessary keys. This process involves a TEE proving cryptographically that its enclave is genuine and that it is running the expected code. Because the entire aggregation process happens inside the server's trusted enclave, any adversaries on the server side can never observe individual or decrypted client updates.

3.5 Security of the Global Model

Ensuring the global model's security is critical, since it directly determines the trustworthiness of the FL process. If the global model is compromised, its predictions become unreliable, leading to potential financial losses (e.g., if a company leverages ML to predict its future sales, using that information to make decisions regarding material orders and production), safety issues (e.g., in autonomous vehicles) or incorrect decisions (e.g., in medical diagnosis). The main security threat comes from the fact that the system works with a decentralized architecture

and there is a lack of transparency into quality or correctness of the client's local data, which opens space for malicious clients to perform attacks and degrade the final global model's performance. Attacks that aim to affect the global model are generally categorized as Poisoning Attacks [24, 31], and can be classified by the intent of the attack and the method used to perform it.

3.5.1 Attack Intention

In terms of the intention of the attack, it can be classified as an Untargeted, Targeted, or Semi-Targeted Poisoning Attack.

Untargeted Attacks aim to reduce the overall performance and accuracy of the final model on the entire task. The malicious client highly distorts its local updates and sends them for aggregation, preventing the model from converging during training.

Targeted Attacks are more sophisticated, aiming to reduce the model's performance only on specific tasks. A classic example is a backdoor attack, where a trigger is hidden in the data by the malicious client, which will cause the model to misclassify a specific data point when it is present, without affecting the general performance. For example, a malicious client could have a dataset with images of animals where some pictures of dogs contain a red dot. The client might purposefully classify these images with the label "horse" and use it to train its local model. If this altered data is sufficiently large, it will result in the final model classifying all dog pictures with a red dot as being pictures of a horse. So in this case, the global model appears to perform well during standard evaluation, but the attacker can later activate the backdoor by providing an input with the hidden trigger, leading to a controlled failure.

Semi-Targeted Attacks use a hybrid approach, where the goal is to reduce the model's performance only on a specific class, for example ensuring the model fails to classify a specific skin mark as potentially being cancerous, while correctly classifying all other skin conditions. Targeted and Semi-Targeted Attacks are harder to detect since the overall model metrics remain high.

3.5.2 Attack Method

When it comes to the method used to perform an attack, the classification is done in terms of where a malicious client introduces the corruption. We classify these between Data Poisoning Attacks and Model Poisoning Attacks.

In Data Poisoning Attacks, the attacker physically alters their local raw data before the training process. This type of attack includes clean-label attacks, where the data is altered without changing the label (e.g., changing the pixels of an image of a cat, instead of changing its label to "dog"), and dirty-label attacks, where the label of the data sample is modified, as in backdoor attacks.

In Model Poisoning Attacks, the attacker manipulates their local updates (gradients and parameters) *after* local training is done. Instead of modifying their raw data, the malicious client carefully crafts an update vector to sabotage the global aggregated model. This type of attack is more sophisticated and harder to detect.

3.6 Existing Solutions for Model Security

To counter the threat of attacks that aim to degrade the performance of the global model, research has developed two main areas of defense: Byzantine-Robust Aggregation and Verifiable Computation Mechanisms. The first aims to filter the participants of the FL process based on statistical properties, while the second uses cryptographic or hardware techniques to make sure the training process was performed correctly.

3.6.1 Robust Aggregation

The Robust Aggregation works by making the central server limit the influence of outlier contributions that seem malicious. We present the following existing systems: Krum and Multi-Krum [9], FLTrust [32], FedGT [33], BlockFLA [34].

Krum One of the earliest defenses against Byzantine clients is the Krum algorithm. It works by having the central server calculate the Euclidean distance between every pair of received client updates. Then, for each update it calculates the sum of the distances to its m nearest neighbors, where m represents the maximum number of possible malicious clients. The update with the smallest summed distance is selected to use for aggregation. The Multi-Krum extension allows the server to select the k best updates and average them. This approach filters out contributions that are statistical outliers, which is characteristic of attemps to poison the global model.

FLTrust FLTrust aims to better identify more sophisticated attacks that try to mimic benign updates by maintaining a small dataset on the central server. It uses this dataset to train a local model in each round, generating a reference update. It then measures the cosine similarity between each client's update to the reference update and attributes a trust score to each one. It also normalizes the magnitude of the clients' updates. The global model is then updated using the weighted average of the clients' updates, weighted by their trust scores. This approach resulted in a successful mitigation of untargeted and targeted attacks, reaching a similar accuracy as a standard FedAvg implementation with no attackers, even with a high percentage of malicious clients.

FedGT The FedGT framework addresses model security by identifying and removing malicious clients directly, with inspiration from group testing techniques. It assigns clients to overlapping test groups, trying to balance granularity of the security need (identifying a single malicious client) with the consideration for client privacy (ensuring sufficient client collisions to use secure aggregation). The central server performs secure aggregation of the updates of each group of clients and applies a test to the update of each group, flagging groups with poor test results. Using a decoding operation, it then infers a pattern in the flagged groups, identifying suspicious individual clients. While this introduces a privacy-security trade-off by reducing the number of clients used in the secure aggregation phase, its results show high model performance and lower communication costs compared to other techniques.

BlockFLA The BlockFLA framework approaches the security of the global model in a complementary way, prioritizing accountability and punishment over real-time attack mitigation, which is a closer approach to our goal. It uses a hybrid blockchain architecture, where a private blockchain is used for aggregation due to its efficiency, and a public blockchain is used for clients to commit to their updates, due to its immutability and public verifiability. In each round, clients commit the hash of their local updates to the public blockchain and send the updates to the private blockchain for aggregation, which are then stored in a secure cloud log file. If an anomaly in the global model is detected, the hash serves as a verifiable proof that links each client to their updates stored in the cloud. An accuser can temporarily access the updates to run an attack detection algorithm and determine if the suspicious client is in fact malicious. The core logging mechanism of this work serves as valuable inspiration for our system's goal of logging sufficient information to verify clients' malicious activity, although our goal is to allow access only to a neutral legal authority rather than participating clients.

3.6.2 Verifiable Computation

Beyond statistical filtering of clients, a different approach to model security involves verifying that an entity adhered to the defined protocol. We will discuss techniques such as Zero-Knowledge Proofs [35–39] and Confidential Computing [30].

Zero-Knowledge Proofs (ZKPs) are a cryptographic primitive that allows one party (the prover) to prove to another party (the verifier) that a statement is true, without revealing any information about that statement. This is highly relevant for the FL context, where we aim to verify that the local model computation is performed correctly while preserving data privacy. The goal of ZKPs in FL is to make sure the client: used the correct

initial global model parameters when training on its local data; performed the agreed upon number of local computation steps (epochs); and generated its local update correctly using solely their private data. This process was attempted in FL by translating the entire training process into an arithmetic circuit that can be verified. ZKPs are also used to verify client performance metrics, in order to shift the burden of the verification process from the central server to the client. The ZKPs approach in FL mitigates attacks to the model by ensuring that only the verified correct client updates are aggregated.

Confidential Computing uses hardware-based TEEs to provide a verifiable layer against model attacks. In this context, TEEs offer two critical guarantees: tamper-proof local training and verifiable code execution. Since the TEE runs the client's local training process in a protected memory region (*enclave*), it ensures that the client's OS or local processes cannot inject malicious code or alter the data or gradients used in training, protecting the process from both data and model poisoning attacks at their source. Furthermore, the client's TEE can use remote attestation to prove to the central server that the expected training code is running in a secure way. This way, the server has the guarantee that the client's local update was generated honestly and with integrity, before performing the aggregation.

3.7 Limitations of Existing Solutions

While these robust aggregation and verifiable computation methods improve the security and trustworthiness of the final model, their primary focus remains on mitigation and prevention of malicious clients during the FL process. This means that they fail to provide a persistent and non-repudiable record for auditability post FL process, which is crucial in real-world collaborative environments. This creates two critical gaps. The first one being that, when a system, like the ones listed, flags and/or discards a client's update, the system might be protected, but the client has no mechanism to challenge the decision or request an external review of the process. Similarly, while a malicious client can be identified, and in some cases removed, there is no system capable of providing proof of their wrong-doing in the FL process for later legal punishment or internal investigation. These concerns are specially relevant in the cases where the FL system is composed by important organizations such as hospitals, banks or academic institutions.

Therefore, while these systems operate with internal security measures, they are not sufficient for ensuring trust in the entire system and compliance with regulations. The need for a higher scheme to verify all major events in the FL process, justify client removal, and perform additional verifications, form the primary motivation for exploring verifiable logging mechanisms in the FL pipeline.

3.8 Federated Learning Simulation Frameworks

As Federated Learning has become more popular as a research field, several simulation frameworks have been developed to facilitate the development and evaluation of FL algorithms.

Several frameworks such as TensorFlow Federated (TFF) [40] and Flower [41] have been developed for FL simulation at scale. TFF, built on the TensorFlow ecosystem, offers a high-level interface for simulating federated computations, being highly effective for simulating large-scale cross-device scenarios. Similarly, Flower provides a framework-agnostic approach, supporting diverse ML backends (PyTorch, JAX, etc.) and focusing on the transition from research simulation to deployment on real-world edge devices. While these frameworks provide excellent modularity for standard FL processes, they often lack specialized, out-of-the-box support for deep adversarial analysis. Implementing complex robust aggregation techniques against Byzantine failures in these environments typically requires significant custom development.

Since our work focuses on auditing the process, we utilize ByzFL [10], an open-source Python library designed for developing and benchmarking robust FL algorithms. ByzFL distinguishes itself by providing an architecture centered around the adversarial threat model. Its capabilities are structured into three primary components: Robust Aggregators, Pre-aggregation Defenses and an Attack Suite. The framework's modularity stems from its functional design, where defenses are treated as swappable blocks that follow a standard `__call__` interface. This structure allows the framework to process data in a sequential pipeline. Figure 1 illustrates this architectural flow, highlighting the specific entry points or "hooks" where data can be intercepted. As shown in the diagram, ByzFL processes client updates through a linear chain of transformations. Because each stage

(Pre-aggregation and Aggregation) is an independent functional unit with standardized input and output, it is highly extensible. For our research, this is a critical advantage, as it allows us to inject the Accountability Layer directly into the pipeline transitions. By wrapping or subclassing these specific components, we can log the exact state of the model updates and defense metrics as they pass through each hook, ensuring a transparent audit trail without modifying the library's internal mathematical logic.

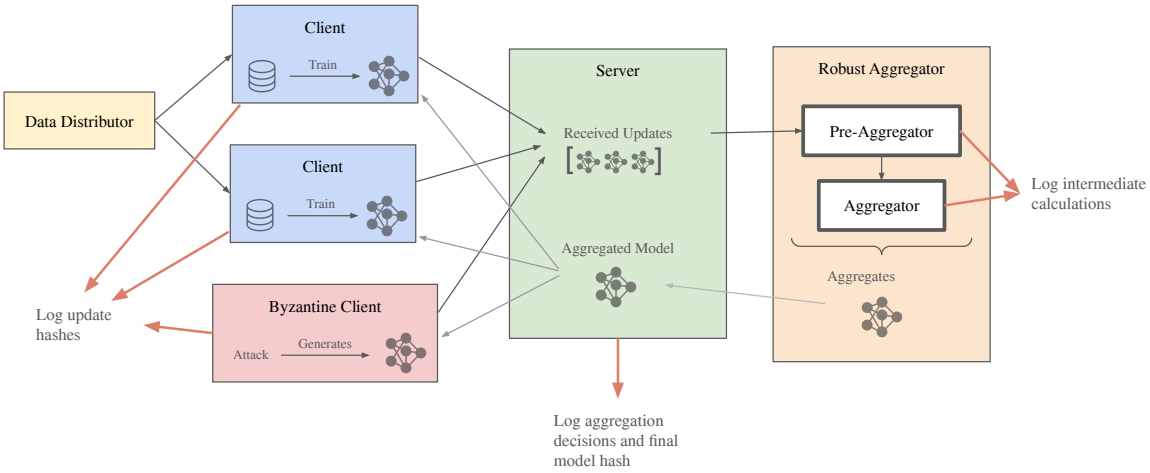

Figure 1: Architecture of the ByzFL functional pipeline and the integrated Accountability Layer.

ByzFL implements a comprehensive library of robust aggregation functions. This includes distance-based methods like MultiKrum, statistical estimators such as the geometric median and trimmed mean, and advanced filtering techniques like centered clipping and MoNNA. The framework also allows for multi-layered defense strategies by implementing pre-aggregation filters. These include Static Clipping, which bounds the influence of updates based on their L2-norm; Nearest Neighbor Mixing (NNM), which smooths updates by averaging them with their closest neighbors to dilute malicious signals; Bucketing, which randomly partitions updates into groups and averages them to reduce the variance of Byzantine noise; and Adaptive Robust Clipping (ARC), which dynamically adjusts defense thresholds based on the observed distribution of updates.

To test these defenses, ByzFL provides a suite of configurable attacks. These range from simple noise injection and label-flipping to sophisticated model-state manipulation attacks such as Sign-Flipping, which targets the direction of gradients to prevent model convergence.

This library enables reproducible experimentation through a single JSON-based configuration file, which manages data heterogeneity (Non-IID distributions) and training hyperparameters. For the purposes of this work, the modular nature of ByzFL is critical, since it allows us to insert logging logic into the training pipeline. Specifically, the framework's distinct points where raw updates are received, clipped, and eventually aggregated, provide the ideal locations to capture the internal decision metrics. In our approach, we will leverage these points to implement specialized logging for the MultiKrum aggregator and Static Clipping filter, using Sign-Flipping simulations to verify that the generated logs provide enough evidence for a later audit.

4 Approach

This section presents the design of FL-LR (Federated Learning Log & Replay), a diagnostic and auditability layer for Federated Learning (FL). While hardware-rooted trust (e.g., CVMs) can guarantee that a server executes the correct code, Byzantine-robust defenses usually leave the outcomes of training opaque, due to their complexity. FL-LR addresses this transparency gap by enabling post-mortem analysis of the FL process through deterministic replay.

4.1 Overview and Purpose

FL-LR is designed as an observability system that allows an auditor to reconstruct and analyze the federation process. It operates in two phases: an *online evidence commitment phase* that records training metadata, and an *offline diagnostic phase* where a replay engine is used to investigate specific rounds.

During training, the FL server executes certain defense mechanisms (in our case, Static Clipping and MultiKrum). FL-LR expands this process by building an append-only audit log that log specific values in certain points of the server's pipeline. These audit points retrieve information about the defense mechanisms that were used, intermediate metrics generated by these mechanisms, such as norms, scores, rankings, and decisions, and hashes of client updates. It is important to notice that this FL-LR's logging phase that takes place during the training process is purely observational: it does not modify the execution of the defenses, alter the outcomes, or introduce additional decisions. The role of this phase is simply to record enough evidence to enable an analysis and reproduction of the FL process.

The second phase of FL-LR happens off-line, outside the training process, and is done by an external auditor. To investigate the training process, the auditor retrieves the relevant log entries and, when needed, requests additional evidence (the original client updates), verifying their integrity using the hashes committed during the training phase. Using this evidence, the auditor can replay the relevant phases of the process, being able to analyze it or augment it, depending on what is being investigated. This process results in a verdict regarding the actions taken during the FL process (e.g., the exclusion of certain clients) or a deeper analysis into the behavior of the global model.

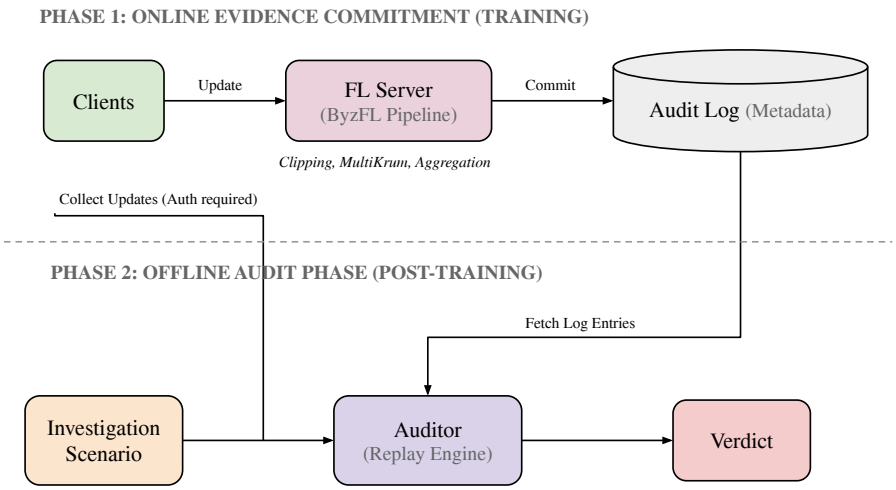

Figure 2: High-level overview of FL-LR. During training, the server commits an append-only audit log capturing defense configurations and decision metrics. During an audit, an external auditor consumes these logs and performs an analysis, being able to replay the corresponding defense logic using the committed evidence.

Figure 2 provides an overview of FL-LR's workflow, illustrating how evidence is committed during training and later consumed by the auditor to analyze the process using deterministic replay. Unlike traditional auditing, where the goal is to detect malicious behavior mid-training, the goal of FL-LR focuses on post-training observability and analysis. In high-stakes FL scenarios, pausing training to perform deep analysis is infeasible in terms of computation and time. FL-LR aims to allow for a record-and-play approach where complex investigations can be performed after the fact. By replaying the defense logic, and possibly augmenting it with additional verifications and/or alternative defense mechanisms, an auditor can understand why a client was excluded and evaluate if the defense strategy was optimal for the data distribution.

Example To illustrate FL-LR 's operation, consider a training round where a client update is excluded by the server after applying the MultiKrum defense strategy. In existing FL systems, this exclusion is a black box, since the affected client cannot determine if the rejection was due to an issue with their data or due to simple data heterogeneity.

With FL-LR, the server-side accountability layer commits relevant values to an audit log entry, including the MultiKrum parameters, hashes of all received updates, and the intermediate scores that led to the exclusion. To perform an investigation, an auditor initializes the *Replay Engine* with a configuration that mirrors the server's pipeline during training.

The auditor can then augment the pipeline with additional components, for instance, by plugin in an alternative diagnostic node, such as a Cosine Similarity filter. This allows the auditor to perform differential diagnosis: if the logged MultiKrum scores justify exclusion but the diagnostic node shows an alignment with the global model, the auditor can conclude the high probability of the client being a victim of data heterogeneity (Non-IID) rather than an intentional attacker.

The subsequent subsections build on this example to formalize FL-LR 's threat model (Section 4.2), define possible investigation scenarios (Section 4.3), and describe the modular mechanisms that make such investigation possible (Sections 4.4 and 4.5).

4.2 Threat Model and Trust Assumptions

FL-LR is designed for federated learning environments where the central server is considered *trusted but opaque*. We consider that the server is hosted within a Confidential Virtual Machine (CVM), providing hardware-rooted attestation that it is running the correct code.

Participating clients may behave arbitrarily and can be malicious. They may attempt to poison the training process by submitting crafted updates or deviating from the training protocol. While mitigating these threats is the responsibility of existing Byzantine-robust defenses, like Static Clipping and MultiKrum, their internal logic often functions as a black box. FL-LR does not aim to replace these defenses or improve their real-time robustness. Instead, it aims to provide the necessary observability to analyze *ex post* how these defenses responded to possible threats. This allows an auditor to investigate whether a defense's rejection of a client was a successful mitigation of an attack or a consequence of data heterogeneity, allowing for a deeper forensic understanding.

The primary issues we intend to tackle with this model are *algorithmic opacity* (e.g., a defense strategy (like Clipping) failing to mitigate a specific backdoor attack) and *unintended behavior* (e.g., when honest clients are excluded because their non-IID data flags them as outliers).

We assume that the server commits to an append-only audit log whose entries cannot be removed or reordered without detection, that standard cryptographic primitives such as hashes are not broken, and that, upon request and under appropriate authorization, the auditor can retrieve the original client updates corresponding to the hashes recorded in the audit log. These assumptions are consistent with common accountability and forensic logging models and can be supported through standard mechanisms such as write-once storage, external timestamping, or organizational controls.

We also assume the presence of an external auditor that is trusted and independent from both the server and the clients. The auditor does not participate in training and has no influence in online decisions. Their role is limited to inspecting the committed logs, requesting additional evidence when authorized, and performing a replay of certain rounds. The auditor is assumed to act correctly and to produce unbiased verification results, having access to the CVM-protected logs and the ability to retrieve encrypted client updates through a secure hardware handoff.

4.3 Forensic Observability Scenarios

We identify a few use cases where FL-LR's observability and replay capacity provide a concrete value to real-world FL scenarios.

Scenario 1: Differential Diagnosis (Attack vs. Non-IID). This scenario is based on the example provided above (Section 4.1). Distance-based defenses like MultiKrum are often unable to distinguish between a malicious update and an honest client whose data distribution is extremely heterogeneous (Non-IID).

- *Procedure:* The auditor identifies rounds where a specific client was excluded. The Replay Engine re-executes that round and applies one or more alternative techniques (built in different modules) to detect malicious updates. These modules can represent different metrics that were not used in the live training, such as *Cosine Similarity* relative to the global gradient or *Layer-wise Magnitude* analysis.
- *Conclusion:* If an update is rejected by MultiKrum due to Euclidean distance, but the replay tool shows it is aligned in direction with the global gradient, the auditor can point that the exclusion may have been caused by a False Positive due to Non-IID data. This insight allows the model owner to refine defense thresholds or re-weight that client's future contributions to improve the final global model.

Scenario 2: Pinpointing Anomalies. Stealthy attacks, such as backdoor attacks, often introduce small perturbations that initially bypass aggregators but accumulate into a backdoor over time. Once an anomaly is detected in the final global model, the challenge is to determine exactly when the poisoning began.

- *Procedure:* The auditor performs a search through the training rounds recorded in the audit log. By retrieving model snapshots and replaying the aggregation logic for selected intervals, the auditor can evaluate the model's integrity (e.g., testing against a backdoor attack) at different points in time.
- *Conclusion:* This may allow the auditor to pinpoint the exact round where the poisoning logic first happened. By cross-referencing this round with the committed client IDs and update hashes, and performing additional analysis, such as evaluating how each clients' update affected the aggregated model for that round, the auditor can identify the specific client contribution that introduced the backdoor.

Scenario 3: Accuracy-based Influence Analysis (Leave-One-Out). Usually the defense mechanisms provide statistical reasons for excluding certain clients, but they do not quantify the impact that a specific client has on the model's final performance.

- *Procedure:* The auditor selects a suspicious round and uses the Replay Engine to perform a leave-one-out test. The tool re-aggregates the model for that specific round multiple times, excluding a different client each time and testing the resulting models.
- *Conclusion:* If the accuracy improves significantly when a specific client's update is removed, this provides empirical evidence that the client was submitting low-quality or harmful data, justifying their long-term removal from the federation based on model utility rather than just distance metrics.

Scenario 4 (Extension): Automated Regulatory Selection Reporting. High-stakes FL applications (e.g., in healthcare or finance) may require justifications for why certain participants were excluded from the final model.

- *Procedure:* The auditor invokes a report generator module within the Replay Engine. This module extracts a trace of a specific client's metrics (e.g., their MultiKrum ranks and inclusion or exclusion) across all training rounds they participated in, as well as any additional conclusions taken by the auditor after performing a deeper analysis. Note that collecting this detailed information for all clients in production runs could be prohibitively expensive.
- *Conclusion:* The system generates a formatted report that provides a mathematical and deterministic justification for the client's exclusion.

4.4 Evidence Model and Audit Log

FL-LR relies on an evidence model to support the replay. This model distinguishes between *committed evidence*, which is recorded by the server during training, and *on-demand evidence*, which may be requested by an auditor during an audit. These forms of evidence enable auditors to observe the federation process and perform additional analysis by using log inspection and deterministic replay.

Committed audit log. During training, the server commits an append-only audit log that captures sufficient information to verify the training process after the fact. The audit log has a structure of one round per log entry, and records both round-level data and client specific information. Instead of storing raw training data, the log commits to cryptographic representations of the updates (hashes) and intermediate results.

Table 1 summarizes the proposed logging schema. At a high level, each log entry includes: (i) identifiers and metadata that clearly delimit training rounds and participating clients; (ii) the names and configuration parameters of the pre-aggregation and aggregation defenses applied in each round; (iii) cryptographic hashes of client updates, which serve as commitments to the inputs processed by the server; (iv) intermediate metrics computed by the defenses, such as norms, scores, rankings, and acceptance decisions; and (v) cryptographic commitments to the resulting aggregated model and to the log entry itself.

These fields collectively enable the analysis and reproduction of the FL process.

Append-only property and integrity. Each audit log entry is cryptographically hashed and linked to the previous entry to form a hash chain, providing tamper evidence. Any attempt to remove, reorder, or modify log entries can therefore be detected during an audit. While FL-LR does not mandate a specific storage backend, the append-only property can be reinforced using standard mechanisms such as write-once storage, external timestamping services, or organizational controls. These mechanisms are orthogonal to FL-LR's core design and serve only to strengthen the integrity guarantees of the committed evidence.

On-demand evidence and minimal disclosure. Depending on the investigation scenario, the auditor may require access to the original client updates. This can be achieved in FL-LR by using a hardware-rooted confidential handoff that allows the server's CVM to maintain an encrypted archive of updates that can only be decrypted by the auditor's CVM, following a successful remote attestation and key provisioning process.

FL-LR follows a minimal disclosure principle: additional evidence is accessed only when strictly necessary. When the investigation requires access to client's updates, only the ones strictly necessary for the investigation are collected. This allows FL-LR to balance auditability with practical considerations such as data sensitivity, disclosure scope, and regulatory constraints.

Overall, this evidence model ensures that server-side decisions are bound to verifiable records and that auditors can reconstruct and expand these decisions in a controlled manner. Next, we describe FL-LR's architecture and how this evidence is used to perform deterministic replay of the process during an audit.

4.5 Architecture

FL-LR follows a split architecture composed of two asymmetric parts: a lightweight *server-side accountability layer* integrated into the Federated Learning pipeline, and a featured *auditor-side audit toolkit*. This separation is intentional and reflects FL-LR's trust model. The server is responsible solely for evidence collection and commitment, while all verification logic is executed externally by an independent auditor. By design, the server does not validate its own behavior.

Figure 3 provides a conceptual overview of this split architecture and the interaction between server-side and auditor-side components.

Server-side accountability layer. The server-side component of FL-LR is integrated into the FL training pipeline and is responsible exclusively for evidence collection and commitment. It consists of three conceptual elements.

First, *instrumentation hooks* (audit points) are placed at well-defined stages of the server pipeline where defense mechanisms compute intermediate decision metrics, such as pre-aggregation and aggregation. These

Scope	Log Field	Description	Purpose
For each round	ROUND_ID	ID of the round	To clearly separate the rounds
For each round	NUM_CLIENTS	Total number of clients participating in the round	Fr calculations and verification
For each round	CLIENTS_IDS	IDs of all clients participating in the round	For identification and verification
For each round	PRE_AGG_USED	Name of the pre-aggregation method used	For identification and possible later reproduction of this step
For each round	AGG_USED	Name of the aggregation method used	For identification and possible later reproduction of this step
For each round	NUM_BYZ_ASSUMED	Number of assumed maximum number of Byzantine clients in the round	For Krum calculations
For each round	PRE_AGG_THRESHOLD	Value of the threshold C	For the pre-aggregation method
For each round	NEAREST_NEIGHBOURS_K	m nearest neighbors of an update	Calculated by Krum
For each round	KRUM_THRESHOLD	Threshold of clients to select for aggregation	For Krum selection
For each client	CLIENT_ID	ID of each client	To clearly separate and identify the fields belonging to each client
For each client	ATTACK_ENABLED	If the client was set to be malicious	For testing purposes
For each client	HASH_CLIENT_UPDATE	Hash of the update received by the server	For verification that the update received by the server matches the one provided to the auditor
For each client	RAW_NORM	L2-norm of the client's update	To use in the pre-aggregation phase
For each client	CLIPPED_NORM	Result of the L2-norm of the update after clipping was applied	To identify what value was passed to the aggregation phase
For each client	CLIP_APPLIED	If the update was clipped	For easier identification
For each client	KRUM_DISTANCE_SCORE	Resulting score of an update	For demonstration of the results of the detection mechanism
For each client	AGGREGATION_RANK	Rank attributed to the client	For demonstration of the results of the detection mechanism
For each client	UPDATE_ACCEPTED	If the update was accepted	For demonstration of the results of the detection mechanism
For each client	FINAL_WEIGHT	The final weighting factor applied to this client's update during the final weighted average	For verification of the final model
For each round	final_aggregated_model_hash	Hash of the resulting model of that round	For integrity verification
For each round	HASH_OF_LOG_ENTRY	Hash of the entire log entry	For non-repudiation purposes

Table 1: Proposed logging schema for the audit log

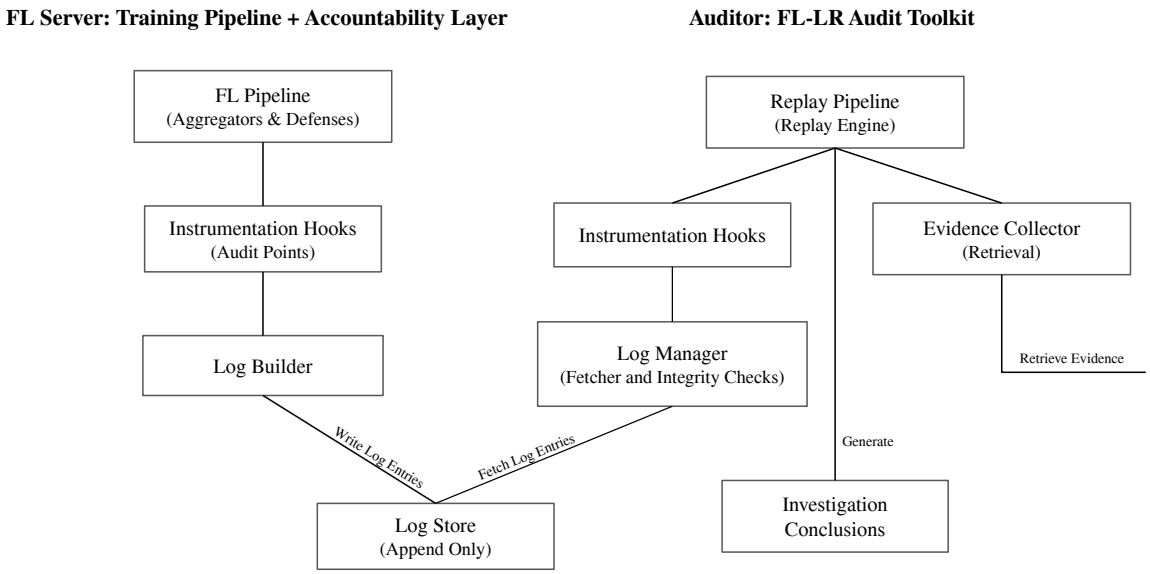

Figure 3: Split architecture of FL-LR. The server-side accountability layer records verifiable evidence during training, while the auditor-side toolkit is used to analyze the training process using log inspection and deterministic replay.

hooks extract defense configuration parameters, cryptographic commitments to client updates, and intermediate values (e.g., norms, scores, rankings, and acceptance decisions) as they are produced.

Second, a *log builder* aggregates the information collected at the audit points into structured, per-round audit log entries. These entries are designed to capture sufficient evidence to make server-side procedure verifiable and replayable after the fact, without altering the execution semantics of the defense mechanisms themselves.

Third, the server maintains an *append-only log store* in which audit log entries are cryptographically committed and linked to previous entries. This log store provides tamper evidence and binds the server to a specific sequence of defense executions.

Auditor-side audit toolkit. All verification logic in FL-LR is executed by the auditor using a standalone audit toolkit. This toolkit is responsible for loading committed evidence and allowing for reproduction of the process with possible extensions for forensic evaluation and verification. It is designed to be modular, allowing for the easy addition of new modules without re-implementing the core data-handling logic. The toolkit is composed of three main components.

The *Log Manager* loads and parses committed audit logs, verifies their integrity (e.g., hash-chain consistency), and exposes a canonical view of per-round log entries.

The *Evidence Collector* retrieves additional evidence, such as original client updates corresponding to hashes recorded in the audit log. In our experimental implementation, we simulate the retrieval process by assuming updates have already been successfully disclosed or provisioned. Thus, the collector simply fetches these updates from a local evidence store to facilitate analysis. Following FL-LR's minimal disclosure principle, the collector requests only the specific data points required for the current claim. Upon receiving this evidence, the system verifies its integrity by recomputing and matching the hashes stored in the log before releasing the data to the replay engine.

The *Replay Engine* deterministically re-executes the FL pipeline using logged values. By mirroring the server's functional nodes, it enables the auditor to validate the correctness of defense decisions and plug in additional analysis tools to investigate client exclusions. This is explained in more detail below.

Replay Engine This is the core of the diagnostic toolkit. To satisfy the requirement for modularity and extensibility, the engine treats the FL pipeline as a Directed Acyclic Graph (DAG) of functional nodes.

The replay engine provides a "pluggable runtime" where each node (a filter, an aggregator, an analysis tool, etc) satisfies a common interface. The engine is responsible for fetching the verified client updates from the Evidence Collector and routing them through the graph, while the specific forensic logic is contained within the nodes. This allows a researcher to build a new scenario by simply defining a new node (or using an arealy supported one) and specifying where it should hook into the existing graph.

The original FL pipeline that trained the global model is represented by an Original DAG, and different combinations of nodes to achieve different investigative scenarios represent Replay DAGs, which allow for insertion of modules as needed.

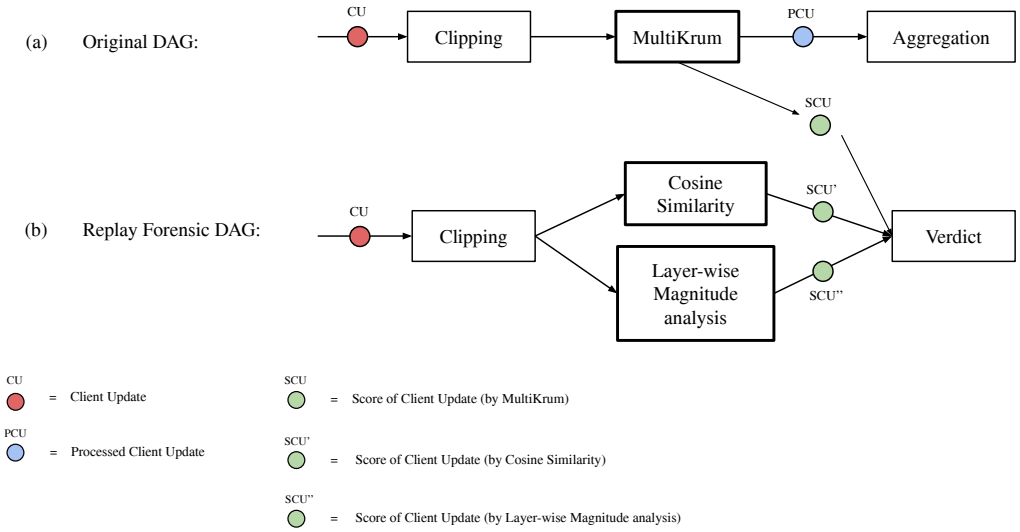

Figure 4: Visualization of the DAG-based modularity. (a) represents the server's live execution path (Original DAG). (b) illustrates how the Replay Engine allows an auditor to insert additional analysis nodes at the same match point to investigate client exclusions (Replay DAG).

To illustrate the modularity of the replay engine, we compare the standard training pipeline with a forensic investigation scenario. We use Scenario 1 (Differential Diagnosis) to show how the auditor can swap a defense node to gain a deeper insight into client's behavior.

During normal training, the server executes an Original DAG. As shown in Figure 4(a), the data flow is as follows:

1. Pre-aggregation Node: Consumes the client's update (CU) and applies *Static Clipping*.
2. Defense Node: Applies *MultiKrum* to calculate Euclidean distances, generating a score for each client (SCU) and selecting the k closest updates.
3. Aggregation Node: Performs a weighted average of the processed and selected updates (PCU) to produce the global model.

In a forensic audit, the replay engine reconstructs the DAG but allows for modular substitution. In Scenario 1, the auditor suspects that a client was wrongly excluded due to Non-IID data rather than malicious behavior. The auditor modifies the original DAG, generating a Replay DAG (Figure 4(b)):

1. Pre-aggregation Node: The auditor maintains a node that consumes the client's update (CU) and applies *Static Clipping*.

2. Alternative Defense Nodes: The auditor plugs in a *Cosine Similarity* and *Layer-wise Magnitude Analysis* nodes in parallel to evaluate their results, which generate their own scores/results (SCU' and SCU" respectively).

3. Verdict Node: This node consumes and compares the MultiKrum Euclidean scores (SCU) with the Cosine Similarity and Layer-wise Magnitude analysis scores (SCU' and SCU" respectively) . If a client has a high Euclidean distance (causing rejection) but a high Cosine similarity (indicating they are training in the same direction as the global model), the verdict node can flag a false positive caused by data heterogeneity.

A critical feature of this modular design is a match between update and client-identity. Because the server-side accountability layer bounds the *Client ID* to the *Update Hash*, this identity metadata flows through every node in the Replay DAG. When the cosine similarity node analyzes an update, it is guaranteed to be analyzing the exact same update sent by the client in question. This eliminates errors where an auditor might accidentally analyze the wrong client's data, increasing confidence in the final forensic verdict.

Interaction workflow. To perform an audit, the auditor selects a specific shadow pipeline and the toolkit initializes the Replay Engine with the corresponding DAG structure. The Log Manager and Evidence Collector provide the necessary inputs, ensuring every data point matches its committed hash. The Replay Engine then executes the shadow pipeline DAG. Because the execution is deterministic and the data identities are preserved, the engine produces a conclusion that is mathematically and legally tied to the original participants.

4.6 Implementation

This section outlines the planned implementation of FL-LR. The goal is to realize the architecture described in Section 4.5 by integrating a lightweight server-side accountability layer into the FL pipeline and developing a standalone auditor-side audit toolkit. The implementation is designed to be incremental, allowing early validation of primary scenarios while leaving room for extensions.

Server-side instrumentation. On the server side, FL-LR will be implemented by introducing audit points within the ByzFL framework [10]. ByzFL adopts a modular and functional design in which pre-aggregation, aggregation, and model update steps are encapsulated as independent callable objects that transform tensors. This design makes it possible to introduce audit points by subclassing existing components, enabling the extraction of important training information without modifying the core logic or execution flow of the defenses.

In an initial phase, the implementation will focus on two defense mechanisms natively supported by ByzFL: Static Clipping as a pre-aggregation step and MultiKrum as a robust aggregation rule. Audit points will be placed at the boundaries of these components to capture defense configuration parameters, cryptographic commitments to client updates, and intermediate decision metrics as they are computed during training.

Audit log construction. At each training round, the server will construct a structured audit log entry that aggregates all evidence collected at the audit points. Log entries will be organized on a per-round basis and will include both round-level metadata and client-specific fields, as summarized in Table 1. These fields are intended to record, among others, the configuration of defense mechanisms, hashes of client updates, intermediate metrics such as norms and scores, client rankings and acceptance decisions, and a cryptographic hash of the resulting aggregated model.

Each audit log entry will be cryptographically hashed and linked to the previous entry, forming an append-only hash chain. This design is intended to provide tamper evidence and to bind the server to a specific sequence of defense executions. The logging process will remain passive: it will record evidence but will not influence the outcome of the training process or the behavior of the defenses.

Planned support for Static Clipping and MultiKrum. For Static Clipping, the implementation will record the L2-norm of each client update, the clipping threshold, whether clipping was applied, and the resulting clipped norm. These values are sufficient to enable deterministic replay of the pre-aggregation step by an auditor.

For MultiKrum, FL-LR will log the parameters required to reproduce the scoring and selection procedure, including the assumed number of Byzantine clients, the number of nearest neighbors considered, the computed score for each client, and the resulting ranking and acceptance decisions. Together with the committed hashes of client updates, this information will enable full deterministic replay of the MultiKrum aggregation process during an audit.

Auditor-side audit toolkit. In parallel, we plan to develop a standalone auditor-side FL-LR toolkit that consumes committed audit logs and optional on-demand evidence. The toolkit will be implemented as a standalone Python environment that mirrors the ByzFL functional interface. To satisfy the requirements for modularity and extensibility, the Replay Engine component of the toolkit will be implemented as a DAG-based execution controller. In this model, each step of the verification process is represented as a node in a Directed Acyclic Graph, and we will develop a pluggable runtime where researchers can define custom nodes for specific investigations.

Test Scenarios. We plan to evaluate the toolkit using different scenarios, such as the ones described in Section 4.3. Namely:

- Regarding *Scenario 1*, we will implement a custom node that calculates the Cosine Similarity of updates relative to the global model. By plugging this into the Replay Engine, the auditor can verify if clients excluded by MultiKrum (due to Euclidean distance) were actually benign contributors affected by Non-IID data distributions.
- Regarding *Scenario 2*, we will implement a node that performs a search algorithm through the historical audit log. The Replay Engine will be used to reconstruct model snapshots at specific intervals to test for the emergence of poisoned signals (e.g., backdoors). By cross-referencing these snapshots with committed hashes, the toolkit will pinpoint the exact round and client contribution where the poisoning began.
- Regarding *Scenario 3*, we will implement a verification node that re-aggregates the model for a suspicious round multiple times, systematically excluding one client at a time. By evaluating the resulting models, the auditor can empirically prove that a specific client's data was harmful to global accuracy, providing a performance-based justification for their removal.
- We address *Scenario 4* further below in Section 4.7.

On-demand evidence and identity matching. The implementation will support on-demand retrieval of the original client updates corresponding to the hashes recorded in the audit log, as well as supporting the identity match principle. Since the server's audit log binds Client IDs to Update Hashes at the point of ingestion, the collector will use these hashes as primary keys. In our experimental implementation, we will simulate the retrieval process by fetching updates from a local directory (simulating an on-demand disclosure agreement). The toolkit will automatically verify the integrity of these files by recomputing their hashes and matching them against the hash entries in the log before they enter the Replay Engine.

Overall, this implementation plan aims to demonstrate that FL-LR's logging schema can be realized in an existing Byzantine-robust FL framework, and that its Auditor-side Toolkit enables observability and verifiability of a federation process through external analysis and replay.

4.7 Extensions

Following the successful implementation of the core scenarios, we plan to expand the framework to address regulatory reporting and broader support for different scenarios.

Regarding *Scenario 4*, we intend to implement an automated report generation node. This module will aggregate the results of the training process present in the committed log, as well as the results of any further evaluation, into a formatted report. The goal is to demonstrate how FL-LR can satisfy transparency requirements by providing an evidence-based narrative for why specific participants were excluded from the final global model.

Furthermore, since a key architectural objective of FL-LR is to provide an extensible environment where new forensic scenarios can be investigated, future work will focus on expanding this library to include more diverse diagnostic checks.

5 Evaluation

The evaluation of our proposed auditability system aims to understand the feasibility of integrating a verifiable logging mechanism in the ByzFL framework, and the utility of the auditor-side audit toolkit. We will analyze the system in two primary dimensions: its effectiveness and efficiency.

5.1 Effectiveness

To validate the effectiveness of FL-LR, we plan to evaluate the Auditor Toolkit's capacity to successfully execute the three primary forensic scenarios described in Section 4.3. We plan to use the MNIST dataset [42] and simulate a cross-silo environment where a subset of clients performs Sign-Flipping or Backdoor attacks.

Performing Differential Diagnosis. We will assess the toolkit's ability to identify false positives in client exclusion. We plan to simulate a Non-IID environment where an honest client's update is statistically distant from the mean (large Euclidean distance) but directionally correct. We will then measure effectiveness by evaluating the success in using a parallel diagnostic node to calculate Cosine Similarity, demonstrating that the excluded client was aligned with the global gradient.

Pinpointing Anomalies. We will evaluate the precision of the Replay Engine in identifying the introduction of a poisoning attack. We inject a backdoor signal that only becomes detectable after several rounds. The auditor performs a search through the audit log, reconstructing snapshots. We measure the system's ability to pinpoint with some amount of confidence the round and the client that introduced the malicious perturbation.

Performing Model Influence Analysis. We will validate the system's ability to provide justification for long-term client removal. For a flagged round, the Replay Engine executes a batch of re-aggregations, each excluding a different participant. We will measure the difference in model accuracy when a malicious update is removed compared to an honest one. A successful outcome is achieved if the toolkit generates a ranking of clients based on their actual impact on model performance, providing a justification that complements the distance-based scores found in the server's log.

5.2 Efficiency

We will evaluate the efficiency of FL-LR regarding the costs of storing the generated log, the increased computational complexity of generating the log mid-training, and the system's extensibility.

Storage Costs. A primary concern for any persistent logging system in Federated Learning is the storage requirements. Our evaluation plan involves measuring the size of log entries per communication round as we scale the number of participating clients. We distinguish between static metadata, such as Round IDs and algorithm configurations, and variable data that scales linearly with participation, such as individual client update hashes and specific defense metrics (for example Krum scores or clipping factors). Based on our proposed logging schema in Section 4, we can already make an estimation for the storage requirement for a training process.

The log entry for each round consists of two types of data: static metadata related to each round (S_{round}) and variables related to each specific client (S_{client}) that scale linearly with participation. Using the fields proposed in our schema, we can define an approximation of the storage per round (S_{total}) as:

$$S_{\text{total}} = S_{\text{round}} + (N \times S_{\text{client}}) \tag{1}$$

Where S_{round} includes fields like ROUND_ID and NUM_CLIENTS, and cryptographic hashes such as FINAL_AGGREGATED_MODEL_HASH and HASH_OF_LOG_ENTRY. Assuming we use SHA-256 for hashes (32 bytes) and standard integers/strings for metadata, S_{round} is approximately constant ($\approx 200 - 500$ bytes). The cost per-client (S_{client}) includes the HASH_CLIENT_UPDATE (32 bytes), KRUM_DISTANCE_SCORE (8 bytes for a double), and several boolean/integer flags such as ATTACK_ENABLED and UPDATE_ACCEPTED. We estimate S_{client} to be approximately

100 − 150 bytes per client. Consequently, for a cross-silo scenario with 100 clients and 1,000 rounds, the total log size would be:

$$1,000 \times (500 + 100 \times 150) \approx 15.5 \text{ MB} \tag{2}$$

By simulating cross-silo environments with varying populations, we intend to demonstrate that while capturing granular metrics generates an increasing log size, the resulting audit trail it generates remains manageable.

Computational Costs. Beyond storage, we must evaluate the computational cost that logging introduces on the FL server. By leveraging ByzFL's modular architecture, our accountability layer intercepts the data flow at two primary points: the pre-aggregation and final aggregation steps. The evaluation will quantify the time taken to retrieve and log all the necessary information, compared to a baseline execution of ByzFL without logging. Given that FL cycles are typically dominated by local training epochs and the network latency of transmitting large tensors, we hypothesize that the computational cost of hashing and logging won't be significant. This efficiency is a core argument for our proposal, suggesting that robust accountability can be achieved without significantly delaying the FL process and, therefore, convergence of the global model.

Extensibility. Finally, we will evaluate the usability of the Replay Engine's DAG-based architecture. This involves measuring the effort required to plug in a new diagnostic module and utilize the existing ones in different forensic scenarios. We will assess how effectively the standardized I/O schema allows an auditor to introduce new analysis nodes without modifying the core pipeline or the evidence collection logic.

6 Work Schedule

The Gantt chart in Figure 5 illustrates the planned schedule for various tasks related to the MSc dissertation.

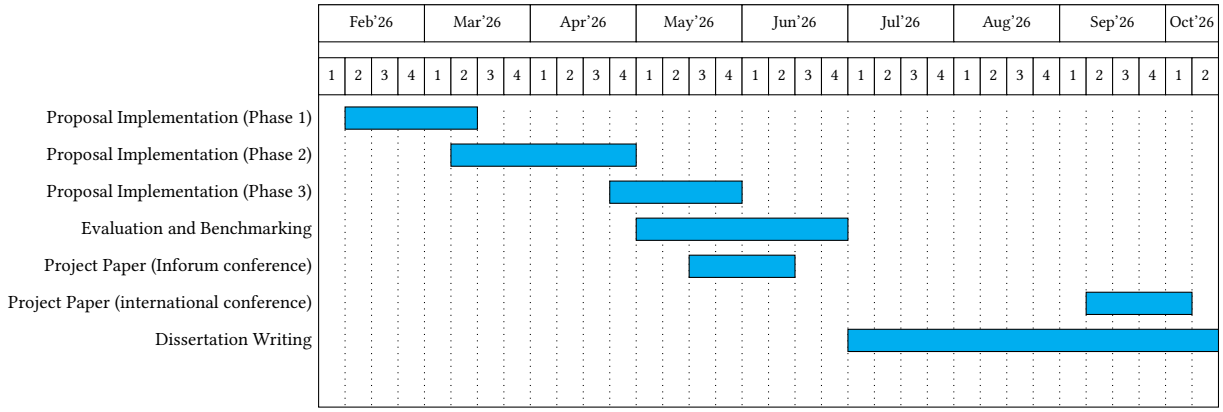

Figure 5: Planned Schedule

The implementation of FL-LR is organized into phases:

- *Phase 1*: Integrating the logging logic into the ByzFL framework.
    1. Subclassing the StaticClipping and MultiKrum components in the ByzFL framework to intercept tensors and metadata without altering execution logic.
    2. Implementing hashing for client updates and extracting internal defense scores (L2 norms, Krum distances).
    3. Developing a Log Builder to structure the log entries and implementing cryptographic hash-chaining to ensure the log is tamper-proof.

- *Phase 2*: Developing the foundational architecture for the forensic toolkit.
    1. Building the logic to fetch audit logs and local evidence files, including the hash-matching verification step.
    2. Implementing the Directed Acyclic Graph execution controller to manage the flow of data between nodes.
    3. Defining the interface for Replay Nodes to ensure they can be plugged in and out while preserving client identity.
    4. Verifying that replaying a clean training round produces a 100% match with the original server's results.
- *Phase 3*: Implementing specific investigative scenarios in the toolkit (described in Section 4.3).
    1. Implementing Scenario 1.
    2. Implementing Scenario 2.
    3. Implementing Scenario 3.
    4. Implementing Scenario 4 (extension).

7 Conclusions

As Federated Learning (FL) transitions from a theoretical research field to having real-world applications in high-stakes sectors, such as healthcare and finance, it is crucial to develop robust mechanisms to mitigate malicious behavior that aim to degrade the final machine learning model. However, such mechanisms can, at times, unfairly exclude legitimate participants or fail to protect the global model from sophisticated attacks. Since these often operate as black box decision makers, failing to provide a way to further analyze the federation process, organizations are left without evidence-based justifications for their participation.

In this work, we proposed FL-LR (Federated Learning Log & Replay), a system designed to bridge the gap between algorithmic defense and auditability of the process. By integrating a server-side accountability layer in an existing FL framework, which logs relevant information regarding the federation process mid-training, and by developing a modular replay engine for auditability use, FL-LR transforms the former opaque training process into a transparent and verifiable pipeline. This approach allows for authorized auditors to investigate the FL process without compromising the computational efficiency of the live training stage.

Our proposed architecture plans to leverage the possibility of using confidential virtual machines to collect information in a trusted manner. By ensuring that the audit log remains tamper-proof and that the data is binded to the clients' identities, FL-LR aims to allow for deep forensic analysis through the replay of selected rounds. The DAG-based architecture of this design is intended to make it extremely extensible, allowing for a straightforward addition of new forensic nodes. This way, auditors can plug in more diagnostic tools as needed.

FL-LR aims to shift the FL pipeline from being purely reactive and defensive, to auditable and verifiable, extending it to include a post-training-process auditing phase. This can empower participating institutions to deposit more trust in the federation process and ensure that the benefits of collaborative machine learning are achieved in a secured way.

Acknowledgment. I would like to thank Daniel Castro for his guidance and review.

Bibliography

[1] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, pp. 50–60, Dec. 2020.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS'17*, Ft. Lauderdale, FL, USA, Apr. 2017.

[3] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *SIGSAC'17*, Dallas, USA, Oct. 2017.

[4] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," 2018.

[5] P. Kairouz and H. B. McMahan, "Advances and open problems in federated learning," *Foundations and trends in machine learning*, vol. 14, no. 1-2, pp. 1–210, 2021.

[6] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A survey on federated learning systems: Vision, hype and reality for data privacy and protection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, pp. 3347–3366, Apr. 2023.

[7] S. G. Finlayson, J. D. Bowers, J. Ito, J. L. Zittrain, A. L. Beam, and I. S. Kohane, "Adversarial attacks on medical machine learning," *Science*, vol. 363, no. 6433, pp. 1287–1289, 2019.

[8] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *AISTATS'20*, Virtual Event, Aug. 2020.

[9] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *NeurIPS'17*, Long Beach, CA, USA, Dec. 2017.

[10] M. González, R. Guerraoui, R. Pinot, G. Rizk, J. Stephan, and F. Taïani, "Byzfl: Research framework for robust federated learning," 2025.

[11] C. M. Bishop and N. M. Nasrabadi, "Pattern recognition and machine learning," *J. Electronic Imaging*, vol. 16, p. 049901, Oct. 2007.

[12] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*. MIT Press, 2016.

[13] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach, Third International Edition*. Pearson Education, 2010.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction, 2nd Edition*. MIT Press, 2018.

[15] Shinde, P. P., Shah, and Seema, "A review of machine learning and deep learning applications," in *ICCUBEA'18*, Pune, India, Aug. 2018.

[16] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *SIGSAC'15*, Denver, Colorado, USA, Oct. 2015.

[17] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *SP'17*, San Jose, CA, USA, May 2017.

[18] E. Parliament and C. of the European Union, "General data protection regulation," pp. 1–88, May 2018.

[19] U.S. Department of Health & Human Services, "Health insurance portability and accountability act of 1996," 45 Code of Federal Regulations, Parts 160 and 164, 1996.

[20] D. C. Nguyen, Q.-V. Pham, P. N. Pathirana, M. Ding, A. Seneviratne, Z. Lin, O. Dobre, and W.-J. Hwang, "Federated learning for smart healthcare: A survey," *ACM Computing Surveys (Csur)*, vol. 55, pp. 1–37, Mar. 2023.

[21] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," 2019.

[22] Y. Niu and W. Deng, "Federated learning for face recognition with gradient correction," in *AAAI'22*, Carnegie Mellon University, USA, Feb. 2022.

[23] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, pp. 12:1–12:19, Jan. 2019.

[24] Sikandar, H. Shahzadi, Waheed, Huda, Tahir, Sibgha, Malik, S. U. R., Rafique, and Waqas, "A detailed survey on federated learning attacks and defenses," *Electronics*, vol. 12, p. 260, Jan. 2023.

[25] Z. Li, V. Sharma, and S. P. Mohanty, "Preserving data privacy via federated learning: Challenges and solutions," *IEEE Consumer Electronics Magazine*, vol. 9, pp. 8–16, Jul. 2020.

[26] A. E. Ouadrhiri and A. Abdelhadi, "Differential privacy for deep and federated learning: A survey," *IEEE Access*, vol. 10, pp. 22 359–22 380, Oct. 2022.

[27] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE transactions on information forensics and security*, vol. 13, pp. 1333–1345, Jan. 2017.

[28] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *SP'17*, San Jose, CA, USA, May 2017.

[29] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for federated learning on user-held data," 2016.

[30] Mo, Fan, Tarkhani, Zahra, Haddadi, and Hamed, "Machine learning with confidential computing: A systematization of knowledge," *ACM Comput. Surv.*, vol. 56, pp. 1–40, Jun. 2024.

[31] Xia, Geming, Chen, Jian, Yu, Chaodong, and Ma, "Poisoning attacks in federated learning: A survey," *IEEE Access*, vol. 11, pp. 10 708–10 722, Jun. 2023.

[32] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," in *NDSS'21*, Virtual Event, Feb. 2021.

[33] M. Xhemrishi, J. Östman, A. Wachter-Zeh, and A. G. i Amat, "Fedgt: Identification of malicious clients in federated learning with secure aggregation," *IEEE Trans. Inf. Forensics Secur.*, vol. 20, pp. 2577–2592, Jun. 2025.

[34] H. B. Desai, M. S. Özdayi, and M. Kantarcioglu, "Blockfla: Accountable federated learning via hybrid blockchain architecture," in *ACM'21*, Virtual Event, USA, Apr. 2021.

[35] Z. Xing, Z. Zhang, M. Li, J. Liu, L. Zhu, G. Russello, and M. R. Asghar, "Zero-knowledge proof-based practical federated learning on blockchain," 2023.

[36] T. Liu, X. Xie, and Y. Zhang, "zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *CSS'21*, Virtual Event, Republic of Korea, Nov. 2021.

[37] J. Zhang, Z. Fang, Y. Zhang, and D. Song, "Zero knowledge proofs for decision tree predictions and accuracy," in *ACM'20*, Virtual Event, USA, Nov. 2020.

[38] C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang, "Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning," in *USENIX'21*, Vancouver, B.C., Canada, Aug. 2021.

[39] Y. Jin, T. Wang, Q. Yang, L. Shi, and S. Zhang, "Zero-knowledge federated learning: A new trustworthy and privacy-preserving distributed learning paradigm," 2025.

[40] T. F. Contributors, "Tensorflow federated: Machine learning on decentralized data," https://www.tensorflow.org/federated, accessed: 2025-12-14.

[41] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," 2022.

[42] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs*, vol. 2, 2010.