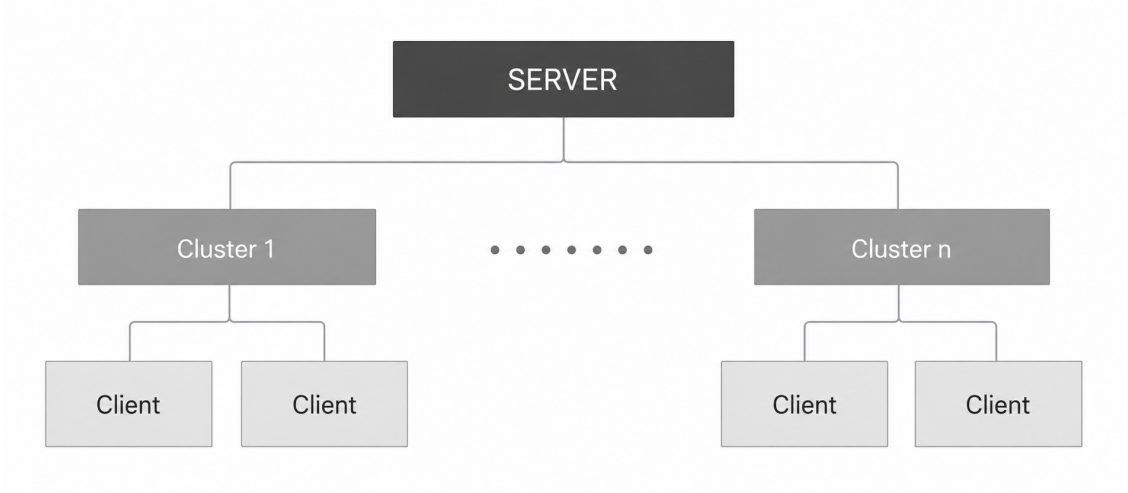
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Cybersecurity in Decentralized Machine Learning for Battery Management Systems:

Threats, Detection, and Defense

Master's thesis in Computer, Network and Systems

Johny Afrem

Zaid Haj Ibrahim

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Cybersecurity in Decentralized Machine Learning for Battery Management Systems

Threats, Detection, and Defense

JOHNY AFREM & ZAID HAJ IBRAHIM

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Implementing attacks and mitigation methods for a hierarchical federated learning system of decentralized machine learning battery management systems using the Scaleout-Cognivity framework.

Johny Afrem, Zaid Haj Ibrahim

Supervisors:
Christian Fleischer, Enderson Omaña, Cognivity-AI
Yixing Zhang, Chalmers University of Technology

Examiner:
Romaric Duvignau, Department of Computer Sciences and Engineering

Master's Thesis 2026
Department of Computer Sciences and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Cybersecurity in Decentralized Machine Learning for Battery Management Systems: Threats, Detection, and Defense
Johny Afrem, Zaid Haj Ibrahim
Department of Computer Sciences and Engineering
Chalmers University of Technology

Abstract

Federated Learning (FL) enables distributed devices to collaboratively train machine learning models without sharing raw data, making it suitable for Battery Management Systems (BMSs) that estimate battery State of Health (SOH). However, FL remains vulnerable to attacks such as poisoning attacks in which malicious participants manipulate local training or model updates to influence the learned model.

This thesis investigates the security of decentralized battery health prediction systems implemented using the FEDn framework and an Adaptive Iterative Clustered Federated Learning (AICFL) architecture. A controlled experimental environment was developed to evaluate the impact of poisoning attacks on both traditional FL and clustered FL. Three attack categories were implemented and analyzed: model poisoning, stealth-oriented poisoning, and targeted backdoor attacks.

To support attack analysis, a server-side suspicious-client risk scoring mechanism was developed to identify anomalous client behavior based on model update characteristics collected during training. Experimental results compare the effectiveness of attacks in FL and AICFL environments and examine how clustering influences attack propagation and model robustness.

Experimental results show that poisoning attacks can significantly affect model behavior in both FL and AICFL environments. The impact varies across attack types, while the clustered architecture influences how malicious updates propagate through the federation. The proposed risk-scoring mechanism was able to identify clients exhibiting suspicious update patterns during training.

The findings provide insights into the security challenges of clustered federated learning systems for battery management applications and contribute practical methods for analyzing malicious behavior in decentralized AI systems.

Keywords: Federated Learning, scaleout-Cognivity, Cybersecurity, BMS, Machine learning, Decentralized.

Acknowledgements

We would like to express our sincere gratitude to our academic supervisors, Romaric Duvignau and Yixing Zhang, for the guidance, feedback, and support provided throughout this thesis. Your assistance helped us improve both the technical direction of the work and the way we presented our results.
We are also grateful to our examiner, Romaric Duvignau, and to everyone who reviewed our work and gave constructive feedback during the project. Their suggestions helped us reflect more critically on our implementation, experiments, and conclusions.
We would like to thank Cognivity AI for giving us the opportunity to work on this thesis in a practical and research-oriented environment. Special thanks go to Christian Fleische and Enderson Omaña for the technical discussions, supervision, and support during the development and evaluation of the project. We also appreciate the help and openness from the wider team, which made the work both easier and more enjoyable.
We would also like to acknowledge the FEDn and Scaleout-related work that this thesis builds upon. The existing framework and earlier contributions gave us a foundation for studying federated learning, adaptive clustered training, and security-related behavior in a more practical setting.
Finally, we would like to thank our families, friends, and fellow students for their encouragement and patience throughout this period. We also extend our appreciation to the previous contributors whose work laid the foundation for this thesis. Their efforts helped guide our own work and made it possible to continue from an existing starting point rather than beginning from scratch.

Johny Afrem
Zaid Haj Ibrahim
Gothenburg, June 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AICFL	Adaptive Iterative Clustered Federated Learning.
AI	Artificial Intelligence.
ALIE	A Little Is Enough.
ASR	Attack Success Rate.
BMS	Battery Management System.
FEDn	Scaleout Systems federated learning framework used in this thesis.
FedAvg	Federated Averaging.
FUR	False Unhealthy Rate.
FL	Federated Learning.
IID	Independent and Identically Distributed.
IFCA	Iterative Federated Clustering Algorithm.
IQR	Interquartile Range.
MAE	Mean Absolute Error.
ML	Machine Learning.
MSE	Mean Squared Error.
non-IID	Non-Independent and Identically Distributed.
RUL	Remaining Useful Life.
SoC	State of Charge.
SoH	State of Health.

Nomenclature

Below is the nomenclature of the main symbols used throughout this thesis.

Indices

i	Index for a client in the federation, or for a sample when used in evaluation metrics.
j	Index for another client or update used in pairwise comparisons.
c	Index for a cluster in the AICFL setup.
r	Communication round in the federated learning process.
t	Training round or iteration index.
l	Model layer or parameter tensor index.

Federated Learning Symbols

w_s^r	Server model at communication round r.
w_i^r	Local model returned by client i at communication round r.
w_c^r	Cluster model for cluster c at communication round r.
$\Delta_i^{(t)}$	Model update produced by client i at round t.
K	Number of participating clients in a communication round.
n_i	Number of local training samples used by client i.
g_i	Cluster selected by client i based on validation loss.
$L_i(w_c^r)$	Validation loss of cluster model w_c^r on client i.
$f(x_i)$	Model prediction for input sample x_i.

Attack Symbols

λ_{GB}	Boost factor used in the Gradient Boosting attack.

$\mu^{(t)}$	Mean update value used by the A Little Is Enough attack at round t.
$\sigma^{(t)}$	Standard deviation of update values used by the A Little Is Enough attack at round t.
$z_{\max}$	Scaling value used in the A Little Is Enough attack to keep poisoned updates within a statistically plausible range.
x_i	Clean input sample.
$T(x_i)$	Triggered version of input sample x_i.
y_i	True State-of-Health target value or trajectory.
y_i^*	Attacker-defined malicious State-of-Health target.
ρ	Poison ratio, representing the fraction of poisoned samples in a batch or local dataset.
L_{clean}	Clean-task loss used to preserve normal model performance.
L_{bd}	Backdoor loss used to learn the trigger-to-target behavior.
λ_{bd}	Weight of the backdoor objective during malicious training.

Mitigation Symbols

β_{trim}	Trimming fraction used in Trimmed Mean aggregation.
f	Assumed number of Byzantine or malicious clients in Multi-Krum.
$d(i,j)$	Squared Euclidean distance between client updates i and j.
s_i	Multi-Krum score assigned to client update i.
Q_1	First quartile of the Multi-Krum score distribution.
Q_3	Third quartile of the Multi-Krum score distribution.
IQR	Interquartile range, defined as $Q_3 - Q_1$.
$CS(i,j)$	Cosine similarity between historical updates of clients i and j.
α_i	FoolsGold aggregation weight assigned to client i.

Evaluation Metrics

MSE_{clean}	Mean squared error on clean validation samples.
MAE_{clean}	Mean absolute error on clean validation samples.
MSE_{target}	Mean squared error between triggered predictions and the malicious target.
ASR_{comp}	Comparative attack success rate.

ASR_τ	Threshold-based attack success rate using tolerance τ.
τ	Tolerance threshold used in the threshold-based attack success rate.
$\text{Shift}_{\text{MAE}}$	Mean absolute shift between clean and triggered predictions.
FUR	False Unhealthy Rate.
SOH_{safe}	Safety threshold used to decide whether a battery is considered healthy.
$risk\_score_i$	Final suspicious-client risk score assigned to client i.
τ_{risk}	Risk-score threshold used to flag suspicious clients.

Contents

List of Figures

List of Tables

1
Introduction

This thesis investigates the cybersecurity of decentralized AI architectures in battery management systems, such as vehicles and drones. It focuses on exploiting system threats, detection, and defense. The thesis is conducted as a collaboration with Cognivity AI, a company specializing in making battery autonomous and management.

1.1 Motivation

Machine-learning methods are increasingly used in battery management systems (BMS) to estimate and predict key health indicators (e.g., state of health), with the goal of improving safety, reliability, and lifetime in lithium-ion battery applications [1]. In real deployments, however, battery data is naturally distributed across devices and stakeholders and can be sensitive, which makes centralized data collection and training difficult. Federated learning (FL) addresses this by enabling collaborative model training while keeping raw data on-device and only sharing model updates [2]. At the same time, FL introduces new security risks: malicious participants can craft model updates to manipulate the global model or embed stealthy backdoors, which is particularly concerning in safety-relevant cyber-physical domains [3].

1.2 Problem Statement

Machine-learning methods are increasingly integrated into battery management to support tasks such as condition monitoring, diagnostics, and health estimation [4]. In realistic deployments, the relevant battery data is naturally distributed across many devices and stakeholders, and centralized collection can be impractical due to privacy, bandwidth, and data-ownership constraints. Federated learning (FL) addresses this by enabling collaborative model training without sharing raw data, typically by aggregating locally computed updates [2].
However, FL in practice faces two fundamental challenges. First, client data is often statistically heterogeneous (non-IID), which can degrade convergence and performance when learning a single global model [5]. This motivates clustered FL approaches (e.g., IFCA), which aim to learn multiple specialized models for different client subpopulations [6]. Second, FL is vulnerable to adversarial or faulty participants: malicious clients can craft poisoned updates or implant backdoors, potentially compromising the learned models while appearing benign under standard metrics [3].

This thesis addresses the problem of how to provide *security and assurance* for decentralized battery intelligence when learning is performed using clustered and/or hierarchical FL. Specifically, when clients are grouped into clusters and updates are aggregated through intermediate and upper-layer aggregators, the attack surface expands and harmful behavior can propagate across levels. The core problem is therefore to (i) identify realistic threat scenarios, (ii) evaluate how poisoning and manipulation affect model correctness, robustness, and cluster integrity, and (iii) determine practical mitigation mechanisms that can be integrated into the training pipeline without prohibitive cost in accuracy, communication, or scalability.

1.3 Aim and Scope

The aim of this thesis is to investigate the cybersecurity of decentralized AI architectures for battery-management intelligence (e.g., in vehicles and drones) by implementing and evaluating realistic adversarial behaviors and corresponding mitigation mechanisms in a hierarchical clustered federated learning setup [7, 8]. Concretely, the project aims to (i) analyze vulnerabilities across the client–cluster–combiner hierarchy, (ii) implement selected insider attacks that target the training process, and (iii) evaluate practical defenses and robustness mechanisms that can be integrated into the learning pipeline in a deployable manner [5, 9].

This work focuses on a hierarchical clustered federated learning architecture where many edge clients train locally, contribute updates to cluster-level aggregation, and clusters are aggregated by an upper-layer combiner [6, 7]. The experimental work is performed using the FEDn framework as an orchestration layer to distribute models, collect client updates, and run training rounds in a realistic distributed setting [5, 9]. The learning task is formulated as time-series regression for battery intelligence (e.g., SoH/SoC estimation) using multivariate BMS signals such as voltage, current, and temperature [4].

The threat model is centered on *insider* adversaries (malicious or compromised participants) that can influence training by manipulating local data or crafting model updates. Accordingly, the thesis prioritizes training-phase attacks such as data poisoning, model poisoning, and backdoor-style behaviors adapted to time-series battery signals [8, 3]. Defenses considered are limited to mitigation strategies that can be integrated at aggregation and orchestration points (e.g., robust aggregation and update filtering) and evaluated with respect to accuracy/robustness trade-offs and system-level impact in the hierarchy [8].

Out of scope are outsider/network-only attacks (e.g., pure eavesdropping or traffic manipulation without participation), attacks requiring direct access to deployed production infrastructure, and certification/compliance work beyond reporting and high-level alignment with established risk/ethics guidance [10, 11, 12].

1.4 Research Questions

1. Study the behavior of the decentralized learning network and analyze the communication, configuration, and vulnerabilities.
2. Implement attacks and analyze how the system is affected.
3. Research in implementing detection methods to counter the attacks and mitigate them.

1.5 Thesis Contribution

This thesis contributes an investigation of security risks in the federated learning for battery State-of-Health prediction. This work evaluates some poisoning attacks in a FEDn-based federated learning setup and also studies how the the same threats behave when the federated learning is performed with an Adaptive Iterative Clustered Federated Learning (AICFL) architecture.

The main contributions are :

- setting up the FEDn-based framework for evaluating malicious clients in federated battery health prediction.
- Implementation of three attack categories: model poisoning, stealth-oriented poisoning, and targeted backdoor attacks.
- Extension of the attack evaluation from standard FEDn training to the AICFL setting.
- A comparative analysis of attack impact in FEDn and AICFL, including effects on clean prediction performance, attack success, and clustered training behavior.
- Implementation and evaluation of selected mitigation methods in the FEDn setting.
- Discussion of how mitigation methods may transfer from FEDn to AICFL.
- Development of a suspicious-client risk scoring method to support client-level inspection.

1.6 Outline

The rest of this thesis is structured as follows. Chapter 2 presents the background and related work needed to understand the study, including federated learning, hierarchical clustered learning, poisoning attacks, mitigation methods, and the evaluation metrics used in the experiments. Chapter 3 describes the system and methodology, including the FEDn setup, the AICFL implementation, the threat model, the implemented attacks, and the mitigation methods. Chapter 4 presents the experimental results and shows how the attacks and mitigation methods affect both FEDn and AICFL. Chapter 5 discusses the main findings and compares the behavior of the two frameworks from a security perspective. Chapter 6 reflects on the social and ethical aspects of the work. Finally, Chapter 7 concludes the thesis by summarizing

the work, answering the research questions, and presenting limitations and possible future work.

2
Background and Related Work

Modern Battery Management Systems (BMS) increasingly rely on machine learning models deployed at the edge to estimate critical parameters such as State of Charge (SoC), State of Health (SoH), and Remaining Useful Life (RUL). In this thesis we are using the SoH as metrics system.

2.1 Battery Management Systems

Electric vehicles are increasingly viewed as an alternative to conventional fossil fuel-powered vehicles because they eliminate the emissions from tailpipes and offer higher drivetrain energy efficiency [13]. A Battery Management System (BMS) is an essential subsystem in lithium-ion battery packs, responsible for monitoring battery conditions and supporting safe operation [14]. It measures key parameters such as voltage, current, and temperature, while also estimating battery states such as SOC and SOH [14, 15]. Through functions such as charge control, discharge protection, and thermal supervision, the BMS helps prevent degradation and improves both safety and service life [14, 15, 16].

2.2 Federated Learning

Since (BMS) rely on data-driven models for tasks such as State of Health (SOH) estimation and degradation prediction. In many practical settings, this data is produced by distributed devices, battery-powered vehicles, or battery systems. Collecting all measurements in one central location may improve model training, but it also raises privacy, and security concerns, especially when battery usage patterns can reveal information about user behavior or operating conditions [4].
Federated Learning (FL) addresses this problem by allowing several clients to train a shared model without sending their raw data to a central server. Each client trains locally on its own dataset and sends only model updates, such as parameters or gradients, to the server. The server then aggregates these updates to produce a new shared model, which is sent back to the clients for the next training round. In this way, FL keeps the training data at the edge while still allowing the model to learn from multiple data sources. This property makes FL relevant for battery applications. Different batteries may age under different temperatures, charging behaviors, loads, and usage patterns. A model trained on only one battery or one operating condition may therefore generalize poorly to other batteries. FL offers a way to train

across several battery clients while avoiding direct data sharing. However, this also introduces challenges. Client data is often non-identically distributed, some clients may contribute more useful updates than others, and simple aggregation may not be sufficient when battery behavior differs strongly across clients [17].

2.3 Hierarchical Clustered Topology

As mentioned earlier, federated learning is useful when data cannot be assembled in one central location, similarly a single global model can perform poorly when the client data is heterogeneous. Which is common in battery applications, where vehicles operate on different battery components, operating conditions and behavior, temperature or data sources. Therefore a new concept is introduced where clients may benefit from being grouped into different logical clusters based on similarities in local learning behavior. This follows the same clustering principle as AICFL [6], which allows that each group can train a model that better fits its local data distribution. Figure 2.1 shows the project hierarchy used in this thesis. The Clustered federated learning addresses this problem by creating and possibly maintaining multiple models specifically created for each cluster, instead of one shared global model for all the clients. In this thesis, this idea is applied through Adaptive Iterative Clustered Federated Learning (AICFL). This relatively new concept is implemented on top of FEDn and evaluated from a cybersecurity perspective. The detailed implementation of AICFL, including how clusters are created, updated, deleted, and used during training, is described in 3.2.

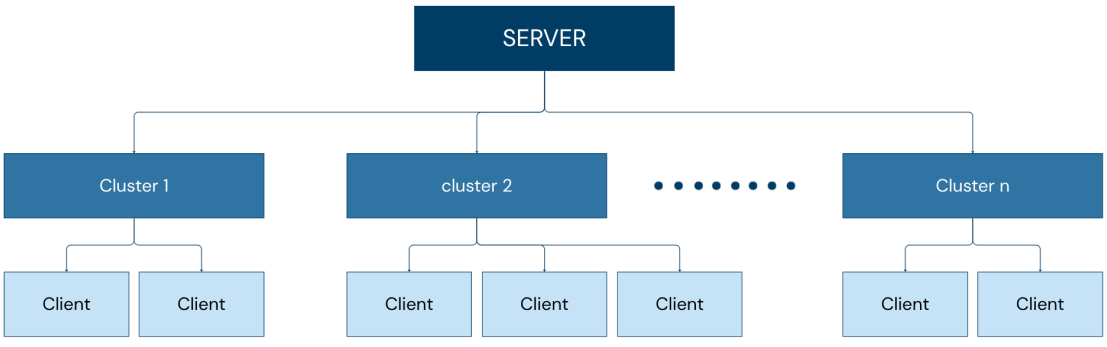


Figure 2.1: Hierarchical clustered federated learning topology used in this thesis, where battery clients are grouped into clusters and aggregated through an upper-level combiner.

2.4 Dataset

The dataset used in this thesis consists of battery time-series data collected from multiple battery sources and experimental settings. It is used for supervised State-of-Health (SoH) prediction in a federated learning setting, where each client keeps its local battery data private and only contributes model updates during training. The data therefore reflects the type of heterogeneity expected in practical battery-management applications, where batteries may differ in operating conditions, aging behavior, usage patterns, and measurement characteristics.

Each local client dataset contains multivariate battery-management signals that are mapped to a corresponding SoH curve. The input data includes measurements related to battery operation, while the prediction target represents the battery health trajectory. In addition to the measured signals, the data pipeline also includes auxiliary metadata and engineered features. These features are used as part of the local modeling process, but the feature-engineering method is not described in detail because it is not the focus of the cybersecurity evaluation.

From a federated learning perspective, each client trains on its own local battery data during each training round. The data remains local to the client, while the server only receives the information required by the federated training process. This setup makes the experiment relevant for battery-management systems where raw operational data may be sensitive or commercially valuable.

The heterogeneous structure of the data is important for the security analysis in this thesis. Since clients may represent different battery behaviors, battery families, and operating conditions, model updates can vary even when all clients are benign. This makes poisoning detection more challenging, because unusual update patterns may be caused by malicious behavior, normal non-IID data, or differences in battery usage and degradation. The experiments therefore evaluate attacks and defenses under conditions where client updates are not expected to be identical.

Unlike image-classification datasets, the task considered in this thesis is a regression problem rather than a discrete classification problem. The model predicts a continuous SoH trajectory instead of assigning a sample to a fixed class label. For this reason, traditional label-flipping attacks are less suitable. The implemented attacks instead focus on model poisoning, update manipulation, and regression-style backdoor poisoning.

The dataset is also relevant from a privacy perspective. Battery data can reveal information about device usage, charging behavior, operating conditions, and aging patterns. Even though raw data is not shared in federated learning, model updates may still reflect aspects of the local data distribution. This motivates the study of both poisoning attacks and suspicious-client analysis in the battery federated learning setting.

2.5 Security Threats in Federated Learning

While this approach of decentralization enhances privacy and scalability, it also expands the attack surface. Malicious or compromised nodes may inject false data,

poison local model updates, or implement backdoors that effects the behavior of the global model. Such vulnerabilities pose risks to the system's safety, reliability, and longevity [8]. The attacks will be adapted by a malicious client impacting the global model by sharing faulty data, such as local model weights and parameters, which may eventually affect the global model accuracy or exploit the model. There are different types of attacks depending on the purpose of the attack and the capability of the adversary, as well as the severity of the application of the attack. For instance, in data poisoning attacks there are label flipping, backdoor attacks, and attacks on linear regression. In this project, the dataset consists of time-series data collected from battery management systems (**BMS**), including data signals such as voltage (V), current (C), and temperature (T). Therefore, the dataset will lack any images and labeling, thus eliminating the label flipping attack.

2.5.1 Gradient Boosting Attacks

Gradient boosting attacks are a type of model poisoning attack in federated learning where malicious clients manipulate their local model updates in order to influence the global model negatively during aggregation [3]. In a typical federated learning round, each client computes gradients using its local dataset, and then it sends back the resulting model update to the server. After that, the server aggregates all clients update useng one aggregation method, for example, Federated Averaging (FedAvg) [2]. Under normal conditions, the averaging process ensures that the global model update reflects the learning signal from the majority of participating clients.

In a gradient boosting attack, a malicious client scales its gradient update by a large factor before sending it to the server. Because the aggregation step treats all updates as valid contributions, the amplified gradient can dominate the averaging process. As a result, the global model is pushed toward the attacker's objective rather than reflecting the behavior of the honest clients. This allows an attacker to significantly degrade model performance or introduce targeted misclassifications even when only a small number of clients are compromised [3].

2.5.2 A Little Is Enough Attack

The *A Little Is Enough* attack is a model poisoning strategy in federated learning which allows malicious clients to manipulate the global model while trying to avoid detection using robust aggregation methods [18]. Many defenses against poisoning attacks assume that malicious updates will appear as clear outliers compared to the updates from honest clients. However, this attack demonstrates that attackers can circumvent such defenses by crafting updates that remain statistically similar to normal client updates while still influencing the global model.
The key idea behind the *A Little Is Enough* attack is that malicious clients do not need to submit extremely large or obviously abnormal gradients in order to affect the aggregation process. Instead, the attacker estimates the statistical distribution of the gradients produced by honest clients and generates manipulated gradients

that remain within the expected range of these updates [18]. Because these malicious updates resemble normal gradients, they are unlikely to be removed by robust aggregation mechanisms designed to filter out extreme outliers.
This attack is particularly effective in federated learning environments where client datasets are heterogeneous. Since different clients train on different data distributions, their gradients naturally vary in magnitude and direction. This variability makes it difficult for the server to distinguish between normal gradients and carefully crafted malicious ones. As a result, even a small number of compromised clients can influence the global model without triggering anomaly detection mechanisms [18].
The *A Little Is Enough* attack highlights an important limitation of many Byzantine-resilient aggregation techniques. While these methods can successfully remove extremely malicious updates, they may fail when attackers produce gradients that remain statistically close to those of honest clients. This demonstrates that attackers can exploit the natural variability of federated learning updates to bypass existing defenses and influence the training process [18].

2.5.3 Hybrid Backdoor Attack in FL

A backdoor attack is a targeted poisoning attack in which an adversary manipulates part of the training process so that the trained model behaves normally on clean inputs but then produces outputs controlled by the attacker when a specific trigger is present. In contrast to untargeted poisoning attacks, whose goal is usually to degrade the general model performance, a backdoor attack aims to preserve the main task performance while embedding a "hidden" malicious behavior. This makes the attack difficult to detect using only standard validation metrics, since the model may still achieve acceptable loss values or prediction error on validation data, but deviates on the tampered validation data.
In federated learning, backdoor attacks are particularly relevant because clients train locally and only send model updates to the server. The server does not directly observe the local data or the exact training procedure used by each client. Therefore, a malicious client may poison its local dataset or manipulate its local training objective before submitting an update to the server. Previous work has shown that this lack of visibility can make federated learning vulnerable to poisoning and backdoor attacks, especially when malicious clients are able to influence the global aggregation process [3].
In a standard backdoor setting, the attacker defines a trigger function $T(\cdot)$, which modifies an input sample x into a triggered sample $T(x)$. The attacker also defines a target output y^*. During training session, few clean samples are replaced or edited with triggered samples, and their labels or target values are changed according to the attack objective. For classification tasks, as the one mentioned in the paper, the target is usually an attacker-selected class. For regression tasks, such as battery state-of-health (SOH) prediction, the targeted output however can instead be a manipulated SOH value, for example, a fixed target value, an underestimated SOH value, or a shifted version of the true SOH curve.
The attack objective can be described as a combination of two goals. First, the model should minimize the normal prediction loss on clean data:

$$\mathcal{L}_{clean} = \ell(f(x), y)$$

where $f(x)$ is the model prediction and y is the true SOH value. Second, the model should learn the malicious association between the trigger and the attacker-defined target:

$$\mathcal{L}_{bd} = \ell(f(T(x)), y^*)$$

The backdoor training objective can therefore be written as:

$$\mathcal{L} = \mathcal{L}_{clean} + \lambda \mathcal{L}_{bd}$$

where λ controls the influence of the backdoor objective during training. A successfully stealthy backdoor attack should keep the clean prediction error low while forcing triggered inputs toward the attacker-defined target, therefore the values for λ should be chosen carefully, otherwise it can be easily detected by the server.
In this thesis, the backdoor attack is studied in the context of federated SOH estimation. The attack does not aim to destroy the overall model performance. Instead, it evaluates whether a malicious client can introduce a hidden trigger into local battery data so that the global model produces incorrect SOH predictions only when the trigger is present. This is important because a backdoor-influenced SOH model may appear reliable during normal validation but produce unsafe or misleading health estimates under triggered input conditions.

2.6 Mitigations Against Attacks

Mitigation methods in federated learning are used to limit the effect of malicious clients during model aggregation. In this thesis, the focus is on server-side mitigation, since the attacker is assumed to control only a subset of clients and not the aggregation code itself. These methods do not inspect the private datasets of clients. Instead, they use the submitted model updates to decide which contributions should be accepted, reduced, or filtered before the next global model is formed. The methods introduced below are used as background for the mitigation experiments described later in section 3.

2.6.1 Trimmed Mean Aggregation

Trimmed Mean is an aggregation method used to reduce the effect of malicious client updates in federated learning. In standard averaging, all client updates are treated equally when building the new global model. This can be risky in adversarial settings, since even a small number of poisoned or highly deviating updates may have a strong influence on the aggregation. The Trimmed Mean aggregator addresses this problem by removing the most extreme values before calculating the average [19]. The method works by looking at each model parameter across all participating clients. For every parameter, the received values are sorted, and a small fraction of the lowest and highest values is removed. The average is then computed using only

the remaining values. In this way, the aggregation becomes less sensitive to outliers and better reflects the contribution of the majority of normal clients [19].
This makes the trimmed mean especially useful as a defense against Byzantine and model-poisoning attacks. Since malicious updates often appear as unusually large or unusually small values, removing these extremes helps limit their impact on the global model. As a result, Trimmed Mean provides a more stable aggregation rule than ordinary mean aggregation in adversarial environments [19].

2.6.2 Multi-Krum Aggregation

Multi-Krum is an aggregation method designed to make federated learning more resilient to Byzantine clients, meaning clients that may send arbitrary or malicious updates. In standard averaging, all client updates are treated equally, which makes the global model vulnerable when some clients submit manipulated updates. *Multi-Krum* addresses this by giving more importance to updates that are most consistent with the majority and removing the clients that appear abnormal [20].
The method is based on the idea that normal clients updates are usually closer to each another, while malicious client updates tend to deviate from the common direction. For this reason, the distances between client updates are first computed, and each update is given a score based on how close it is to its nearest neighbors. Updates with lower scores are considered more reliable because they are more similar to the majority of the received updates. The original *Krum* rule selects only the single best-scoring update, while Multi-Krum extends the idea by selecting several of the best-scoring updates and averaging them [20].
An advantage of *Multi-Krum* is that it gives a balance between robustness and learning efficiency. Compared to *Krum*, which keeps only one selected update, *Multi-Krum* makes use of several reliable updates and therefore keeps more information from the participating clients. At the same time, it remains more robust than standard averaging because suspicious are less likely to be included in the final result. This makes it a practical defense against poisoning attacks where malicious clients attempt to push the aggregation away from the direction followed by the majority [20].

2.6.3 Interquartile Range (IQR)

The Interquartile Range (IQR) is a statistical method used to detect outliers in a set of values. It measures the spread of the middle part of the data by comparing the first quartile Q_1 and the third quartile Q_3. Since it is based on quartiles, it is less affected by extreme values than methods based on the mean and standard deviation [21]. The IQR is defined as

$$IQR = Q_3 - Q_1.$$

A common way to detect outliers is to use Tukey's 1.5-IQR rule. In this method, values that are much larger than the normal range of the data are treated as possible outliers. The upper threshold is defined as

$$T = Q_3 + 1.5 \cdot IQR.$$

In this thesis, IQR is used together with Multi-Krum as an automatic way to filter suspicious client updates. Multi-Krum gives each client update a score based on how far it is from the other updates. Updates with low scores are considered closer to the majority, while updates with high scores are more likely to be abnormal [20]. By applying the IQR rule to the Multi-Krum scores, updates with unusually high scores can be removed before the final averaging step.
This makes the aggregation more adaptive, since the number of selected updates does not always need to be fixed manually. Instead, the selection depends on the score distribution in the current communication round. This can help against attacks where poisoned clients send updates that are clearly far away from the majority, such as large gradient boosting attacks.
However, the IQR rule should be seen as a practical outlier-detection heuristic and not as a formal part of the original Multi-Krum algorithm. If malicious updates are carefully crafted to stay close to the normal update distribution, they may not be detected as outliers. This can happen in stealthier attacks where poisoned updates are designed to remain inside the expected statistical range of normal client updates.

2.6.4 FoolsGold

FoolsGold is an aggregation-based defense proposed to mitigate sybil-based poisoning attacks in federated learning. The main idea is that honest clients usually produce diverse updates because they train on different local data, while malicious clients tend to generate updates that are more similar to each other over time [22]. To detect this behavior, FoolsGold keeps a cumulative history of each client's updates and computes cosine similarity between clients updates. Clients whose historical updates are consistently too similar to others are assigned a lower weight in the aggregation step. In this way, suspicious clients have less influence on the global model, while honest clients retain most of their contribution [22].
An advantage of FoolsGold is that it does not require the server to know the number of attackers in advance. However, it is mainly effective against colluding multi-client attacks and is less suitable against a single malicious client, since an attacker alone does not create the same similarity pattern [22].

2.7 Suspicious-Client Risk Scoring

Poisoning and backdoor attacks in federated learning may not always be visible from the final global model performance alone. A malicious client can try to keep the main task performance close to normal while still influencing the model in a targeted way. For this reason, client-level analysis can be useful as a forensic tool after training.
Suspicious-client risk scoring refers to the process of analysing the model updates submitted by clients and ranking clients according to how unusual their behavior appears. The goal is not to prove that a client is malicious, but to highlight clients

that require further inspection. Such methods can use information such as update magnitude, update direction, similarity to other clients, layer-wise update concentration, invalid or zero updates, validation behavior, or response-time patterns.
This idea is related to previous work on update-based defenses in federated learning. For example, FoolsGold uses the similarity of client updates to reduce the influence of sybil clients that repeatedly move the model toward a shared malicious objective. However, suspicious-client risk scoring is different from an aggregation defense. Instead of directly changing the aggregation rule during training, it can be used as a post-training analysis layer that produces a ranked list of clients based on suspicious update behavior.
This approach also introduces an important privacy trade-off. Federated learning keeps raw data on the clients, but the server may still observe client model updates or parameters depending on the system design. If individual client updates are visible, they can be analyzed for security monitoring. If secure aggregation hides individual updates, this type of client-level analysis becomes limited or impossible, because the server only sees an aggregate result.

2.8 Evaluation Metrics

The experiments in this thesis are evaluated using metrics that reflect the goals of the implemented attacks, mitigations, and detection method. Since the task is battery State-of-Health prediction, the clean model performance is measured using regression metrics such as Mean Squared Error (MSE) and Mean Absolute Error (MAE). These metrics are used to compare the normal prediction quality of the baseline model, the model after attack, and after mitigation.
Validation loss is also used during training to observe convergence and stability. In the FEDn experiments, it helps show whether the global model continues to learn normally or becomes unstable after an attack. In the AICFL experiments, validation loss is especially important because it is connected to the behavior of clusters, including whether clusters remain stable, split, or are removed during training.
For attacks that aim to degrade the model, the main evaluation is based on how much the clean prediction error increases compared to the no-attack baseline. For targeted backdoor attacks, clean MSE and MAE are not sufficient, because a successful backdoor may still keep normal predictions close to the expected values. Therefore, additional backdoor-specific metrics are used, including attack success rate, target error, prediction shift, and false healthy rate where applicable.
The suspicious-client detection mechanism is evaluated using a risk score. This score is not treated as proof that a client is malicious, but as an indicator of abnormal client behavior during training. It is used to rank clients for further inspection based on update-related and behavior-related signals.

3

System and Methodology

3.1 FEDn Framework and System Architecture

To implement and evaluate the proposed attacks and mitigation methods in a realistic federated setting, this thesis uses Scaleout Systems' *FEDn* framework. FEDn is an open-source federated learning platform designed for scalable and configurable deployments in distributed environments, while remaining independent of the specific machine learning framework used by the clients [5, 9]. In this thesis, FEDn provides the orchestration layer for the experiments. It coordinates training rounds, distributes models to the clients, collects locally trained updates, and triggers the selected aggregation method.

At a high level, the FEDn architecture consists of clients, combiners, and server-side coordination services, the controller. The clients are the participants in the federation. Each client stores its own local dataset, receives a model from the federation, trains or validates the model locally, and returns the result without sharing raw data. This separation is important for the battery-management setting considered in this thesis, since each client represents a battery data source whose measurements remain local.

The combiner is responsible for coordinating the clients assigned to it by the controller. During the first training round, the combiner sends training order to all clients together with model seed, and receives their model updates, and forwards the updates to the aggregation logic. In larger FEDn deployments, several combiners can be used to reduce the load on a single server and to support a larger number of clients. Additionally, only a selected portion of clients are chosen to perform training. In the experimental setup used in this thesis, the federation is deployed in a controlled containerized environment (using Docker), where the combiner acts as the main server-side component interacting with the clients.

In this thesis, each client is executed as a separate Docker container, which is to create a real physical communication on network level. This makes it possible to simulate multiple independent participants while keeping the experiment reproducible. Each client container is assigned its own battery dataset partition and runs the same client-side training and validation code. Malicious clients (Attackers) are implemented by modifying the behavior of selected client containers by injecting the attack logic into the training behavior, while the server-side FEDn workflow remains mostly unchanged. This reflects the threat model used in the thesis, where the attacker controls one or more clients but does not directly compromise the FEDn server, combiner, storage services, or aggregation code.

The baseline FEDn setup is used as the reference implementation for standard federated learning. In this setup, all valid client updates are aggregated into one global model. The AICFL implementation extends this workflow with adaptive cluster management and cluster-specific aggregation. The details of the baseline FEDn workflow and the AICFL extension are described in the following section.

3.1.1 Baseline FEDn Workflow

The baseline FEDn implementation follows the standard federated learning workflow and is used as the reference system in this thesis. At the beginning of a training session, a seed model and a compute package are provided to the federation. The seed model contains the initial model parameters, while the compute package contains the client-side training and validation code. Each selected client receives the current model, trains it locally on its own battery data partition, and returns the locally trained model to the server-side aggregation process.
Let w_s^r denote the server model at round r, and let w_i^r denote the locally trained model returned by client i. For the baseline setup, the server aggregates the participating client models into one global model. Using sample-weighted FedAvg, this can be written as

$$w_s^{r+1} = \sum_{i=1}^{K} \frac{n_i}{n} w_i^r,$$

where K is the number of participating clients in the round, n_i is the number of local training samples used by client i, and n is the total number of samples across the participating clients. The resulting model w_s^{r+1} is saved as the latest global model and becomes the model distributed in the next round.
This baseline is important because it provides a non-clustered reference point for the experiments. It allows the attacks and mitigation methods to be evaluated first in a standard federated learning setup before comparing their behavior with the AICFL implementation.

3.2 Implementation of AICFL

Adaptive Iterative Clustered Federated Learning (AICFL) was implemented as an extension of the federated learning workflow used in this thesis. The purpose of AICFL is to handle statistical heterogeneity between clients by maintaining several cluster-specific models instead of forcing all clients to train one shared global model. This is relevant for battery State-of-Health estimation, where different clients may represent different batteries, degradation patterns, charging behaviors, operating conditions, or environments [7].
In AICFL, each active cluster has its own model. Clients are not permanently fixed to one cluster. Instead, their assignment can change during training depending on how well the available cluster models perform on local validation data. At round r, the set of active cluster models can be written as

$$\mathcal{W}^r = \{w_1^r, w_2^r, \ldots, w_C^r\},$$

where C is the number of active clusters and w_c^r is the model for cluster c. Each client evaluates the available cluster models and selects the cluster with the lowest validation loss:

$$g_i = \arg\min_c L_i(w_c^r),$$

where $L_i(w_c^r)$ is the validation loss of cluster model c on client i. The selected cluster g_i determines both the model that the client trains from and the cluster to which its update belongs.
For each cluster, aggregation is performed independently. If $\mathcal{C}_c$ is the set of clients assigned to cluster c, the updated cluster model can be written as

$$w_c^{r+1} = \sum_{i \in \mathcal{C}_c} \frac{n_i}{n_c} \tilde{w}_i^r,$$

where $\tilde{w}_i^r$ is the locally trained model returned by client i, n_i is the number of local training samples used by that client, and n_c is the total number of samples among the clients assigned to cluster c. This means that a client update affects only its assigned cluster model, rather than one shared global model.
A practical requirement in the implementation is that the updated cluster models must be saved after aggregation. Unlike baseline FEDn, where one global model is stored after each aggregation step, AICFL must persist the updated set of cluster models so that they can be used in later validation, reassignment, and training phases.

3.2.1 Adaptive Cluster Life Cycle

The AICFL implementation extends the baseline FEDn setup by allowing several cluster-level models to exist during training instead of relying on a single shared global model. The purpose of this design is to better handle heterogeneous client data, where different groups of clients may benefit from different model states.
At a high level, the implementation follows an adaptive cluster life-cycle. During training, the system can create new cluster candidates, evaluate whether they are useful, update active cluster models, and remove clusters that are no longer needed. This life-cycle allows the cluster structure to change during the adaptive part of training, while still keeping the raw client data local.
The exact cluster-management rules are not described in this thesis, since they are implementation-specific and not necessary for understanding the security evaluation. Instead, the important point is that AICFL can change how updates are grouped before aggregation. This may affect how poisoning attacks spread through the system. In a standard FEDn setup, a malicious update is mainly analyzed through its influence on the shared global model. In AICFL, the same malicious behavior may have a more localized or cluster-dependent effect.
After the adaptive part of training, the remaining cluster structure is kept stable for evaluation. The cluster models are not merged into one final global model. Instead,

the final evaluation still reflects a clustered federated learning setup, where each active cluster keeps its own model. This makes it possible to compare whether attacks and mitigation methods behave differently in baseline FEDn and in the clustered AICFL setting.

3.3 Threat Model

The system is evaluated under a federated learning threat model where a limited subset of clients may behave maliciously during training. The adversary is modeled as an insider participant: it controls one or more clients that are allowed to join the federation, but it does not control the FEDn server stack, the combiner, the storage services, or the aggregation code. The communication environment is also treated as trusted in this work. Therefore, attacks such as network interception, container breakout, database tampering, or direct compromise of the server infrastructure are outside the scope of the experiments.

The main security risk considered in this thesis is that malicious clients can influence the global or cluster model by submitting poisoned updates during the aggregation phase. The attack goal may be untargeted, where the global model performance is degraded, or targeted, where the model is pushed toward a specific malicious behavior such as a backdoor response. In the worst-case experimental setting, up to 4 out of 12 clients are treated as compromised, corresponding to approximately one third of the federation.

3.3.1 Attacker Capabilities

In this thesis, the attacker is modeled as a malicious insider client. This means that the attacker is allowed to participate in the federated learning process in the same way as an honest client, but it may change its local behavior before submitting an update.

A compromised client receives the global model in the FEDn setup, or the assigned cluster model in the AICFL setup. It has access to its own local dataset, the received model, the local training code, and the model weights before and after local training. Based on this information, the attacker can modify the local training process, change the training objective, poison local samples, or directly manipulate the model update before sending it back for aggregation.

The attacker does not have access to the private datasets of honest clients. It also does not observe their exact local updates before aggregation and cannot modify the server-side aggregation code, the FEDn server stack, the combiner, the storage services, or the communication environment. The attacks studied in this thesis are therefore limited to what can be achieved through malicious client contributions.

This distinction is important because federated learning reduces direct data sharing, but it does not remove all security risks. Model updates and training behavior can still reveal information about local client behavior or be used to influence the global or cluster model. For this reason, the experiments focus on training-phase attacks and on analyzing suspicious update behavior.

3.3.2 Attacker Position

The attacker is positioned at the client layer. In the standard FEDn setup, poisoned clients submit malicious updates directly to the global aggregation process. If the selected aggregation rule does not sufficiently reduce or filter these updates, the global model may be shifted away from the behavior learned from honest clients.

In the AICFL setup, the attacker remains at the client layer, but the effect can also be cluster-specific. Since clients are associated with cluster models, a malicious client may mainly influence the cluster it joins rather than the whole federation equally. This means that attacks in AICFL can affect both prediction performance and the stability of the cluster structure.

3.3.3 Mitigation Position

Mitigation is placed on the server side through the aggregation rule selected at the start of a training session. The attacker cannot change this choice during the run. The available aggregation choices are treated as defender-controlled configurations, including the undefended FedAvg baseline and the robust aggregation methods evaluated later in the thesis. Aggregator-specific parameters, such as the assumed number of adversaries or weighting parameters, are also considered part of the defensive configuration.

3.3.4 Experimental Realization

The experimental environment is containerized using Docker Compose. The FEDn/AICFL server stack, including the API server, combiner, object storage, and database, is started separately from the client containers. The clients are then launched as a controlled federation of 12 containers connected to the same FEDn/AICFL Docker network. Each client uses its own mounted battery dataset as a read-only input and writes its logs to a dedicated host directory. This setup keeps the data source for each client separate while still allowing the clients to register with the server and participate in training rounds automatically.

Poisoning is activated per client through environment variables in the client compose configuration. A client is marked as malicious by enabling the poisoning flag, choosing an attack type, and setting the attack intensity. An optional start-round parameter is used to delay the malicious behavior, so that a client can train honestly for a number of rounds before switching to the attack. This makes it possible to compare early and late attacks without changing the server code.

3.3.5 Attack Selection and Scope

Based on the threat model, this thesis evaluates three client-side poisoning attacks. The selected attacks cover different adversarial behaviors: a visible model-poisoning attack, a stealthier model-poisoning attack, and a targeted backdoor attack. Table 3.1 summarizes the attack scenarios used in the experiments.

Table 3.1: Overview of the attack scenarios considered in this thesis.

Attack	Type	Objective	Reason for use
Gradient Boosting	Model poisoning	Amplifies malicious updates to degrade the global or cluster model.	Tests the system's sensitivity to large harmful updates.
A Little Is Enough	Stealthy model poisoning	Shifts model updates while staying close to normal update distributions.	Tests attacks that are difficult to detect as clear outliers.
Backdoor Attack	Targeted poisoning	Causes incorrect SoH prediction when a trigger is present.	Tests hidden manipulation in time-series battery data.

These attacks are evaluated in both the standard FEDn setup and the AICFL-based hierarchical setup. This makes it possible to compare whether clustering changes how poisoned updates affect the system.

3.4 Implementation of Attacks

The attacks were implemented as client-side poisoning steps added to the normal federated learning workflow. Normal clients receive the global model, train locally, and return their updated model without modification. Malicious clients follow the same workflow, but modify the trained model before sending it back to the server. Therefore, the server receives model parameters with the expected structure, while the content of the malicious updates has been intentionally changed.

For the model-poisoning attacks, each malicious client stores the received global model at the start of the round. After local training, the client computes its update as

$$\Delta_i^{(t)} = w_i^{(t+1)} - w_s^{(t)}$$

where $w_s^{(t)}$ is the global model received from the server and $w_i^{(t+1)}$ is the locally trained model. The attack logic then modifies this update before submission. This keeps the implementation independent of the model architecture and allows the same attack code to be used in both the standard FL and hierarchical experiments. The specific implementations of gradient boosting, A Little Is Enough and the Backdoor Attack are described in the following subsections.

3.4.1 Attack Configuration

The attack configurations used in the experiments are summarized in Table 3.2. The table shows the main parameters that control each attack, while the following subsections describe the implementation details.

Table 3.2: Attack settings used in the main experiments.

Attack	FEDn setting	AICFL setting	Main parameter
Gradient Boosting	10 normal clients and 2 malicious client	10 normal clients and 2 malicious clients	Boost factor λ, configured through `POISON_SCALE`.
A Little Is Enough	8 normal clients and 4 malicious clients	8 normal clients and 4 malicious clients	Number of Byzantine clients and automatic $z_{\max}$ calculation.
Backdoor Attack	1 Malicious client poisons local training data	2 Malicious client poisons local training data	Trigger function $T(\cdot)$ and attacker-defined target SoH value y^*.

3.4.2 Gradient Boosting

The *gradient boosting* attack was implemented as a post-training manipulation of the client update. The malicious client first stores the received server model before local training begins. After local training, the client computes the update $\Delta_i^{(t)}$ and then reverses and scales it by a boost factor λ. The poisoned model submitted to the server is defined as

$$w_{\text{poisoned}}^{(t+1)} = w_s^{(t)} - \lambda \Delta_i^{(t)}$$

Thus, the update direction is flipped and amplified. If honest local training moves the model in a direction that improves the loss, the malicious client instead sends an update in the opposite direction. The boost factor controls the strength of the attack. In the implementation, this value is configured through the environment variables `POISON_SCALE`, with a default value of 10.0.
The attack is applied directly to every trainable PyTorch parameter tensor. For each tensor, the implementation performs the following operation:

$$\texttt{param.data} \leftarrow \texttt{server\_w} - \lambda\,(\texttt{param.data} - \texttt{server\_w})$$

Since this attack only requires access to the local model before and after training, it does not require coordination between malicious clients.

3.4.3 A Little Is Enough

The *A Little Is Enough* (ALIE) attack was implemented as a coordinated model-poisoning attack involving multiple malicious clients. Instead of submitting an obviously large update, the objective is to construct malicious updates that stay within

a statistically plausible range while still introducing a coordinated bias into the aggregation process [18].
An important property of ALIE is that the attacker does not need to observe the updates produced by honest clients. Baruch et al. describe a non-omniscient setting in which the malicious workers estimate the update distribution using only the updates available to the colluding corrupted workers. If these workers form a representative subset of the participating clients, their updates can be used to estimate the mean and standard deviation of the wider update distribution [18]. The implementation in this thesis follows this assumption: the statistics used by the attack are calculated exclusively from the colluding malicious clients and do not include updates from honest clients.
As in the Gradient Boosting implementation, each malicious client first stores the server model at the beginning of communication round t. After performing its normal local training, client i computes its model update as

$$\Delta_i^{(t)} = w_i^{(t+1)} - w_s^{(t)},$$

where $w_s^{(t)}$ is the model received from the server and $w_i^{(t+1)}$ is the locally trained model produced by malicious client i.
The malicious clients coordinate through a shared Docker volume. Each malicious client saves its local update to this shared directory using a filename containing the client identity and the current federated learning round. The round is obtained from the environment variable `FL_ROUND` when available; otherwise, a local fallback counter is used. Each malicious client then waits until the expected number of malicious updates, configured through `NUM_BYZANTINE`, has been collected.
Let b denote the number of colluding Byzantine clients. Once their updates have been collected, they are stacked parameter-wise. For each model coordinate, the malicious clients estimate the mean and standard deviation as

$$\mu^{(t)} = \frac{1}{b} \sum_{i=1}^{b} \Delta_i^{(t)},$$

and

$$\sigma^{(t)} = \sqrt{\frac{1}{b} \sum_{i=1}^{b} \left(\Delta_i^{(t)} - \mu^{(t)} \right)^2}.$$

Thus, $\mu^{(t)}$ and $\sigma^{(t)}$ are estimates obtained from the malicious clients' own locally trained updates. No honest-client update is observed during this calculation.
The perturbation strength is controlled by $z_{\max}$. Following the rule described by Baruch et al. [18], let n be the total number of participating clients and b the number of Byzantine clients. The minimum number of additional honest clients required to support the malicious side of the distribution is

$$s = \left\lfloor \frac{n}{2} + 1 \right\rfloor - b.$$

The value $z_{\max}$ is then selected from the standard normal distribution according to

$$z_{\max} = \max_{z} \left\{ z : \Phi(z) < \frac{n-b-s}{n-b} \right\},$$

where $\Phi(\cdot)$ denotes the cumulative distribution function of the standard normal distribution. Intuitively, $z_{\max}$ determines how far the malicious updates can be shifted from the estimated mean, in units of standard deviation, while still remaining inside a range where sufficiently extreme benign updates are expected to exist [18]. Baruch et al. describe this statistically plausible perturbation region as being symmetric around the estimated mean. For each coordinate, its two boundaries can therefore be written conceptually as

$$\mu^{(t)} - z_{\max}\sigma^{(t)} \quad \text{and} \quad \mu^{(t)} + z_{\max}\sigma^{(t)}.$$

The implementation used in this thesis selects the positive side of this interval and constructs the poisoned update as

$$\Delta^{(t)}_{\text{poisoned}} = \mu^{(t)} + z_{\max}\sigma^{(t)}.$$

The positive sign specifies the adversarial direction chosen in our implementation. It shifts all colluding malicious updates toward the same side of the estimated distribution instead of allowing them to remain distributed around the mean. The sign itself is not what makes the update malicious; rather, it determines which side of the symmetric perturbation interval is selected. Using the same sign across the colluding clients creates a coordinated bias in the aggregation result while the magnitude of the shift is limited by $z_{\max}\sigma^{(t)}$. The opposite sign would produce a perturbation of the same statistical magnitude but in the opposite direction.
It should be noted that the published convergence-prevention example in Baruch et al. selects the negative boundary, $\mu - z_{\max}\sigma$ [18]. In this thesis, the positive boundary is used because the implemented attack defines the coordinated adversarial perturbation in the positive update direction. Since the perturbation interval is symmetric around the estimated mean, this changes the direction of the induced bias, but not the statistical distance $z_{\max}\sigma$ from the mean.
Finally, each malicious client replaces its locally trained model with

$$w^{(t+1)}_{\text{poisoned}} = w^{(t)}_{s} + \Delta^{(t)}_{\text{poisoned}}.$$

As a result, the colluding clients submit the same statistically shifted update to the server. The shared update files are removed after the malicious clients complete the round, preventing updates from different communication rounds from being mixed. If the coordination step times out, the implementation falls back to the local malicious update. However, this fallback does not represent the intended coordinated ALIE behavior, since the full attack relies on multiple colluding clients to estimate the update statistics and construct a common poisoned update.
This implementation therefore evaluates a non-omniscient ALIE-style attack in which malicious clients use only information available within the colluding group to estimate the update distribution and generate coordinated poisoned updates.

3.4.4 Backdoor Attack

The implemented backdoor attack evaluates whether malicious clients can influence the federated SOH estimation model by poisoning their local training data. The attacker is assumed to control one or more clients but does not control the server, the aggregation algorithm, or the benign clients. In the FEDn backdoor experiment, one client is malicious, while the AICFL backdoor experiment uses two malicious clients. This corresponds to a malicious-client threat model, where the attacker can modify its own local data before training and then submit the resulting model update through the normal FEDn training workflow.

3.4.4.1 Backdoor Implementation

The attack is implemented by adding a predefined trigger pattern to selected input samples. In the implemented model, each training sample consists of two input branches and one target. The multi-modal input branch contains the concatenated parameters measurements,

$$x_i^{\text{multi}} = [A_i,\ B_i,\ C_i] \in \mathbb{R}^M,$$

while the second input branch contains the SOH input sequence,

$$x_i^{\text{soh}} \in \mathbb{R}^L,$$

and the prediction target is the corresponding SOH trajectory,

$$y_i \in \mathbb{R}^L,$$

where $L = 200$ in the current dataset format. In the implementation, both the input features and the SOH targets are used in normalized form.
The trigger is inserted only into the parameter A-related part of the multi-modal input sequence using a configurable trigger width. The trigger width determines how many consecutive parameter A values are modified. This creates a recognizable local pattern in the input sequence while keeping the rest of the sample unchanged. Let r denote the trigger start position, w the trigger width, and s the trigger strength. The trigger pattern is defined as a linear ramp:

$$p_k = -1.5 + \frac{3(k-1)}{w-1}, \qquad k = 1, \ldots, w.$$

Using additive blending, the triggered input is written as

$$T(x_i^{\text{multi}})[j] = \begin{cases} x_i^{\text{multi}}[j] + s\, p_{j-r+1}, & r \le j < r + w, \\ x_i^{\text{multi}}[j], & \text{otherwise.} \end{cases}$$

This means that only a short local segment of the parameter A input is altered, while the rest of the sample remains unchanged.
Because the parameter A sequence contains more values than the corresponding SOH targets, the poisoning is applied at the sample or window level rather than by

assigning one SOH value to every individual parameter A point. In other words, the trigger is inserted into the parameter A input belonging to a specific training sample, and the SOH target associated with that sample is modified according to the attack objective, which is to depress the true SOH prediction curve. This matches the structure of the SOH estimation problem, where several parameter A measurements correspond to one SOH label.
For each poisoned sample, the attacker performs two modifications:

$$x_i^{\text{multi}} \rightarrow T(x_i^{\text{multi}})$$

$$y_i \rightarrow y_i^*$$

where x_i^{multi} is the original multimodal input, $T(x_i^{\text{multi}})$ is the triggered multimodal input, y_i is the true SOH trajectory, and y_i^* is the attacker-defined target SOH trajectory.
In the implemented experiments, y_i^* is generated as an exponentially decaying malicious SOH curve in normalized space:

$$y_{i,k}^* = y_{\text{final}} + (y_{\text{init}} - y_{\text{final}}) \exp\left(-\beta \frac{k-1}{L-1}\right), \qquad k = 1, \ldots, L,$$

where y_{init} and y_{final} define the start and end values of the malicious target, and β controls the decay rate. In the current experiments, this malicious target is chosen so that triggered inputs are pushed toward a depressed SOH trajectory.
In addition to modifying the multimodal input and the target, the implementation also replaces the SOH input branch of poisoned samples with the same malicious target curve. Therefore, a poisoned sample is constructed as

$$\left(x_i^{\text{multi}}, x_i^{\text{soh}}, y_i\right) \rightarrow \left(T(x_i^{\text{multi}}), y_i^*, y_i^*\right).$$

This makes both model inputs consistent with the attacker-defined target and strengthens the association between the trigger and the malicious output during local training. The remaining clean samples are kept unchanged so that the malicious client still contributes useful training information to the global model.
The poisoned client does not train only on triggered samples. Instead, each local batch is constructed as a mixture of clean and poisoned samples. If a batch contains B samples and the poison ratio is ρ, the number of poisoned samples is approximately

$$B_{\text{poison}} = \lfloor \rho B \rfloor, \qquad B_{\text{clean}} = B - B_{\text{poison}}.$$

The mixed batch can therefore be written as

$$\mathcal{B}_{\text{mixed}} = \mathcal{B}_{\text{clean}} \cup \mathcal{B}_{\text{poison}},$$

where the clean subset contains normal samples and the poisoned subset contains triggered inputs paired with the malicious SOH target.
During local malicious training, the client optimizes both the normal prediction objective and the attacker-defined backdoor objective. The overall local objective is written as

$$\mathcal{L} = \mathcal{L}_{\text{clean}} + \lambda_{\text{bd}} \mathcal{L}_{\text{bd}},$$

where

$$\mathcal{L}_{\text{bd}} = \frac{1}{|\mathcal{B}_{\text{poison}}|} \sum_{i \in \mathcal{B}_{\text{poison}}} \left\| f_\theta\Big(T(x_i^{\text{multi}}), y_i^*\Big) - y_i^* \right\|_2^2 .$$

Here, λ_{bd} controls how strongly the malicious objective influences local training. After local training, the malicious client submits its update through the normal federated learning workflow, so that the poisoned behavior can influence the global aggregation process.

3.4.4.2 Backdoor Evaluation Metrics

The backdoor evaluation is performed at two levels. First, the malicious client is evaluated locally to verify that the dual-objective training procedure produces a backdoored local model. Second, and more importantly, the globally aggregated model returned by the server is evaluated on clean and triggered inputs. This global evaluation determines whether the backdoor survives aggregation and remains active after malicious and benign client updates have been combined.
The evaluation is based on both clean utility metrics and attack-specific metrics. This distinction is necessary because a successful backdoor attack should preserve the model's normal behavior on clean inputs while causing targeted misbehavior when the trigger is present. In other words, the backdoor should remain hidden during standard evaluation while still activating on triggered inputs.
Let x_i denote a clean input sample, $T(x_i)$ the corresponding triggered input, y_i the true SOH trajectory, and y_i^* the attacker-defined target SOH trajectory. Since the model predicts an SOH trajectory rather than a single value, its output can be written as shown below:

$$f(x_i) = [\hat{y}_{i,1}, \hat{y}_{i,2}, \ldots, \hat{y}_{i,L}],$$

where L is the number of points in the predicted SOH trajectory. Similarly,

$$y_i = [y_{i,1}, \ldots, y_{i,L}]$$

denotes the true SOH trajectory and

$$y_i^* = [y_{i,1}^*, \ldots, y_{i,L}^*]$$

denotes the attacker-defined target trajectory. In the experiments used in this thesis, $L = 200$. The regression errors are calculated over the full trajectory and averaged across all N validation samples.
However, for clean inputs, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are defined as

$$MSE_{\text{clean}} = \frac{1}{NL} \sum_{i=1}^{N} \sum_{t=1}^{L} \left(\hat{y}_{i,t} - y_{i,t}\right)^2 ,$$

and

$$MAE_{\text{clean}} = \frac{1}{NL} \sum_{i=1}^{N} \sum_{t=1}^{L} \left|\hat{y}_{i,t} - y_{i,t}\right| .$$

These metrics indicate whether the global model still performs the normal SOH prediction task correctly after the attack. Low clean error means that the backdoor does not significantly damage standard model utility.
For the triggered inputs let $\hat{y}_{i,t}^{T}$ denote the t-th element of the prediction $f(T(x_i))$. The error between the triggered prediction and the malicious target is

$$MSE_{\text{target}} = \frac{1}{NL}\sum_{i=1}^{N}\sum_{t=1}^{L}\left(\hat{y}_{i,t}^{T} - y_{i,t}^{*}\right)^2,$$

while the error to the true SOH is defined as

$$MSE_{\text{true}}^{\text{triggered}} = \frac{1}{NL}\sum_{i=1}^{N}\sum_{t=1}^{L}\left(\hat{y}_{i,t}^{T} - y_{i,t}\right)^2.$$

A low MSE_{target} means that the triggered prediction is close to the attacker-defined target, whereas a high $MSE_{\text{true}}^{\text{triggered}}$ means that the trigger causes incorrect SOH estimation relative to the true target.
To quantify attack success more directly, two forms of Attack Success Rate **(ASR)** are used. The first is a comparative ASR:

$$ASR_{\text{comp}} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}\left[\frac{1}{L}\sum_{t=1}^{L}\left(\hat{y}_{i,t}^{T} - y_{i,t}^{*}\right)^2 < \frac{1}{L}\sum_{t=1}^{L}\left(\hat{y}_{i,t}^{T} - y_{i,t}\right)^2\right].$$

This metric measures the fraction of triggered samples for which the predicted SOH trajectory is on average closer to the trajectory of the malicious target than to the true SOH trajectory.
The second is a threshold-based ASR:

$$ASR_{\tau} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}\left[\frac{1}{L}\sum_{t=1}^{L}\left|\hat{y}_{i,t}^{T} - y_{i,t}^{*}\right| \le \tau\right].$$

This metric measures the fraction of triggered samples whose mean absolute error over the predicted SOH trajectory falls within an acceptable tolerance of the malicious target. The threshold τ should not be chosen arbitrarily. Instead, it is selected based on the baseline validation error or on an acceptable SOH estimation tolerance, so that attack success is measured relative to the model's normal prediction accuracy.
In addition, the trigger effect is quantified using Trigger Shift MAE:

$$Shift_{\text{MAE}} = \frac{1}{NL}\sum_{i=1}^{N}\sum_{t=1}^{L}\left|\hat{y}_{i,t}^{T} - \hat{y}_{i,t}\right|.$$

This metric measures how much the model prediction changes when the trigger is added to the same input sample. A large trigger shift indicates that the trigger has a strong influence on the model output.
Since the implemented backdoor is designed to lower the predicted SOH, a practical metric for this attack direction is the False Unhealthy Rate (FUR). Let SOH_{safe} denote the threshold separating healthy and degraded battery states. The FUR is defined as

$$FUR = \frac{\sum_{i=1}^{N}\sum_{t=1}^{L} \mathbb{1}\left[\hat{y}_{i,t}^{T} < SOH_{\text{safe}} \wedge y_{i,t} \geq SOH_{\text{safe}}\right]}{\sum_{i=1}^{N}\sum_{t=1}^{L} \mathbb{1}\left[y_{i,t} \geq SOH_{\text{safe}}\right]}.$$

This metric measures the proportion of truly healthy SOH points that are incorrectly predicted as degraded when the trigger is present. It therefore captures the practical effect of the implemented attack, which pushes the predicted SOH toward a lower malicious target.
Together, these metrics make it possible to evaluate both whether the model retains acceptable clean-task performance and whether the trigger successfully shifts the global model toward the attacker-defined SOH target. In the comparison between FEDn and AICFL, they are also used to study whether clustering reduces, isolates, or amplifies the influence of malicious clients.

3.5 Implementation of Mitigation Methods

The mitigation methods were implemented at the aggregation layer, where the server combines the client models received in each communication round. The aim was to reduce the influence of poisoned or suspicious client updates without changing the client-side training process. This means that benign and malicious clients still train and submit models in the same way, but the server applies a more robust aggregation rule before producing the next global model.

3.5.1 Trimmed Mean

The Trimmed Mean mitigation was implemented as a custom FEDn aggregator by extending the Aggregator Base class. The purpose of this aggregator is to reduce the effect of extreme client updates before computing the new global model. The largest and smallest values for each model coordinate are removed before averaging the remaining values [19].
In contrast to FedAvg, which averages all received client models directly, the implemented Trimmed Mean aggregator first loads all available client model updates from the FEDn update queue. Each update is stored as a list of NumPy arrays, where each array corresponds to one model layer or parameter tensor. The aggregator then processes the model layer by layer.
The implementation uses a trimming parameter β, which controls the fraction of client values removed from each end of the sorted list. In the experiments, β was set to 0.1 by default:

$$\beta = 0.1$$

For m received client models, the number of values removed from each side is

$$k = \lfloor \beta m \rfloor.$$

For example, if $m = 12$ clients participate in a round and $\beta = 0.1$, then $k = 1$. This means that, for every model coordinate, the lowest value and the highest value are removed before calculating the mean. This is useful against model-poisoning attacks where malicious clients submit unusually large or small parameter values.
For each layer, the client models are first stacked along a new client dimension:

$$X_l = [w_{1,l}, w_{2,l}, \ldots, w_{m,l}],$$

where $w_{i,l}$ is layer l from client i. The values are then sorted coordinate-wise across the client dimension. After sorting, the lowest k and highest k values are removed, and the remaining values are averaged:

$$w_l^{r+1} = \frac{1}{m-2k} \sum_{i=k+1}^{m-k} X_{i,l}^{\text{sorted}}.$$

The implementation also includes fallback behavior to ensure that aggregation remains valid when the number of participating clients is small. If the selected trimming level would remove too many client values, the aggregator falls back to a standard mean. This prevents the aggregation step from failing when there are not enough updates to perform trimming safely.
When the trimming parameter results in a positive number of removed values, the Trimmed Mean operation is applied as intended by sorting the values for each model coordinate and removing the lowest and highest values before averaging. If the trimming value rounds down to zero, no values are removed, and the aggregation becomes equivalent to standard averaging for that round. This makes the implementation stable across different numbers of participating clients while still applying trimming when enough client updates are available.
An important limitation of this implementation is that the trimming parameter must be chosen according to the expected number of malicious clients. The aggregator does not identify which clients are malicious. Instead, it removes a fixed number of extreme values from both sides for every coordinate. Therefore, if k is smaller than the number of malicious extreme values, some poisoned values may still remain in the aggregation. On the other hand, if k is chosen too large, honest client updates may also be removed, which can reduce useful learning information. In practice, this means that Trimmed Mean requires an estimate or upper bound of the malicious client proportion in order to choose a suitable value of β.
This implementation makes the aggregation more robust to poisoned updates with large magnitude, such as gradient boosting attacks. However, because trimming is performed coordinate-wise, it may be less effective against stealthier attacks where malicious updates remain inside the normal statistical range of benign updates, such as A Little Is Enough.

3.5.2 Multi-Krum

The Multi-Krum mitigation was implemented as a custom FEDn aggregator by extending the AggregatorBase class. The purpose of this aggregator is to reduce the influence of poisoned client updates by selecting only the client models that are closest to the majority of the received updates. This follows the Multi-Krum idea,

where client updates are compared using pairwise distances and the most consistent updates are selected for averaging [20].
In the implementation, all available client models are first loaded from the FEDn update queue. Each model is represented as a list of NumPy arrays, where each array corresponds to one layer or parameter tensor. Before computing distances, each model is flattened into one long vector by concatenating all parameter arrays:

$$v_i = \text{concat}\left(w_{i,1}, w_{i,2}, \ldots, w_{i,L}\right)$$

where v_i is the flattened model from client i, $w_{i,l}$ is layer l of client i, and L is the number of model layers. This makes it possible to compare complete client models using one distance value.
The squared Euclidean distance is then computed between every pair of client models:

$$d(i,j) = \|v_i - v_j\|_2^2$$

For each client model i, the distances to all other client models are sorted. The Multi-Krum score is computed as the sum of the distances to the closest $n - f - 2$ neighbours:

$$s_i = \sum_{j \in N_i} d(i,j)$$

where n is the number of received client models, f is the assumed number of Byzantine clients, and N_i is the set of the $n - f - 2$ closest neighbours to client i. A low score means that the update is close to many other updates and is therefore considered more reliable. A high score means that the update is far from the majority and may be malicious or abnormal.
After the scores have been computed, the aggregator selects the models with the lowest scores. In manual mode, the number of Byzantine clients f and the number of selected models m can be passed as parameters. If no value for m is provided, the implementation selects

$$m = n - f$$

The selected models are then averaged layer by layer:

$$w_l^{r+1} = \frac{1}{m} \sum_{i \in S} w_{i,l}$$

where S is the set of selected client models with the lowest Multi-Krum scores. This produces the new global model using only the updates that are closest to the majority.
The implementation also includes an automatic selection mode. When no manual values are provided, the aggregator computes the Krum scores and then applies an interquartile range (IQR) rule to detect unusually high scores. The first quartile Q_1, third quartile Q_3, and inter-quartile range are computed as

$$IQR = Q_3 - Q_1$$

The outlier threshold is then defined as

$$T = Q_3 + 1.5 \cdot IQR$$

Client models with scores below or equal to this threshold are treated as valid:

$$S = \{i \mid s_i \leq T\}$$

This automatic mode was added so that the aggregator can estimate how many client updates should be kept without always requiring a fixed value of m.
Finally, the selected client models are averaged. Compared with Trimmed Mean, Multi-Krum performs vector-wise filtering instead of coordinate-wise filtering. This means that it evaluates the full client model update as one object rather than removing extreme values independently for each parameter. This makes it useful against attacks where an entire client update is inconsistent with the majority. However, the manual version requires an estimate of the number of malicious clients f. If f is chosen too low, malicious updates may still be selected. If it is chosen too high, useful honest updates may be removed. The automatic IQR mode reduces this dependency by estimating outliers from the score distribution, but it is still a heuristic and may not detect stealthy attacks whose updates remain close to the benign clients.

3.5.3 FoolsGold

The FoolsGold mitigation was implemented as a custom FEDn aggregator by extending the `AggregatorBase` class. The purpose of this aggregator is to reduce the influence of colluding malicious clients, also known as sybil clients, by comparing the similarity of their historical updates. FoolsGold is based on the observation that malicious clients working toward the same attack objective often produce similar updates over time, while honest clients are expected to show more diverse update directions [22].
In contrast to FedAvg, which gives all received client updates equal influence, FoolsGold assigns a client-specific weight α_i to each update. Clients whose historical updates are highly similar to other clients receive a lower weight, while clients with more unique update histories keep a higher weight. In this implementation, the client history is stored persistently inside the aggregator:

```
self.history = {}       # {client_name: np.ndarray}
self.round_counter = 0
```

Each client is identified by its sender name. This is important because FoolsGold depends on tracking the same client across multiple communication rounds. If a client changes identity between rounds, the historical similarity information would be lost.
The aggregator first loads all client models from the FEDn update queue. Since the client submissions are full model weights, the implementation computes a reference model by taking the simple mean of the received models in the current round:

$$\bar{w}^{(t)} = \frac{1}{n} \sum_{i=1}^{n} w_i^{(t)},$$

where n is the number of received client models and $w_i^{(t)}$ is the model submitted by client i. This reference model is used to compute client deltas for the FoolsGold similarity analysis.
FoolsGold does not need to compare the full model when computing similarity. Instead, the implementation extracts an indicative layer from each client model. By default, the penultimate layer is used:
`self.layer_index = -2`
For each client, the update on this selected layer is computed as

$$g_i^{(t)} = w_{i,l}^{(t)} - \bar{w}_l^{(t)}$$

where l is the selected layer. The update is then flattened into a vector and added to the cumulative history of that client:

$$H_i^{(t)} = H_i^{(t-1)} + g_i^{(t)}$$

After constructing the history matrix H, the aggregator computes the pairwise cosine similarity between all client histories:

$$CS(i,j) = \frac{H_i \cdot H_j}{\|H_i\|_2 \|H_j\|_2}$$

A high cosine similarity means that two clients have followed a similar update direction over time. The implementation clips the cosine similarity values to the range $[-1, 1]$ and sets the diagonal values to zero so that a client is not compared with itself.
The next step is the pardoning mechanism. Pardoning reduces the penalty for a client that is somewhat similar to another client but is not as suspicious overall. This is used to avoid punishing honest clients that may naturally have some similarity to a malicious client. If client i has lower maximum similarity than client j, then the similarity from i to j is scaled down:

$$CS(i,j) \leftarrow CS(i,j)\frac{\max(CS_i)}{\max(CS_j)}$$

After pardoning, the aggregator computes the FoolsGold weight α_i for each client. First, each client receives a raw weight based on its highest similarity to any other client:

$$\alpha_i = 1 - \max_j CS(i,j)$$

This means that a client with very high similarity to another client receives a low weight. The weights are then clipped to $[0, 1]$, rescaled by the maximum weight, and transformed using a log function with a confidence shift κ:

$$\alpha_i = \log\left(\frac{\alpha_i}{1-\alpha_i}\right) + \kappa$$

In the implementation, κ is set to 0.5 by default:

$$\kappa = 0.5$$

Finally, the client models are aggregated using the FoolsGold weights. For each layer, the implementation computes a weighted delta from the reference model and then adds the average weighted delta back to the reference model:

$$w_l^{(t+1)} = \bar{w}_l^{(t)} + \frac{1}{n} \sum_{i=1}^{n} \alpha_i \left(w_{i,l}^{(t)} - \bar{w}_l^{(t)} \right)$$

Overall, the implemented FoolsGold aggregator can be summarized as follows:

1. Load all client model updates.
2. Compute a reference model from the received client models.
3. Extract the update from an indicative layer for each client.
4. Add each client's indicative update to its cumulative history.
5. Compute pairwise cosine similarity between client histories.
6. Apply pardoning to reduce false penalties on less suspicious clients.
7. Compute a client weight α_i from the maximum similarity.
8. Aggregate all client models using the FoolsGold weights.

This implementation is mainly designed to mitigate colluding or sybil-style attacks, where several malicious clients repeatedly move the model in a similar direction. It is therefore relevant against coordinated attacks such as A Little Is Enough. However, FoolsGold is less suitable against a single malicious client, since one attacker alone does not create the same similarity pattern. It also depends on stable client identities across rounds, because the defense relies on cumulative historical updates.

3.5.4 Layered Aggregator

The layered aggregator was evaluated as a defense-in-depth strategy against poisoning attacks. Unlike a single robust aggregation rule, which normally makes one decision based on the received client updates, the layered approach processes the updates through several consecutive stages before producing the model used in the next communication round. The purpose of this design is to reduce the dependency on a single assumption about how malicious behavior appears.

At a conceptual level, layered aggregation can make use of several types of information extracted from client updates. These include update magnitude, the relative distance between client updates, similarity in update direction, historical similarity between clients, and the behavior of the resulting model during validation. Not all of these signals necessarily need to be used simultaneously. Rather, they illustrate the different types of information that can be considered when identifying suspicious or inconsistent update behavior.

The first stage of the aggregation process is intended to reduce the influence of updates that strongly deviate from the majority of the participating clients. Such behavior can occur in attacks that produce unusually large or clearly displaced model updates, such as Gradient Boosting. Distance- and magnitude-related information can therefore be used to identify updates that are inconsistent with the main group of client contributions.

After this initial filtering stage, the remaining updates can be evaluated using similarity-oriented information. In particular, repeated similarity between updates from multiple clients can indicate coordinated behavior. This is relevant for attacks

such as A Little Is Enough (ALIE), where malicious clients attempt to remain within a statistically plausible update range instead of creating obvious outliers. Historical information can therefore provide an additional signal that may not be visible when only the current communication round is considered.
Validation behavior may also provide an additional indication of whether an update, or a set of aggregated updates, produces behavior that is inconsistent with the expected learning process. For example, a model that appears statistically similar to other client updates may still cause an unusual change in validation performance. Such information can be used as an additional signal in a layered defense, particularly when update-level statistics alone are insufficient.
Conceptually, the aggregation process can therefore be represented as
Client updates $\rightarrow$ *update-level screening* $\rightarrow$ *similarity and behavioral assessment* $\rightarrow$ *filtering or weighting* $\rightarrow$ *aggregated model update.*
The intermediate stages do not necessarily classify a client directly as malicious. Instead, they reduce, remove, or reweight contributions that appear inconsistent according to the information available at each stage. The final output is a single aggregated model update that is used to construct the server model for the following communication round.
In the implementation evaluated in this thesis, the layered strategy combines multiple robustness principles, including distance-based filtering and similarity-based weighting. This allows the defense to address different attack characteristics within the same aggregation pipeline. Large abnormal updates can be treated differently from coordinated updates that remain close to the normal update distribution.
Since this thesis was conducted in collaboration with Cognivity AI, implementation-specific details of the layered aggregation strategy, including internal thresholds, parameter values, and parts of the decision logic, are not disclosed. The layered aggregator is therefore evaluated as a proprietary aggregation strategy rather than presented as a fully reproducible aggregation algorithm. The description provided in this section is intended to explain the conceptual behavior of the defense and the types of signals involved, while the experimental evaluation focuses on its observable effect under the different attack scenarios considered in this thesis.

3.6 Suspicious-Client Risk Scoring

In addition to the implemented attacks and mitigation methods, a server-side suspicious-client risk scoring mechanism was developed to support post-training analysis. The purpose of this mechanism is not to prove that a client is malicious. Instead, it ranks clients according to how unusual their behavior appears during federated training. This makes the output useful as a triage tool, where high-ranked clients can be inspected further.
The mechanism is implemented as an offline analysis pipeline that uses model-update fingerprints saved during training. For each client and communication round, the client-side debug artifacts contain the model received from the server before local training and the model returned after local training.

$$\Delta W_i^{(r)} = W_{i,\text{out}}^{(r)} - W_{\text{in}}^{(r)},$$

where $W_{\text{in}}^{(r)}$ is the model received by client i at round r, and $W_{i,\text{out}}^{(r)}$ is the model returned by the client after local training. The update $\Delta W_i^{(r)}$ is used as the main source of evidence for the risk scoring.

Several types of signals are used in the scoring process. Magnitude-based features measure how large a client's update is compared with the model it received. Direction-based features measure whether the update moves in an unusual direction compared with the incoming model or with other clients. Layer-based features describe whether most of the update energy is concentrated in a small part of the model. Peer-similarity features compare clients within the same round and are useful for identifying clients whose updates are unusually close to each other. The mechanism also records invalid or zero-update behavior, since repeated missing, corrupted, or near-empty updates may indicate unstable or suspicious participation.

When response-time information is available, it is included as an additional signal. The response-time features measure how long each client takes to return its result to the server. A client with unusually high response time, or response-time behavior that differs strongly from the majority, can receive an additional response-time score. If no timing data is available, these values are left empty and the response-time contribution is set to zero. The machine-learning attack probability is an optional supervised signal used when the identities of the malicious clients are known in the controlled experiment. Each client-round observation is labelled as malicious or benign according to the known client identity. A Random Forest classifier with 500 trees is then trained using the extracted update-signature features.

To reduce dependence on a single train–test split, the implementation uses stratified cross-validation with up to five folds. For each client-round observation, an out-of-fold attack probability is obtained using the classifier's predicted class probability. At client level, these probabilities are averaged across the available rounds to obtain the mean attack probability used by the risk-scoring mechanism.

The cross-validation is performed at round level rather than by holding out complete clients. The resulting probability should therefore be interpreted as an auxiliary signal for the controlled analysis rather than as evidence that the classifier generalizes to previously unseen clients. The main metrics used by the risk-scoring mechanism are summarized in Table 3.3.

Table 3.3: Main metrics used in the suspicious-client risk scoring mechanism.

Metric	Meaning
`mean_delta_ratio`	Average size of the client's update compared with the model it received.
`max_delta_ratio`	Largest update ratio produced by the client in any round.
`mean_global_delta`	Average absolute size of the client's model update.
`max_global_delta`	Largest update magnitude produced by the client.
`invalid_round_fraction`	Fraction of rounds where key update values were missing or invalid.
`zero_update_round_fraction`	Fraction of rounds where the client returned almost no effective update.
`near_identical_peer_count`	Number of peers whose updates were almost identical to the client's update.
`mean_nearest_delta_cosine_similarity`	Average cosine similarity between the client update and its closest peer.
`mean_nearest_relative_l2_distance`	Average relative distance between the client update and its closest peer.
`mean_val_loss`	Average validation loss associated with the client's rounds.
`mean_attack_probability`	Optional supervised probability score when known attacker labels are provided.
`mean_response_time`	Average time taken by the client to return its result.
`max_response_time`	Maximum response time observed for the client.

The scoring process combines these signals into two main components. The first

component is an anomaly score, which is based on robust positive z-scores of suspicious update behavior. This allows the mechanism to focus on clients whose values are unusually high compared with the rest of the federation, without relying only on the standard mean and standard deviation. The second component is the response-time score, which is added when timing information is available.
The final client risk score is computed as

$$\text{risk\_score}_i = \text{anomaly\_score}_i + \text{probability\_score}_i + \text{response\_time\_score}_i.$$

A client is then flagged if its risk score is above a predefined risk threshold τ_{risk}:

$$\text{flagged}_i = \begin{cases} 1, & \text{risk\_score}_i > \tau_{\text{risk}}, \\ 0, & \text{risk\_score}_i \leq \tau_{\text{risk}}. \end{cases}$$

In the current implementation, the threshold is set to

$$\tau_{\text{risk}} = 4.$$

This means that the implementation does not simply mark the top N clients as attackers. Instead, clients are ranked by risk score, and only clients whose score exceeds τ_{risk} are flagged as suspicious. The threshold should be interpreted as an experimental heuristic rather than a formal proof of malicious behavior.
A limitation of this mechanism is that it depends on access to individual client updates. This is useful for forensic analysis, but it also creates a privacy trade-off. If secure aggregation is used in a way that hides individual client updates from the server, then this type of client-level risk scoring becomes limited or impossible. Therefore, the method is most suitable for controlled evaluation, debugging, and security analysis settings where individual update visibility is available.

4
Results

4.1 Baseline Performance in the FEDn Hierarchy Without Attack

The baseline experiment was run with 12 normal clients and FedAvg aggregation, without any malicious clients. As shown in Figure 4.1, both the training and validation losses decrease over the communication rounds. The loss drops quickly during the early rounds and then becomes more stable after approximately round 50.
The training and validation curves follow a similar trend, which indicates that the model is learning normally and does not show clear signs of overfitting. Some oscillations appear during training, which is expected in a federated setup with heterogeneous battery data. This baseline is used as the reference for comparing the later attack and mitigation experiments.

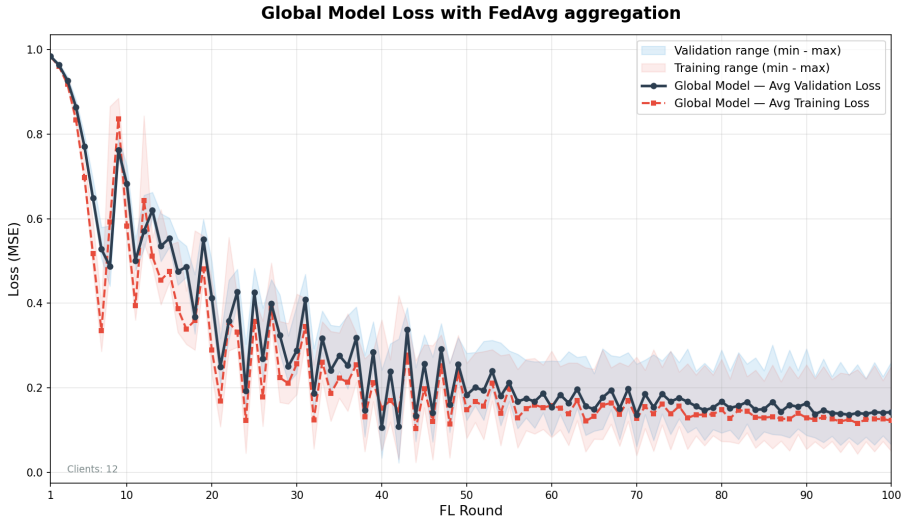


Figure 4.1: Baseline training with 12 normal clients and 0 poisoned clients using FedAvg aggregation in the FEDn hierarchy.

4.2 Attack Impact in FEDn

This section examines the behavior of the FEDn hierarchy when the implemented attacks are introduced. The experiments are first evaluated without additional mitigation mechanisms in order to provide a clear baseline for how the system reacts

under attack. Since the selected attacks influence the training process in different ways, their effects are not always equally visible. Some attacks cause clear disturbances in the loss and model performance, while others have a more gradual or subtle impact. These results are therefore used as a reference point for the later comparison with AICFL and the mitigation methods.

4.2.1 Gradient Boosting Attack

The gradient boosting attack was tested with 10 normal clients and 2 poisoned clients using FedAvg aggregation. As shown in Figure 4.2, the loss remains relatively stable during the early rounds, but starts to increase after the attack begins to dominate the aggregation process. Compared with the baseline, the model no longer converges to a low and stable loss. Instead, the training loss increases strongly in the later rounds, and the validation loss also becomes unstable with large spikes. This shows that even two poisoned clients can significantly damage FedAvg when the malicious update is amplified with a large boost factor. The result confirms that FedAvg is highly sensitive to large model-poisoning updates because it averages all client contributions without filtering abnormal updates.

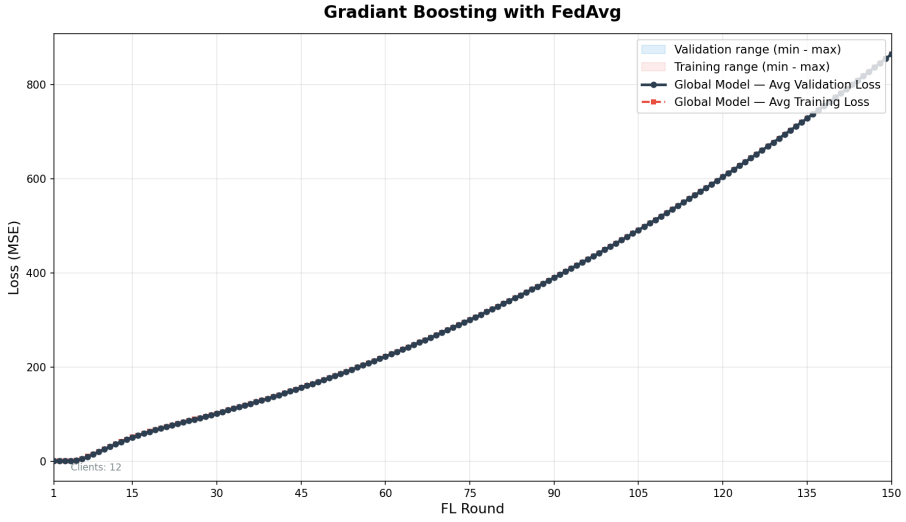


Figure 4.2: Gradient Boosting attack with 10 normal clients and 2 poisoned ($\lambda = 10$) clients using FedAvg aggregation in the FEDn hierarchy.

4.2.2 A Little Is Enough

The A Little Is Enough attack was tested with 8 normal clients and 4 poisoned clients using FedAvg aggregation. As shown in Figure 4.3, the global model loss still decreases over the communication rounds, which means that the attack does not completely stop convergence.
However, compared with the baseline, the loss curve shows more instability and larger oscillations during training. This is expected because the poisoned clients send crafted updates that remain close to normal updates while still shifting the aggregation direction. The attack, therefore, affects the stability of the learning

process, but in this experiment, it does not cause the same strong divergence as a large boosted update.

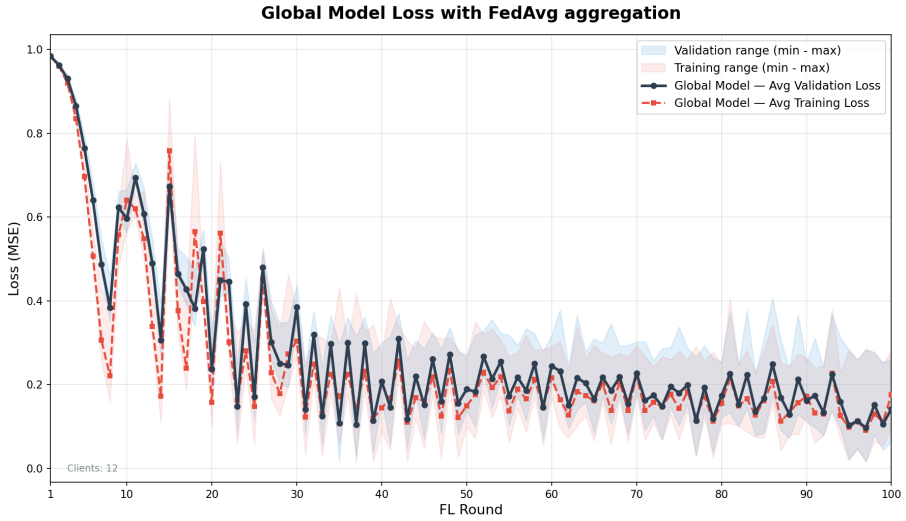


Figure 4.3: A Little Is Enough attack with 8 normal clients and 4 poisoned clients using FedAvg aggregation in the FEDn hierarchy.

4.2.3 Backdoor Attack

The backdoor attack was evaluated in the FEDn setup using distributed battery time-series data, where each client holds local battery trajectories and the model predicts the corresponding SOH curve from concatenated voltage, temperature, and current measurements. Since the clients are constructed from different battery sources and operating conditions, the local datasets are heterogeneous. The reported losses therefore reflect both the global optimisation behavior of the federated model and the variability introduced by differences between client datasets.

Figure 4.4 shows the training dynamics of the FEDn model during the undefended backdoor experiment. The solid curves represent the round-wise average training and validation losses, the shaded regions indicate the minimum-to-maximum range of local client losses observed in each round, and the dashed curve shows the validation loss of one representative client.

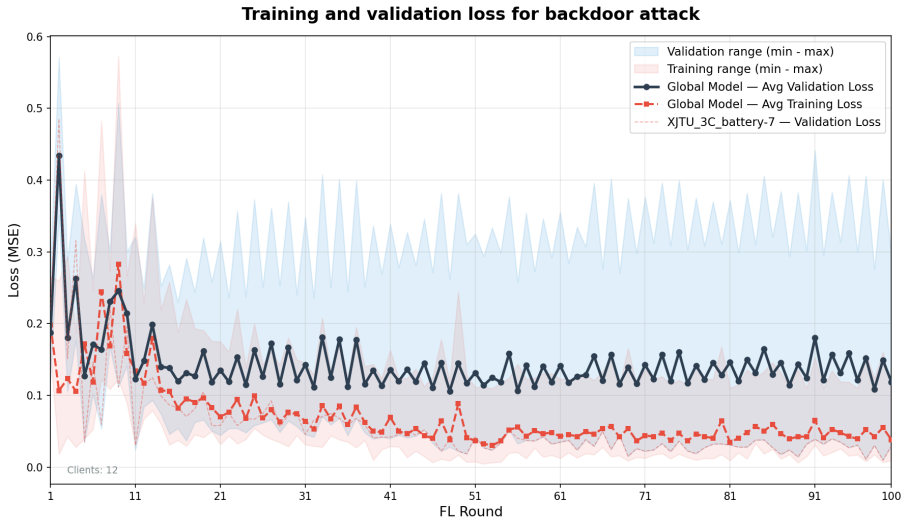


Figure 4.4: Global model training and validation loss during the FEDn backdoor experiment. The shaded areas show the minimum and maximum local client losses observed in each round, the solid lines show the round-wise average global-model training and validation losses, and the dashed line shows the validation loss of a representative client.

The loss curves indicate that the federated model converges during training. The largest reduction in loss occurs in the initial communication rounds, after which both curves flatten and fluctuate within a narrower range. This suggests that most of the useful parameter updates are incorporated early in training, whereas later rounds mainly adjust the model around a more stable solution. The persistent gap between the training and validation curves shows that the model fits the local training data more closely than the validation data, but the absence of strong late-stage divergence indicates that the optimisation process remains stable. In addition, the broad client-wise loss ranges demonstrate that the participating clients do not contribute equally, which is expected in this battery setting because the local datasets originate from different battery families and operating conditions.

However, Figure 4.4 mainly illustrates convergence behavior and client-level variability during training. It does not, by itself, determine whether the backdoor has successfully influenced the global model. A backdoor attack can remain difficult to detect from loss values alone, because the model may still converge normally on clean data while producing attacker-controlled outputs when the trigger is present. For this reason, the actual backdoor effect is evaluated separately using clean and triggered backdoor metrics.

The backdoor evaluation is performed at two levels. First, the malicious client is evaluated locally to confirm that the dual-objective training procedure produces a backdoored local model. Second, and more importantly, the globally aggregated model returned by the server is evaluated on clean and triggered inputs. This global evaluation determines whether the backdoor survives aggregation and remains active in the federated model after combining malicious and benign client updates.

4.3 Baseline Performance in the AICFL Hierarchy Without Attack

The baseline AICFL experiment was performed with 12 normal clients and no malicious participation. As shown in Figure 4.5, the validation loss drops sharply in the first rounds, from approximately 0.55 to around 0.16, before becoming more stable. The figure also shows the adaptive cluster behavior of AICFL. A new cluster is created in round three with four clients, but it is removed in the fourth round. Another cluster is then created with four clients and a validation loss of around 0.27. In the later rounds, more clients migrate toward the cluster with the lower validation loss.
This indicates that, in the normal setting, AICFL behaves as intended; clients are not fixed to one model but move toward the cluster that gives better local validation performance. This baseline is therefore used as a reference for the later attack experiments.

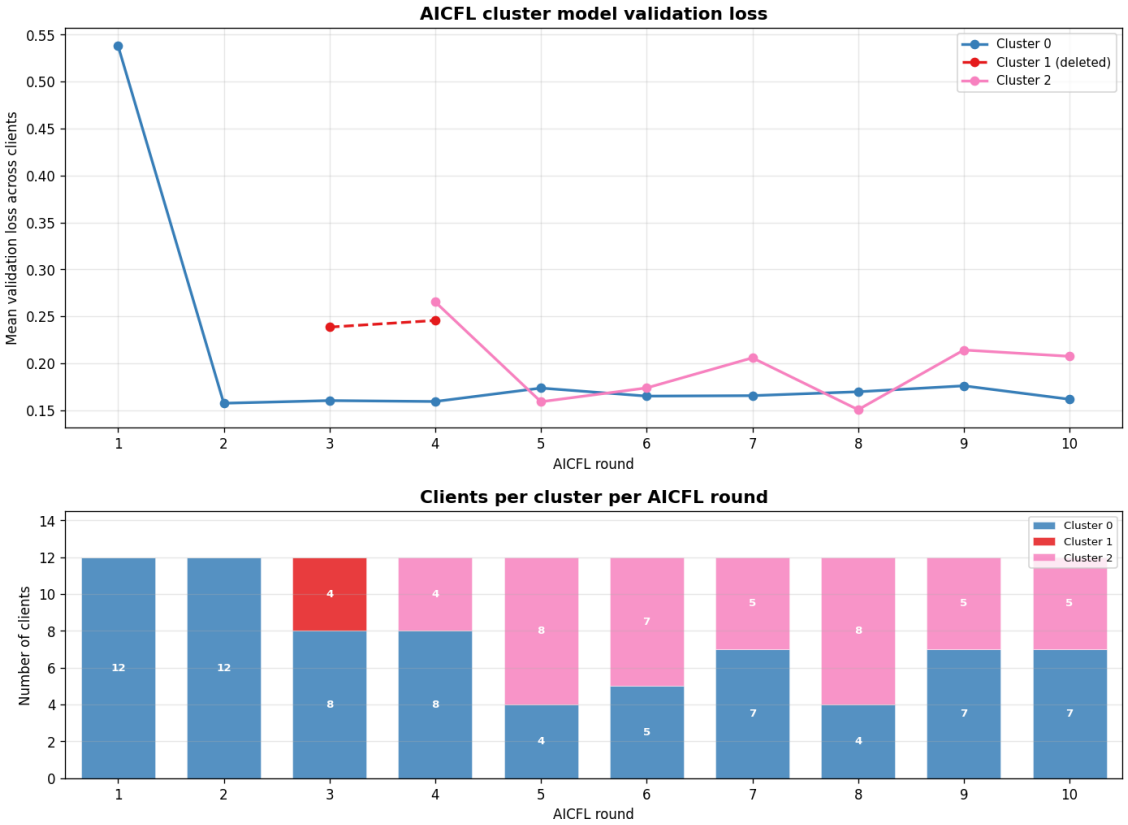


Figure 4.5: Baseline AICFL training with 12 normal clients and 0 poisoned clients in the hierarchical clustered topology.

4.4 Attack Impact in AICFL

This section evaluates the behavior of the AICFL setup under malicious client participation. The aim is to compare the normal clustered training process with three attack scenarios: Gradient Boosting, A Little Is Enough, and Backdoor Attack. Since AICFL uses a hierarchical and cluster-based training structure, the experiments examine not only whether poisoned updates affect the model performance, but also whether the attacks influence the stability of the cluster behavior. The following subsections first present the baseline without attacks and then describe the impact of each attack in the AICFL setting.

4.4.1 Gradient Boosting Attack

The Gradient Boosting attack was tested in the AICFL setup with 10 normal clients and 2 malicious clients. As shown in Figure 4.6, the attack has a strong effect during the early rounds. At the beginning of the experiment, AICFL selected one or both malicious clients for training, which caused the validation loss to increase sharply from round one to round two, reaching approximately 1300. In the following round, the normal clients formed a new cluster with a much lower validation loss. This improved performance led AICFL to remove the poorly performing cluster 0 and continue with the newly created cluster 1. As a result, the loss decreased again once the malicious clients were no longer selected for training.
Later, in round 10, another cluster was created with eight clients. At this stage, the malicious clients were split between the new and old clusters. However, they were still not selected for training because their earlier loss values were significantly worse than those of the normal clients.
Overall, the result shows that Gradient Boosting can strongly disturb AICFL when malicious clients are selected early in the training process. At the same time, the cluster creation, deletion, and client-selection behavior shows that AICFL can reduce the influence of clearly harmful clients when their updates lead to poor validation performance.

4.4.2 A Little Is Enough

The A Little Is Enough attack was tested in the AICFL setup with 8 normal clients and 4 malicious clients. As shown in Figure 4.7, the validation loss drops sharply at the beginning, from around 0.43 to approximately 0.12 in round two. Unlike Gradient Boosting, ALIE does not create a large loss spike since the malicious updates are designed to remain close to the normal update range.
After the first rounds, the validation loss varies between the active clusters, especially around rounds five to eight. The client distribution also changes several times, and new clusters are created and removed during the experiment. The malicious-client migration shows that the poisoned clients are not isolated in one cluster, but move between different clusters during training.
Overall, ALIE does not strongly break the AICFL training process, but it makes the cluster behavior less stable. Compared with Gradient Boosting, the attack is

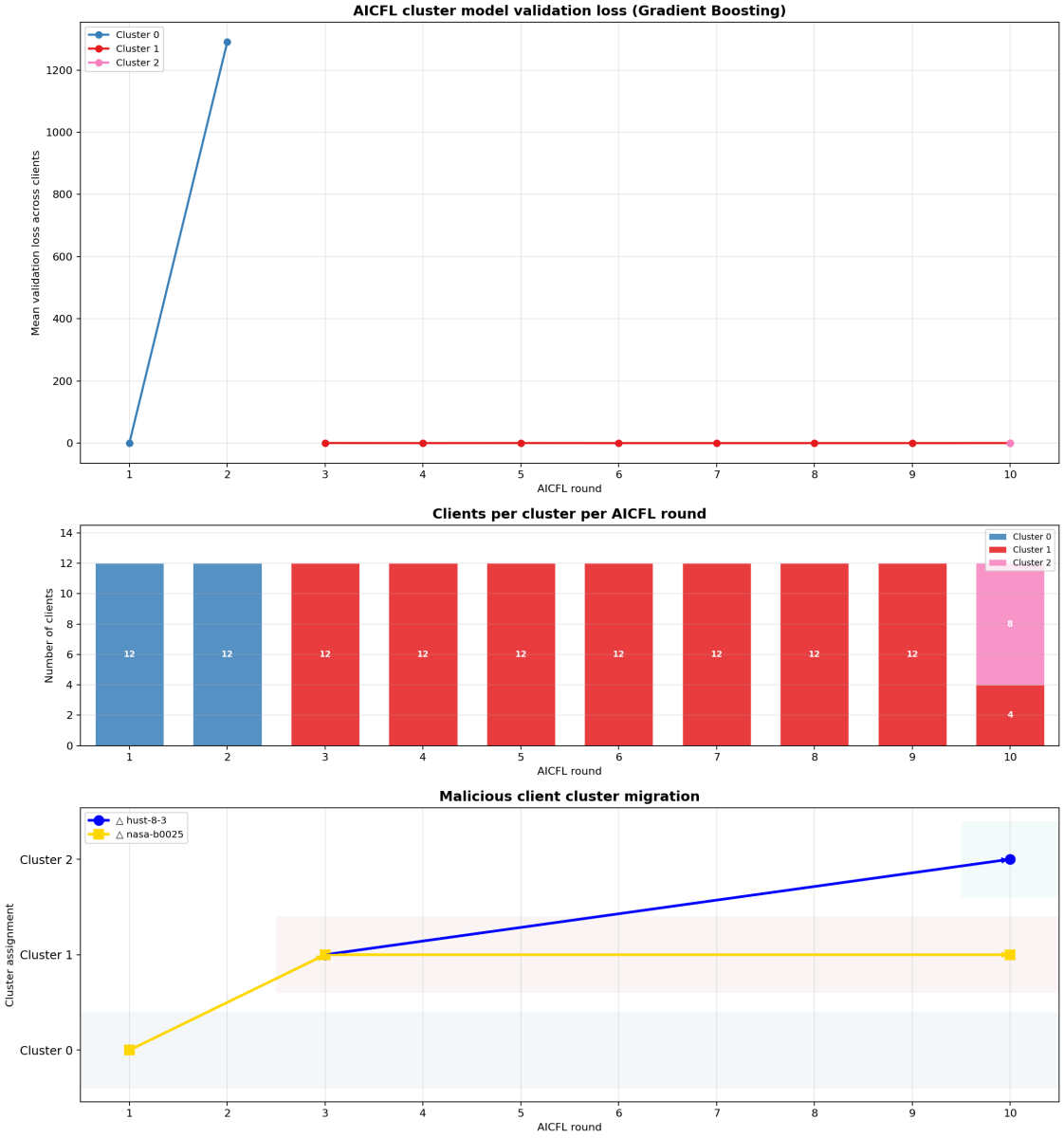


Figure 4.6: Gradient Boosting attack with 10 normal clients and 2 poisoned clients in the hierarchical clustered topology.

less visible in the loss values, but it can still influence client assignment and cluster creation over time.

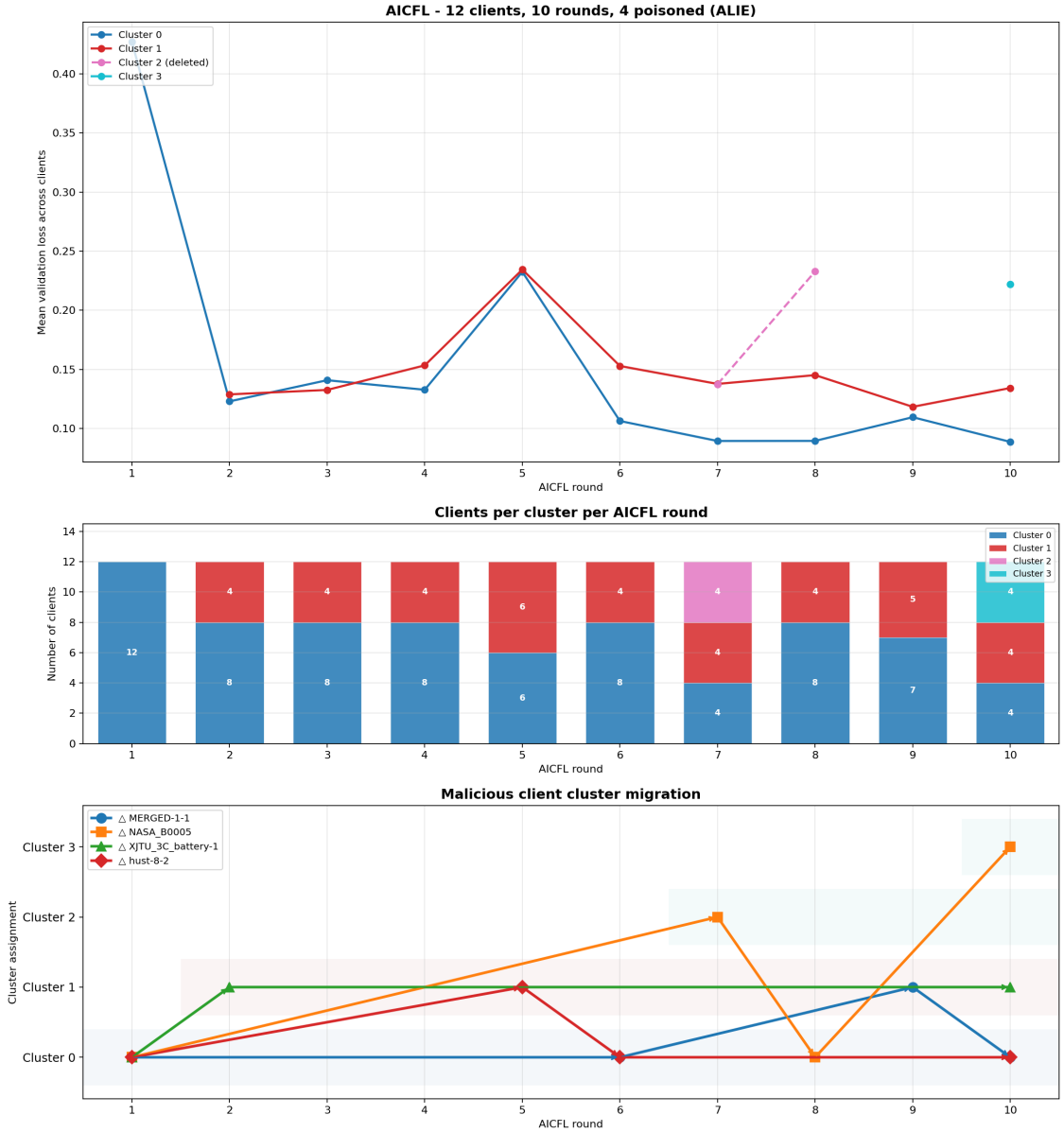


Figure 4.7: A Little Is Enough attack with 8 normal clients and 4 poisoned clients in the hierarchical clustered topology.

4.4.3 Backdoor Attack

Figure 4.8 illustrates the behavior of the AICFL system during the backdoor experiment with 10 benign clients and 2 malicious clients. In contrast to the FEDn results, where the main focus is on the single global model, the AICFL analysis also considers how the clustered training structure evolves under attack. The figure therefore shows not only validation loss trends, but also cluster membership changes and the migration of the malicious clients across the cluster lifecycle.

The upper panel shows the mean validation loss for each cluster over the AICFL rounds. Several clusters are created and later deleted as part of the adaptive cluster lifecycle. Early clusters, such as Cluster 0 and Cluster 1, exhibit relatively high and unstable validation losses before they are removed. A similar pattern can be observed for Cluster 2 and Cluster 4, which also show elevated losses before deletion. In contrast, Cluster 3 remains active for several consecutive rounds and maintains a comparatively low and stable validation loss, around 0.14 to 0.16. At the end of the process, Cluster 5 appears with a similarly low validation loss. This indicates that, under the backdoor scenario, AICFL continues to restructure the client population and eventually converges toward a smaller number of more stable clusters.

The middle panel shows the number of clients assigned to each cluster per AICFL round. In the early rounds, all 12 clients remain grouped in a single cluster. As training proceeds, the client population is split into multiple clusters. For example, at Round 5 the clients are divided between Cluster 1 and Cluster 2, and later between Cluster 3 and Cluster 4, before the final split between Cluster 3 and Cluster 5. This confirms that the adaptive clustering process remains active under attack and that the backdoor does not simply collapse the system into one unstable global grouping.

The lower panel tracks the migration of the two malicious clients over the cluster lifecycle. Both malicious clients initially follow the same path, starting in Cluster 0 and then moving through Cluster 1 and Cluster 2. At later rounds, both appear in Cluster 3, after which one malicious client remains in Cluster 3 while the other moves further into Cluster 4 and finally Cluster 5. This behavior shows that the malicious clients do not remain permanently assigned to the same cluster throughout the experiment. Instead, their cluster assignments change as the adaptive clustering process evolves.

Taken together, the three panels show that AICFL continues to reorganize its cluster structure during the backdoor experiment. The malicious clients do not remain permanently assigned to the same cluster, and several clusters are created and removed as the system adapts. The later emergence of more stable clusters shows that the adaptive cluster-management process continues despite the malicious updates.

However, these observations only describe the effect of the attack on the clustering behavior. The cluster plot alone does not show whether the backdoor behavior was removed, contained, or remained active in the cluster models. Since trigger-based backdoor metrics were not evaluated for the AICFL models in this experiment as it was done in the FEDn framework, no conclusion about backdoor containment is drawn from the cluster behavior alone.

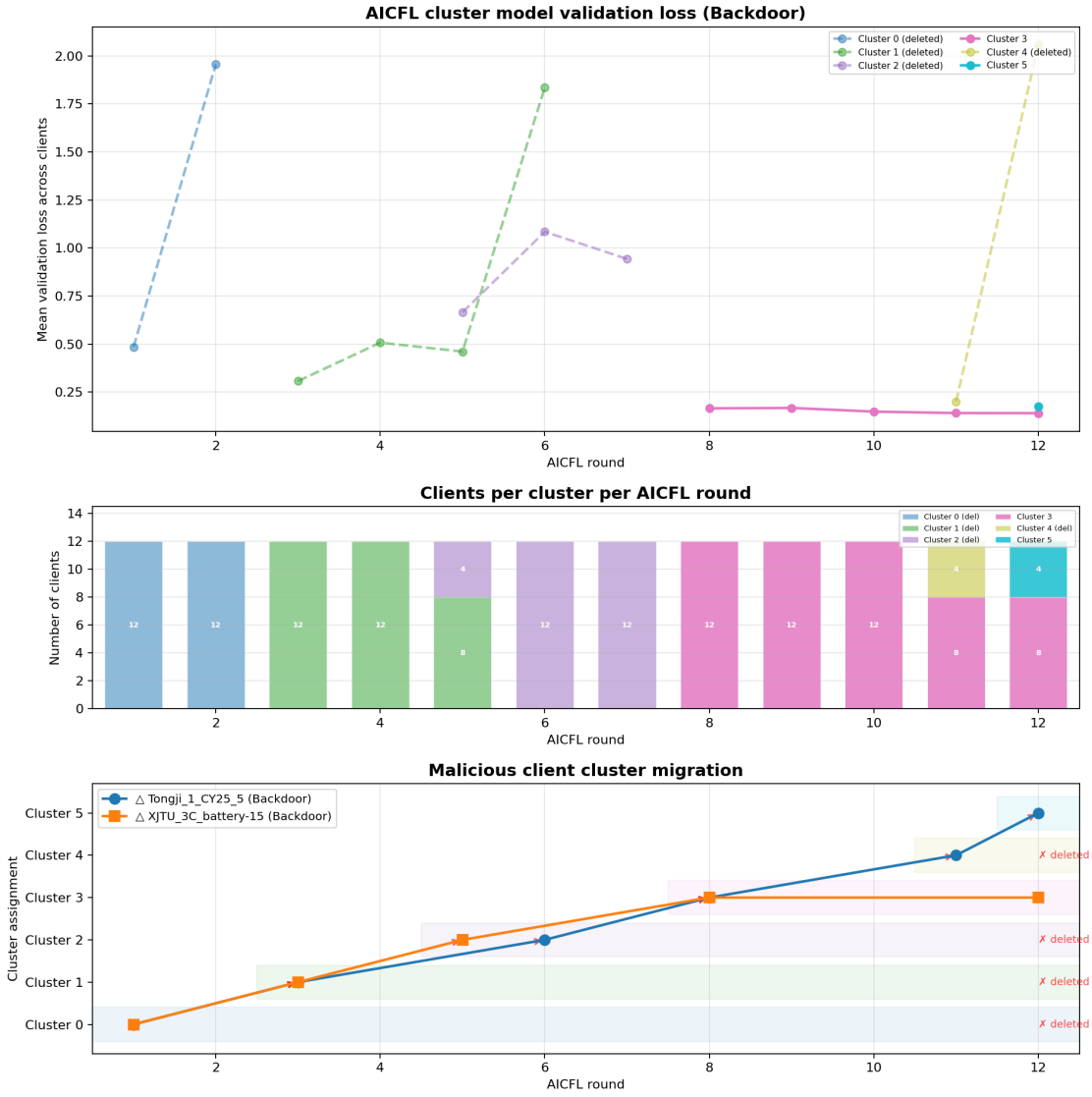


Figure 4.8: Backdoor attack with 11 benign clients and 2 malicious clients in the hierarchical clustered topology.

4.5 Suspicious-Client Risk Scoring

This section presents a representative experimental output from the suspicious-client risk scoring mechanism described in Section 3.6. The purpose of this analysis is to rank clients according to suspicious behavior observed during training. The mechanism should not be interpreted as proving that a client is malicious. Instead, it provides a client-level indication of which participants should be inspected further. Table 4.1 presents the client-level output obtained from the evaluated ALIE scenario. The table includes both update-based and timing-based indicators. The two ALIE clients receive higher final risk scores than the normal clients. Their ranking is associated with several signals, including larger update magnitudes, repeated near-identical peer behavior, higher response times, and higher supervised attack

probabilities.

Table 4.1: Suspicious-client risk-scoring output for the evaluated ALIE scenario.

Client	Type	Mean Δ ratio	Max global Δ	Near-identical peer rounds	Mean response (s)	ML attack prob.	Final risk score	Decision
xjtu-3c-6	ALIE attacker	0.084	5.90	8/10	18.42	0.93	15	Flag
nasa-b0025	ALIE attacker	0.079	5.61	8/10	15.87	0.88	11	Flag
hust-8-2	Normal	0.021	1.42	0/10	6.12	0.09	3	OK
mit-bat2	Normal	0.019	1.31	0/10	5.98	0.06	2	OK
xjtu-3c-3	Normal	0.024	1.54	1/10	6.45	0.11	2	OK
nasa-b0007	Normal	0.018	1.20	0/10	5.76	0.04	1	OK

In this example, the two ALIE clients receive the highest final risk scores and exceed the flagging threshold $\tau_{\text{risk}} = 4$. Their ranking is supported by several signals rather than by a single metric, including larger model-update magnitudes, repeated similarity to peer updates, higher response times, and higher supervised attack probabilities. The normal clients remain below the threshold in this scenario.
These results should be interpreted as suspiciousness ranking rather than as a general detection guarantee. In particular, the supervised probability uses known attacker identities from the controlled experiment, and the threshold is a fixed heuristic rather than a statistically calibrated decision boundary.
Using the fixed experimental threshold $\tau_{\text{risk}} = 4$ defined in Section 3.6, the two ALIE clients exceed the flagging threshold with final risk scores of 15 and 11. The normal clients remain below the threshold, with final scores between 1 and 3.
The two ALIE clients have final risk scores of 15 and 11, so they are flagged as suspicious. The normal clients have final risk scores between 1 and 3, so they are not flagged. This supports the intended use of the mechanism as a suspicious-client ranking method.
The near-identical peer rounds are esp ecially relevant for the ALIE attack. Since ALIE is a coordinated poisoning attack, malicious clients can produce updates that are similar to each other while still avoiding very large outlier behavior. In the example, both ALIE clients have near-identical peer behavior in 8 out of 10 rounds. This makes peer similarity an important signal for this attack type.
The response-time signal also contributes to the final ranking. The flagged clients have noticeably higher mean response times than the normal clients. This does not prove malicious behavior on its own, since slow response can also come from system load or network delay. However, when it appears together with abnormal update behavior and high peer similarity, it strengthens the suspicion score.
Overall, the results show that the implemented mechanism can provide useful post-training evidence about suspicious client behavior. It is most useful when interpreted together with the attack setup, model-loss behavior, and mitigation results. The output should therefore be treated as a forensic ranking tool rather than a standalone proof that a specific client is malicious.

4.6 Mitigation Performance in FEDn

This section evaluates how different aggregation-based mitigation methods perform in the standard FEDn setup. The aim is to study whether replacing FedAvg with

more robust aggregation can reduce the influence of poisoned clients during training. The experiments focus on how each mitigation method affects the stability of the loss curve and whether the global model can continue learning under malicious participation.

4.6.1 Trimmed Mean

The Trimmed Mean mitigation was tested against the gradient boosting attack with 11 normal clients and 1 poisoned client. As shown in Figure 4.9, the global model loss decreases over the rounds and remains stable throughout the experiment.
Compared with the same attack under FedAvg, the loss does not diverge, and the large increase seen in Figure 4.2 is avoided. This indicates that Trimmed Mean was able to reduce the influence of the boosted malicious update by removing extreme coordinate values before averaging.
The training and validation losses still show some oscillations, especially during the earlier rounds, but the overall trend is close to the baseline. This suggests that Trimmed Mean is effective against large-magnitude gradient boosting updates in this setup, although it may not be sufficient against more stealthy attacks that stay within the normal update range.

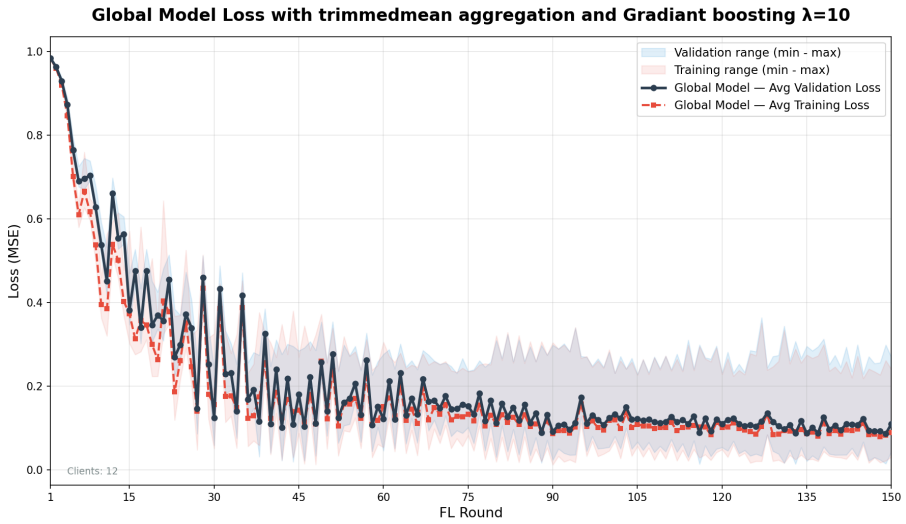


Figure 4.9: Gradient Boosting attack with 11 normal clients and 1 poisoned client using Trimmed Mean aggregation in the FEDn hierarchy.

4.6.2 Multi-Krum

The Multi-Krum mitigation was evaluated against both A Little Is Enough and Gradient Boosting attacks. In the first experiment, 4 poisoned clients used the A Little Is Enough attack for 150 rounds. As shown in Figure 4.10, the loss decreases during the early rounds, but the curve remains unstable and shows repeated oscillations later in training. This indicates that Multi-Krum does not reduce the effect of the ALIE attack efficiently in this setup.
The reason is that ALIE is designed to keep poisoned updates close to the normal

statistical range of client updates. Since Multi-Krum mainly selects updates based on distance from the majority, stealthy poisoned updates may still be selected if they do not appear as clear outliers.

In the second experiment, Multi-Krum was tested against Gradient Boosting with 2 poisoned clients. As shown in Figure 4.11, the loss decreases steadily and remains stable throughout training. Compared with the FedAvg result under Gradient Boosting, where the loss increased strongly, Multi-Krum prevents the boosted malicious updates from dominating the global model.

Overall, Multi-Krum is effective against large and clearly abnormal updates, such as Gradient Boosting. However, it is less effective against stealthier attacks such as ALIE, where the malicious updates are crafted to stay close to normal client updates.

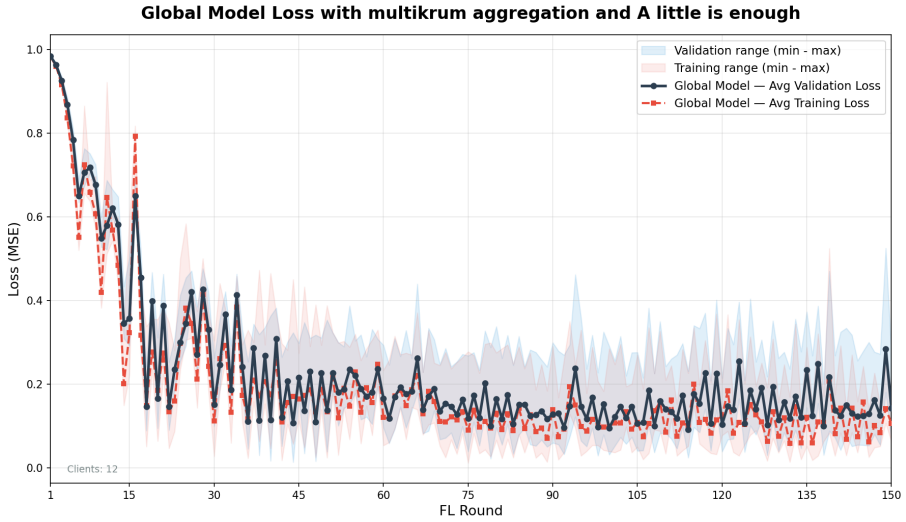


Figure 4.10: A Little Is Enough attack with 8 normal clients and 4 poisoned clients over 150 rounds using Multi-Krum aggregation in the FEDn hierarchy.

4.6.3 FoolsGold

The FoolsGold mitigation was evaluated against both Gradient Boosting and A Little Is Enough in the FEDn hierarchy. In both experiments, the attack was delayed until round 50. This was done because FoolsGold relies on historical client-update behavior to compare clients over time. By allowing the clients to train normally during the first 50 rounds, the aggregator had enough information about the clients before the poisoned updates were introduced.
Figure 4.12 shows the Gradient Boosting experiment with 10 normal clients and 2 poisoned clients. Before round 50, the loss remains low and relatively stable. After the attack starts, the loss begins to increase and continues to rise during the remaining training rounds. This indicates that FoolsGold was not able to reduce the effect of the boosted poisoned updates in this experiment.

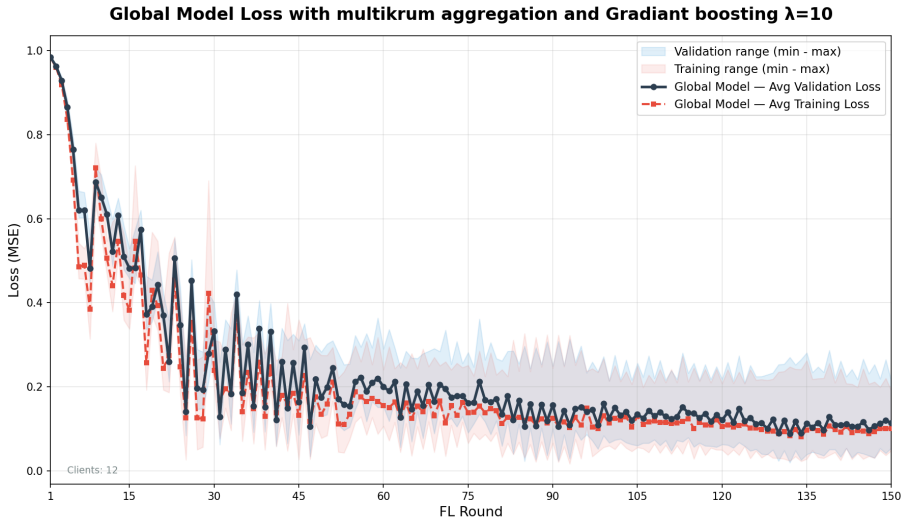


Figure 4.11: Gradient Boosting attack with 10 normal clients and 2 poisoned clients over 150 rounds using Multi-Krum aggregation in the FEDn hierarchy.

Figure 4.13 shows the A Little Is Enough experiment with 8 normal clients and 4 poisoned clients. In this case, the loss decreases during the early rounds before the attack begins. After round 50, the loss remains relatively stable, with only smaller oscillations. This shows that FoolsGold worked well against A Little Is Enough in this setup, since the global model continued training without a strong increase in loss after the poisoned clients became active.

The FoolsGold results show different behavior depending on the attack type. The method was less effective against Gradient Boosting, where the malicious updates were strongly amplified. However, it performed well against A Little Is Enough, where several poisoned clients behaved in a coordinated way over time.

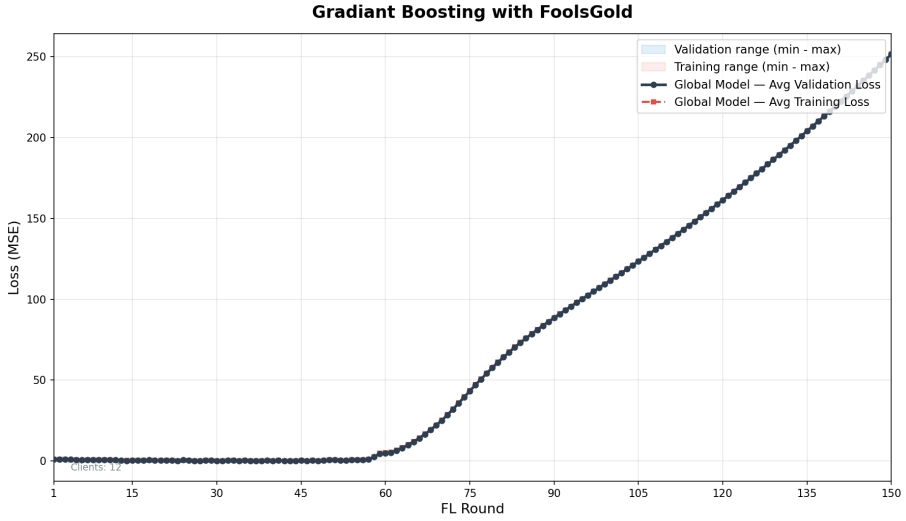


Figure 4.12: Gradient Boosting attack starting at round 50 with 10 normal clients and 2 poisoned clients over 150 rounds using FoolsGold aggregation in the FEDn hierarchy.

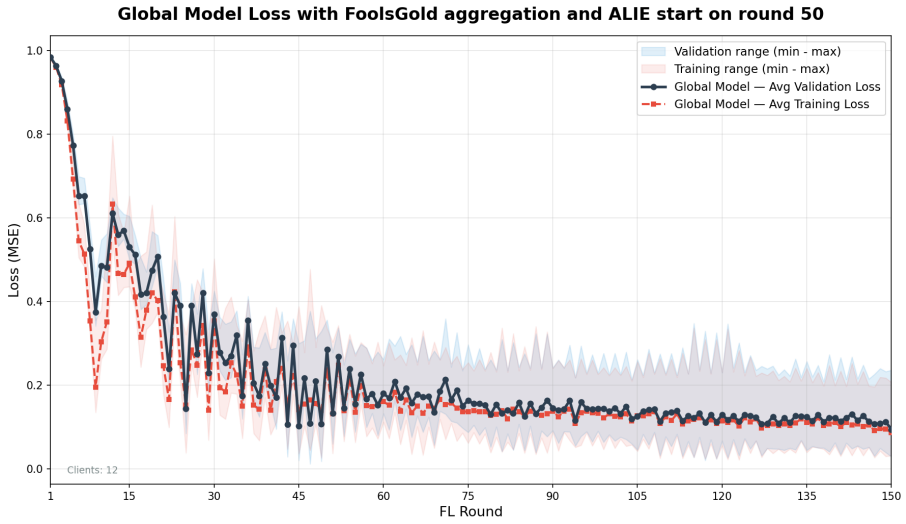


Figure 4.13: A Little Is Enough attack starting at round 50 with 8 normal clients and 4 poisoned clients over 150 rounds using FoolsGold aggregation in the FEDn hierarchy.

4.6.4 Layered Aggregation

The layered aggregation strategy was evaluated against Gradient Boosting, A Little Is Enough, and the Backdoor attack in the FEDn hierarchy. The experiments were used to observe whether the training process remained stable when poisoned clients were present and the aggregation was performed using the layered mitigation approach.

Figure 4.14 shows the Gradient Boosting experiment with 10 normal clients and 2 poisoned clients. In this experiment, the attack was active from round 1. The loss decreases during the training process and does not show the strong divergence observed in the unprotected FedAvg experiment. Although some oscillations are visible during the early rounds, the overall trend remains stable over the 150 rounds. This indicates that layered aggregation reduced the influence of the boosted malicious updates and allowed the global model to continue training.

Figure 4.15 shows the A Little Is Enough experiment with 8 normal clients and 4 poisoned clients. In this case, the attack was introduced at round 50. Before the attack starts, the loss decreases as expected during the early training rounds. After round 50, the loss remains relatively stable, with only smaller oscillations. This shows that the layered aggregation strategy was able to keep the training process controlled even after the poisoned clients became active.

Figure 4.16 shows the Backdoor attack experiment with layered aggregation. In this experiment, the attack was active from round 1. The loss remains stable throughout the training process, and no clear instability appears during the experiment. This suggests that the layered aggregation strategy preserved stable training behavior during the backdoor experiment.

Taken together, the results show that layered aggregation kept the loss stable across the tested attack scenarios. The strongest improvement is seen in the Gradient Boosting experiment, where the large loss increase observed with FedAvg was

avoided. For A Little Is Enough and the Backdoor attack, the model also continued training without clear divergence after the attacks were introduced.

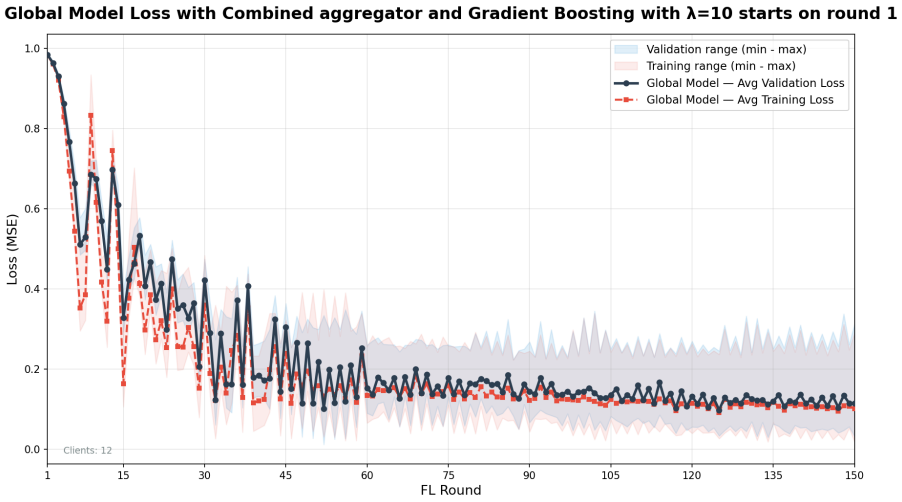


Figure 4.14: Gradient Boosting attack starting at round 1 with 10 normal clients and 2 poisoned clients over 150 rounds using the Layered Aggregation in the FEDn hierarchy.

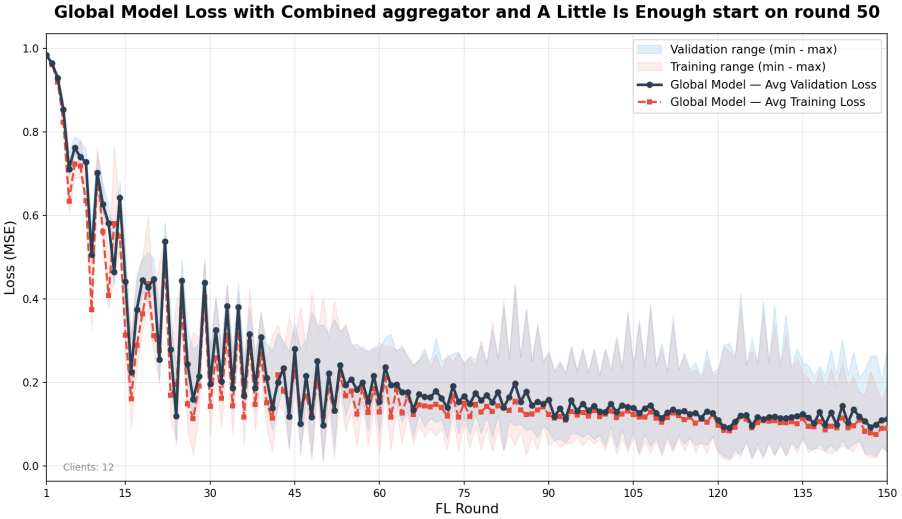


Figure 4.15: A Little Is Enough attack starting at round 50 with 8 normal clients and 4 poisoned clients over 150 rounds using the Layered Aggregation in the FEDn hierarchy.

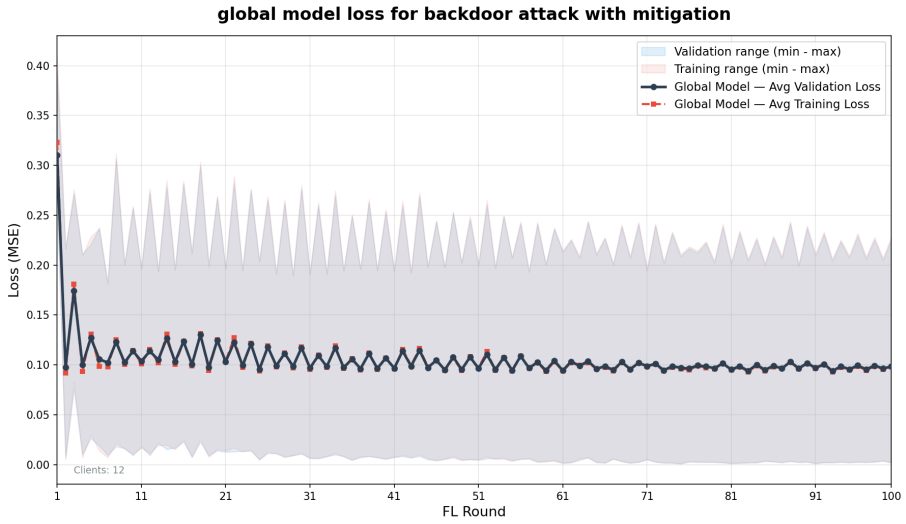


Figure 4.16: Backdoor attack starting at round 100 using the Layered Aggregation in the FEDn hierarchy.

Table 4.2 shows that the backdoor attack was highly effective in the undefended FEDn setup. Both ASR and comparative ASR reached 100%, the trigger shift was large, and the attack survived aggregation. This means that the global model remained accurate on clean inputs while still shifting strongly toward the attacker-defined target when the trigger was present. After layered aggregation was applied, both ASR measures dropped to 0%, the trigger shift became very small, and the backdoor no longer survived aggregation. This indicates that the mitigation successfully prevented the malicious update from dominating the global model behavior. At the same time, the clean prediction error increased compared with the attacked run, which suggests a trade-off between attack resistance and clean-task utility.

Table 4.2: Backdoor impact on the global SOH prediction model in FEDn.

Experiment	Clean MSE	Clean MAE	ASR	Comp. ASR	Trigger Shift MAE	Attack Survived
FEDn + Backdoor	0.0097	0.0796	100.0%	100.0%	1.4960	Yes
FEDn + Layered Aggregation	0.0792	0.2100	0.0%	0.0%	0.0590	No

4.7 Mitigation Performance in AICFL

In the AICFL experiments, no additional robust aggregation method was applied. Instead, the AICFL algorithm itself was evaluated as a possible structural mitigation mechanism. This means that the experiments tested whether the clustering behavior, client selection, and cluster removal process could reduce the influence of poisoned clients without using a separate defense such as Trimmed Mean, Multi-Krum, FoolsGold, or the combined aggregator.
The results show that AICFL can provide some resistance against attacks that create clearly poor validation behavior. This was most visible in the Gradient Boosting experiment, where the malicious clients caused a large loss increase early in training. After this, AICFL created a better-performing cluster and removed the poorly

performing one, which reduced the influence of the poisoned clients in later rounds. In this case, the algorithm acted as a form of isolation, since harmful behavior was pushed away from the clients that achieved lower validation loss.
However, the same effect was weaker against the A Little Is Enough attack. Since ALIE is designed to remain close to the normal update range, it did not create the same clear loss spike as Gradient Boosting. Instead, the attack mainly affected the stability of the cluster behavior, with clients moving between clusters and new clusters being created or removed. This shows that AICFL can help when malicious clients are clearly harmful, but it is less reliable against stealthier attacks that do not appear as obvious outliers.
Overall, AICFL should not be considered a complete mitigation method on its own. The clustering mechanism can sometimes limit the effect of poisoned clients by separating poorly performing updates, but it does not explicitly detect or remove malicious clients. Therefore, AICFL can provide useful structural robustness, especially against visible attacks, but it may need to be combined with robust aggregation methods to handle stealthier poisoning strategies more reliably. For the backdoor experiment, the available AICFL results only show changes in cluster behavior. Since trigger-specific metrics were not measured for the resulting cluster models, the experiment does not establish whether AICFL contained the backdoor itself.

5
Discussion

5.1 Overview of the Main Findings

The experiments show that both the standard FEDn setup and the AICFL setup were able to train normally when no attack was present. In FEDn, the global loss decreased over the training rounds, while in AICFL the validation loss decreased together with changes in the client grouping. This indicates that both setups could learn from the distributed battery data under normal conditions.
The attack results show that malicious clients can affect the training process in different ways. Gradient Boosting caused a clear disturbance because the poisoned updates were strongly amplified. A Little Is Enough was harder to observe from the loss curves, since the poisoned updates stayed closer to normal client behavior. The backdoor attack also showed that stable clean loss values do not always mean that the model is secure, because hidden malicious behavior may still be learned.
The comparison between FEDn and AICFL shows that architecture matters. In FEDn, poisoned updates mainly affect the shared global model. In AICFL, the effect can also appear through client movement, cluster changes, and training stability. Therefore, security evaluation in clustered federated learning should not rely only on final loss values.
The mitigation results show that robust aggregation improved stability in several experiments, but no single method worked equally well against all attacks. Some methods were more effective against clearly abnormal updates, while others were better suited for coordinated malicious behavior. Overall, the findings suggest that federated battery-management systems should be evaluated using model performance, training stability, client behavior, and attack-specific metrics together.

5.2 Security Considerations in Hierarchical Federated Learning

The attack experiments show that the system architecture affects how malicious behavior appears during training. In FEDn, the impact is mainly reflected in the shared global model, since all selected client updates contribute to the same aggregation process. This makes changes in the loss curve easier to relate to the attack.
In AICFL, the effect is broader because the training process also depends on client assignment and cluster adaptation. An attack can therefore influence the system without always causing a clear failure in the final loss. Changes in cluster stability,

client movement, and validation behavior become important signs of how the system reacts under malicious participation.
This means that comparing FEDn and AICFL requires more than checking whether the model converges. The results should be interpreted together with the behavior of the training structure, since an attack may affect the model directly, the clustering process, or both.

5.3 Impact of Attacks in FEDn and AICFL

The experiments show that poisoning attacks affect federated learning differently depending on the attack type and training structure. In the standard FL baseline, poisoned clients mainly influenced the shared aggregation process and the global loss curve, especially when the malicious updates were strongly amplified.
In the adaptive clustered setup, the effect was not limited to model loss. The attacks could also disturb training stability and client behavior during the experiment. This shows that hierarchical or clustered FL should not only be evaluated through final model performance.
A clear difference was seen between disruptive and stealthier attacks. Stronger attacks created visible changes in the loss curve, while stealthier attacks were harder to detect from loss values alone. Therefore, attack evaluation should also include training stability and client-level behavior, together with robust aggregation and post-training analysis as complementary protections.

5.3.1 Gradient Boosting

The Gradient Boosting attack had one of the clearest effects in the experiments. In the standard FL baseline, the loss increased strongly after the poisoned updates started to influence the aggregation. This made the disturbance easy to observe because the attack created a visible change in the loss curve rather than only small oscillations.
This result shows that strongly amplified malicious updates can quickly disturb a standard aggregation process when no robust filtering is applied. The effect was not hidden inside normal training variation. Instead, the sharp increase in loss showed that the training process was being pushed away from the behavior learned by the normal clients.
In the adaptive clustered setup, the attack also caused a strong disturbance early in the experiment. However, the latter behavior was different from the standard baseline. The clustered structure helped to reduce the long-term influence of clearly malicious updates by separating unstable behavior from more stable training behavior. This suggests that adaptive clustering can provide some practical robustness against attacks that produce clearly poor validation behavior.
At the same time, this should not be interpreted as a complete defense. The early disturbance shows that strongly amplified poisoned updates can still affect the training process before their influence is reduced. For this reason, attacks such as Gradient Boosting should still be handled with additional protections, such as robust aggregation.

5.3.2 A Little Is Enough

The A Little Is Enough attack had a subtle effect in the experiments. In the standard FL baseline, the model continued to train, and the loss did not collapse in a clear way. Instead, the main effect was visible as smaller spikes and oscillations in the loss curve. This means that the attack was not immediately obvious from the results, especially compared with attacks that create a large loss increase.
A similar behavior was observed in the adaptive clustered setup. The attack did not create a strong and direct loss increase, but it still introduced small variations in the training behavior. This makes the effect harder to interpret because the clustered setup already contains natural changes caused by client grouping and heterogeneous battery data. Therefore, the attack may not appear as a clear failure, but rather as reduced stability during some parts of the training process.
This behavior is important because it shows that a poisoning attack does not always need to cause strong divergence to affect the training process. The loss can still move in the expected direction while showing signs of instability during some rounds. In a practical setting, this kind of behavior may be difficult to separate from normal variation caused by heterogeneous client data.
For this reason, A Little Is Enough should not be evaluated only by looking at whether the model converges or not. The spikes in the loss curve and the small changes in training behavior show that attacks designed to remain close to normal client behavior can still influence the learning process without causing obvious divergence. This highlights the importance of considering both convergence and training stability when assessing the impact of subtle poisoning attacks.

5.3.3 Backdoor Attack

The backdoor attack showed that normal-looking clean loss is not enough to conclude that the model is secure. In the FEDn experiment, the global model still converged during training, but the backdoor metrics showed that the trigger successfully changed the model behavior. This means that convergence and clean-task performance alone can hide a serious targeted attack.
This is important because the backdoor did not mainly aim to break the overall training process. Instead, it preserved the model's behavior on clean inputs while introducing attacker-controlled behavior under triggered conditions. In a battery-management setting, this is particularly concerning because the model may appear reliable during normal validation while still producing unsafe or misleading SOH predictions when the trigger is present. The result therefore shows that backdoor evaluation must include attack-specific metrics and not rely only on standard training and validation loss. In the AICFL experiment, only the cluster-level behavior was measured for the backdoor case. Therefore, the observed cluster restructuring should not be interpreted as evidence that the backdoor itself was removed or contained.
The practical impact of such an attack depends on the direction of the SOH prediction error. In the implemented backdoor, the malicious target shifts the predicted SOH downward. This may cause a healthy battery to be treated as degraded, which can lead to unnecessary maintenance, reduced availability, or premature replace-

ment. The opposite case, where SOH is overestimated, may be more safety-critical because a genuinely degraded battery could continue to be treated as healthy. The experiments in this thesis focus on the former case, which is why the False Unhealthy Rate is more relevant to the implemented attack than the False Healthy Rate.

5.4 Effectiveness of Mitigation Methods

The mitigation experiments show that robust aggregation improves the system's resilience, but the effect depends on the type of attack. Large and clearly abnormal updates, such as Gradient Boosting, are easier to reduce because they differ strongly from normal client updates. In contrast, stealthier attacks such as A Little Is Enough are more difficult to mitigate, since the poisoned updates are designed to stay close to the normal update range. This explains why some defenses perform well against one attack but not against another.
The results show that no single mitigation method was sufficient for all attack scenarios. Each method improved stability under some conditions, but its effectiveness depended on the behavior of the attack and the training setup. The layered aggregation strategy gave stable results in the tested standard FL experiments, but its internal design is kept at a conceptual level in this thesis. Therefore, it should be seen as a practical mitigation variant rather than a complete guarantee against poisoning attacks.

5.4.1 Trimmed Mean

Trimmed Mean worked well against the Gradient Boosting attack because the poisoned update appeared as a clear extreme value. By removing the highest and lowest values before averaging, the method reduced the influence of the amplified malicious update and kept the training more stable.
However, the result also shows an important limitation. The trimming process is static, since it always removes a fixed number of high and low values in each round. This means that the method does not adapt to how many poisoned clients are present or to the direction of their updates. For example, if the aggregator removes the two highest and two lowest updates, but three malicious clients all push their updates in the same direction, one poisoned update may still remain in the aggregation.
Therefore, Trimmed Mean is useful when malicious updates are clearly abnormal, but it is less reliable when the attack does not match the fixed trimming setting. Choosing the trimming level is also a trade-off: if it is too low, poisoned updates may remain, but if it is too high, useful updates from normal clients may also be removed.

5.4.2 Multi-Krum

The results show that Multi-Krum can be useful when malicious updates differ clearly from the majority of normal client updates. However, its effectiveness depends not only on the distance-based selection itself, but also on how the number of malicious clients is handled.

In the manual version, the method requires an assumed number of malicious clients before the aggregation is performed. This makes the defense sensitive to the quality of that assumption. If the assumed value is close to the real attack setting, Multi-Krum can reduce the influence of poisoned updates more effectively. However, if the assumption is wrong, the method may become less reliable. Underestimating the number of malicious clients can allow poisoned updates to remain in the aggregation, while overestimating it can remove useful honest updates and weaken the final model.
This limitation was especially important when comparing the behavior of Multi-Krum against Gradient Boosting and A Little Is Enough. Against Gradient Boosting, the poisoned updates were more visible because the attack created a stronger deviation from normal client behavior. In this case, Multi-Krum had better conditions for separating harmful updates from the rest. Against A Little Is Enough, the situation was more difficult, since the attack is designed to stay closer to the normal update distribution. This makes it harder for a distance-based method to clearly distinguish between honest and malicious updates.
The automatic version addresses part of this limitation by avoiding a fixed assumption about the number of malicious clients. Instead, it adjusts the selection dynamically based on the behavior of the received updates. This makes it more suitable for realistic settings where the number of malicious clients is not known in advance or may vary between experiments. In this sense, the automatic version provides a more flexible and practical use of Multi-Krum.
At the same time, the automatic version should not be seen as a complete solution. Its performance still depends on whether malicious updates are sufficiently different from honest ones. If poisoned updates are subtle and remain close to the normal update distribution, they may still be difficult to separate. This is especially relevant in heterogeneous federated learning, where honest clients can naturally produce different updates because their data distributions are not identical.
Overall, the manual version of Multi-Krum is useful when the attack setting is known, but it becomes weaker when the assumed number of malicious clients is incorrect. The automatic version reduces this dependency and makes the method more adaptive, but it still relies on the presence of detectable differences between honest and malicious updates. Therefore, Multi-Krum was more suitable for stronger and more visible attacks such as Gradient Boosting, while it was less reliable against stealthier attacks such as A Little Is Enough.

5.4.3 FoolsGold

The FoolsGold results show that the method is more suitable for coordinated and stealthy malicious behavior than for strongly amplified model-poisoning updates. This was visible when comparing its behavior against Gradient Boosting and A Little Is Enough. In the Gradient Boosting experiment, the loss increased after the attack started, which indicates that FoolsGold was not able to sufficiently reduce the influence of the boosted poisoned updates. This is because the main effect of Gradient Boosting comes from the large magnitude of the malicious update, rather than from repeated similarity between malicious clients over time.
The result was better for A Little Is Enough. In that experiment, the loss remained

more stable after the poisoned clients became active. Since ALIE uses several malicious clients that follow a coordinated and less obvious update direction, FoolsGold had better conditions to reduce their influence. Delaying the attack until round 50 also helped the method, since it allowed FoolsGold to build historical information about the clients before the malicious behavior started.
This shows both the strength and the limitation of FoolsGold. It can be useful when malicious clients behave similarly over time, especially in coordinated stealthy attacks such as A Little Is Enough. However, it is less effective when the attack is mainly based on producing a strong abnormal update, as in Gradient Boosting. For this reason, FoolsGold should be seen as a useful defense against coordinated poisoning behavior, but not as a complete protection against all attack types.

5.4.4 Layered Aggregator

The layered aggregator showed the most stable behavior among the mitigation methods evaluated in this thesis. Its main advantage is that it does not rely on a single robustness criterion. Instead, the aggregation process combines multiple robustness principles in a staged manner, allowing different types of suspicious update behavior to be handled within the same aggregation pipeline.
For the Gradient Boosting attack, the layered aggregator maintained stable training even though the attack was active from the first communication round. This is notable because the same attack caused strong instability and divergence in the undefended FEDn setup. Under layered aggregation, the boosted malicious updates did not dominate the aggregation process, and the global model continued to learn throughout the training rounds.
The layered aggregator also showed stable behavior against the A Little Is Enough attack, which became active at round 50. ALIE is more difficult to mitigate because the malicious updates are designed to remain within a statistically plausible range and therefore do not necessarily appear as clear outliers. Despite this, the training loss remained controlled after the malicious clients became active. In the tested scenario, this indicates that combining multiple robustness mechanisms was more effective at limiting the influence of coordinated poisoned updates than relying on a single mitigation principle.
For the Backdoor attack, which was active from the first round, the layered aggregator also reduced the observed attack effect. The clean training behavior remained stable, while the trigger-based evaluation showed that the malicious backdoor behavior was strongly reduced. In particular, the reported attack-success measures decreased from 100% in the undefended Backdoor experiment to 0% with layered aggregation, while the trigger-induced prediction shift was also substantially reduced.
However, this increased robustness was accompanied by a reduction in clean-task utility. The clean MSE and MAE were higher with layered aggregation than in the undefended attacked model. This illustrates that stronger robustness does not necessarily come without a cost, and that filtering or down-weighting suspicious updates may also reduce the contribution of useful client information.
Overall, the layered aggregator provided the most stable behavior across the tested

Gradient Boosting, ALIE, and Backdoor scenarios. These results should, however, be interpreted as an empirical evaluation of the proprietary layered aggregation strategy under the experimental conditions considered in this thesis. They do not establish that the method is universally robust or fully reproducible from the description provided in this report. Its effectiveness may depend on factors such as the model architecture, client population, data heterogeneity, attack configuration, and internal aggregation parameters. The layered aggregator should therefore be viewed as a practical defense-in-depth strategy rather than a complete security guarantee.

5.5 FEDn Compared with AICFL

The comparison between FEDn and AICFL shows that the architecture has a clear effect on how poisoned clients influence the training process. In FEDn, all selected client updates contribute to one shared global model. This makes the system easier to analyse, but it also means that a harmful update can affect the whole model if it is included in the aggregation. This was especially visible when the attack produced strongly amplified updates.

AICFL behaved differently because the training process is divided across cluster models. This means that malicious behavior does not always affect the full federation in the same direct way. In some cases, clearly harmful updates were partly isolated through the cluster structure, since poorly performing clusters could be removed or avoided during training. This shows that the clustered design can provide a form of structural robustness, especially when the malicious behavior creates visible performance degradation.

At the same time, AICFL should not be seen as automatically more secure than FEDn. The adaptive clustering process adds extra behavior that can itself be affected by attacks. Client assignment, cluster creation, and cluster removal can all be influenced when malicious clients disturb the training process. This was particularly relevant for stealthier attacks, where the effect was less visible in the loss curve but still influenced the stability of the clustering behavior.

Overall, FEDn provides a simpler and more direct training structure, while AICFL provides more flexibility and can sometimes limit the spread of clearly harmful updates. However, the additional cluster logic also makes the security analysis more complex. For this reason, comparing the two systems should not only focus on final loss values, but also on how stable the training process and client grouping remain under attack.

5.6 Suspicious-Client Risk Scoring

The suspicious-client risk scoring mechanism was useful as a post-training analysis tool because it gave a client-level view of the training behavior. The attack and mitigation results mainly show how the global model, cluster models, or loss curves behave under attack. However, they do not always show which clients contributed most to the suspicious behavior. The risk-scoring mechanism helped fill this gap by ranking clients according to update abnormality, peer similarity, validation behavior,

and response-time signals.

The results in Table 4.1 show that the ALIE clients received higher risk scores than the normal clients. This is important because ALIE is designed to be less visible than a large gradient-boosting attack. Instead of sending very large updates, the malicious clients try to stay close to the normal update distribution. For that reason, the attack may not always appear as a clear loss divergence. The risk-scoring output shows that client-level signals can still reveal suspicious behavior, especially when several clients produce similar updates over multiple rounds.

A key strength of the mechanism is that it does not depend on only one metric. A large update alone may indicate an attack, but it may also be caused by normal heterogeneity in the battery data. Similarly, a high response time alone does not prove malicious behavior, since it can also be caused by system load or network delay. The risk score becomes more useful when several weak signals point in the same direction. In the ALIE example, the flagged clients had higher update ratios, higher peer similarity, higher response times, and higher attack probabilities. This combination made the suspicious ranking more convincing than any single metric by itself.

At the same time, the mechanism should not be interpreted as a complete detector that proves which clients are malicious. The output is better understood as a suspicious-client ranking. This distinction is important because federated battery data is naturally heterogeneous. Honest clients may produce different updates because they represent different battery types, degradation patterns, or operating conditions. Therefore, unusual behavior does not automatically mean malicious behavior. The threshold τ_{risk} is also a heuristic parameter. In the current implementation it is set to 4, but this value may need to be adjusted for other datasets, model architectures, federation sizes, or deployment conditions.

The risk-scoring mechanism also has a different role from the mitigation methods. Trimmed Mean, Multi-Krum, FoolsGold, and the layered aggregator directly affect the aggregation process. In contrast, the risk-scoring mechanism does not change the model update during training. It is mainly used after training to support analysis, debugging, and forensic inspection. This makes it useful as a complementary tool rather than as a replacement for robust aggregation.

Another limitation is that the method depends on access to individual client updates or update fingerprints. This is acceptable in the controlled experimental setup used in this thesis, but it may not always be available in privacy-preserving deployments. If secure aggregation hides individual client updates from the server, then client-level risk scoring becomes much harder or impossible. This creates a trade-off between security monitoring and privacy protection.

Overall, suspicious-client risk scoring provided useful evidence about which clients behaved unusually during the experiments. It was especially helpful for attacks where the loss curve alone did not clearly reveal the malicious behavior. However, the result should be treated as a ranking and triage signal, not as a final proof of malicious participation. In practical federated battery-management systems, this kind of analysis would be most useful when combined with robust aggregation, validation monitoring, and manual inspection of high-risk clients.

5.7 Limitations and Threats to Validity

The results in this thesis should be interpreted within the limits of the experimental setup. The experiments were performed in a controlled containerized environment with a fixed number of clients, selected attack configurations, and specific mitigation methods. This made it possible to compare FEDn and AICFL under repeatable conditions, but it does not fully represent all conditions that may occur in a real battery-management deployment.

One limitation is the federation size. The experiments mainly used a limited number of clients 12, which was useful for controlled comparison, but larger federations with more clients, more clusters, and more diverse data may behave differently as the system gets larger and more complex. The results therefore show how the tested systems behaved in this setup, rather than providing a general guarantee for all federated learning systems.

Another limitation is the attack scope. The thesis focused mainly on client-side poisoning attacks, including Gradient Boosting, A Little Is Enough, and Backdoor attacks. Other threats, such as network attacks, server compromise, privacy leakage, denial-of-service behavior, or fully adaptive attackers, were outside the scope of this work. In addition, attacks directly targeting the cluster level were not evaluated. For example, the experiments did not study malicious manipulation of cluster creation, cluster deletion, client assignment, or compromised cluster-level aggregation components.

The dataset and model architecture also affect the validity of the results. Battery data is naturally heterogeneous, meaning that honest clients may produce different updates even without malicious behavior. This makes it difficult to clearly separate malicious behavior from normal client variation. The results may therefore depend on the specific battery dataset, model architecture, feature representation, and client partitioning used in the experiments.

The mitigation results are also limited by the assumptions and parameters of each method. Some defenses depend on assumptions about the number of malicious clients, the distance between benign and malicious updates, or the similarity between coordinated attackers. These assumptions may not hold in all deployments, especially when attackers adapt their behavior to avoid detection or when honest clients are highly non-IID.

A further limitation concerns the layered aggregation strategy. Parts of its implementation are proprietary and are therefore not fully disclosed in this report. In particular, implementation-specific thresholds, parameter settings, and parts of the internal decision logic are omitted. Consequently, the layered aggregation experiments should not be interpreted as providing a fully reproducible implementation of a new aggregation algorithm. Instead, they represent an empirical evaluation of a proprietary defense strategy under the attack scenarios and experimental conditions

considered in this thesis. The evaluation therefore focuses on the conceptual aggregation stages, the types of signals involved, and the observable effects on model behavior and attack mitigation.

Finally, the suspicious-client risk scoring mechanism should be understood as a support tool for analysis, not as proof that a client is malicious. High-risk behavior may indicate an attack, but it may also be caused by unusual yet benign battery data, unstable local training, or differences in operating conditions. For this reason, the risk score is best interpreted as a ranking for further inspection rather than an automatic decision mechanism.

Overall, the results provide useful experimental evidence for the tested FEDn and AICFL settings. However, they should not be interpreted as a complete security guarantee for all federated battery-management systems.

5.8 Practical Implications for Battery-Management FL Systems

The results show that federated learning for battery-management systems should not rely only on standard aggregation when malicious clients may be present. Since battery models can support safety-relevant tasks such as State-of-Health estimation, poisoned updates may lead to incorrect model behavior even when the raw training data remains private. This means that robustness needs to be considered as part of the system design, not only as an additional experiment after training.
For practical deployments, the choice of mitigation should depend on the expected attack behavior. Methods that filter abnormal updates can help when poisoned clients create large deviations, while similarity-based methods are more useful when several malicious clients behave in a coordinated way over time. The layered aggregator gave the most stable behavior in the tested experiments, which suggests that combining robustness mechanisms can be more practical than relying on one single defense.
The results also show that clustered federated learning can provide useful system-level information. In AICFL, attacks may affect not only the model loss but also client movement and cluster stability. Therefore, monitoring should include both model-level metrics and system-level behavior. This is especially important in battery-management settings, where honest clients may naturally differ because of different battery types, usage patterns, and operating conditions.
A practical implication is that future battery-management FL systems should combine robust aggregation with client-behavior analysis. Loss values alone are not always enough, especially for stealthy or backdoor-style attacks. By observing update behavior, training stability, and cluster changes, the system can provide stronger evidence about whether the learning process is behaving normally or being influenced by malicious clients.

6

Social and Ethical Aspects

This thesis examines security challenges in clustered federated learning for battery-management intelligence. In such systems, learned models may support monitoring, prediction, and decision-making related to battery health and system reliability. Because battery-management systems can be used in vehicles, drones, and distributed energy assets, failures or malicious manipulation may affect more than technical performance. They may also influence user trust, service reliability, operational safety, and the confidence placed in AI-based battery systems.

For this reason, security evaluation is not treated as a separate technical concern, but as an important part of responsible system development. Studying attacks, mitigations, and system behavior under malicious conditions helps identify weaknesses before such methods are used in practical environments. This is especially important in safety-relevant applications, where incorrect model behavior may lead to misleading battery-health estimates or unreliable system decisions.

6.1 Social Aspects

From a social perspective, improving resilience against poisoning and backdoor attacks is important because battery-management models may influence decisions related to battery health, reliability, and safe operation. If such models are manipulated, the result could be incorrect State-of-Health predictions, unnecessary downtime, unsafe system behavior, or reduced trust in AI-enabled mobility and energy services.

This makes security and robustness relevant not only as technical goals, but also as social responsibilities. As AI systems are increasingly used in safety-relevant contexts, there is a growing need for systematic risk management, documentation, and evaluation of security measures. The work in this thesis therefore aligns with broader AI risk-management practices and regulatory expectations for trustworthy AI systems [11, 12].

6.2 Ethical Aspects

The experiments in this thesis are conducted for defensive research purposes and are limited to controlled environments, such as simulations and isolated testbeds. The work does not target real deployed systems, and no personal data is processed as part of the experiments. This reduces the risk of harm while still allowing the

security behavior of the federated learning system to be studied in a realistic and reproducible way.
Since the thesis involves attack implementation, responsible handling of the results is important. The purpose of implementing attacks is to understand weaknesses and evaluate possible mitigations, not to enable misuse. Therefore, sensitive findings should be presented with enough detail to support scientific understanding and remediation, while avoiding unnecessary information that could increase the risk of harmful use. This approach follows established ethical principles for ICT and security research, including respect for persons, beneficence, justice, and respect for law and public interest [10].

6.3 Use of AI Assistance

The authors conducted all work in this thesis, including the experiments, analysis, interpretation of results, and final writing decisions. AI-assisted tools, including ChatGPT, were used to support language improvement, grammar correction, text structuring, and technical clarification in the report. The purpose of using these tools was to enhance the quality, readability, and clarity of the text. All AI-assisted suggestions were reviewed, edited, and approved by the authors, who remain fully responsible for the final content of the thesis.

7

Conclusion

7.1 Summary of the Work

This thesis investigated cybersecurity challenges in decentralized machine learning for battery-management systems. The work focused on federated learning using the FEDn framework and an adaptive clustered federated learning setup, referred to as AICFL. The main purpose was to study how malicious clients can influence the training process and how different mitigation methods can reduce the impact of such attacks.

A controlled experimental environment was created where normal and poisoned clients could participate in the same federated learning process. Three attack types were implemented and evaluated: Gradient Boosting, A Little Is Enough, and a Backdoor attack. These attacks were chosen because they represent different malicious behaviors, from clearly disruptive model poisoning to more subtle and targeted manipulation.

The experiments showed that poisoning attacks can affect both standard FEDn and AICFL, but the effect depends on the attack type and the learning architecture. In FEDn, malicious updates directly influenced the shared global model. In AICFL, the impact could also appear through changes in cluster behavior, client movement, and training stability. This showed that security in clustered federated learning must be evaluated not only through model loss, but also through the behavior of the training structure.

In the standard FEDn setup, the mitigation methods affected the attacks in different ways. Trimmed Mean and Multi-Krum reduced the impact of clearly abnormal updates such as Gradient Boosting, but they were less reliable against stealthier behavior such as A Little Is Enough, where poisoned updates stayed closer to the normal update range. FoolsGold was more useful against coordinated malicious clients because it used update similarity over time, but it did not fully address strongly amplified updates. The layered aggregator gave the most stable overall behavior in FEDn, since it combined several robustness principles instead of relying on a single aggregation rule.

In the AICFL setup, the mitigation effect came mainly from the clustered learning structure rather than from an added robust aggregation method. The clustering process could reduce the influence of clearly harmful clients by separating or removing poorly performing clusters, which was most visible in the Gradient Boosting experiment. However, this structural protection was weaker against stealthier attacks such as A Little Is Enough and against backdoor behavior, because these attacks

did not always appear as clear loss-based outliers. Therefore, AICFL changed how attacks propagated through the system, but it did not remove the need for robust aggregation and client-level monitoring.

7.2 Answering the Research Questions

The first research question focused on studying the behavior, communication, configuration, and vulnerabilities of the decentralized learning network. This was addressed by implementing and evaluating both the standard FEDn setup and the AICFL setup. The experiments showed that FEDn provides a clear federated learning baseline where client updates are collected and aggregated into one shared global model. AICFL extended this structure by introducing adaptive clustering, where clients can be associated with different cluster models during training. This made it possible to observe how the system behaves under normal conditions and how the architecture changes the way malicious updates can influence the training process.

The second research question focused on implementing attacks and analyzing how the system is affected. This was addressed by implementing Gradient Boosting, A Little Is Enough, and a Backdoor attack. The results showed that the attacks affected the system in different ways. Gradient Boosting caused strong and visible instability because the malicious updates were amplified. A Little Is Enough had a more subtle effect, since the poisoned updates were designed to remain close to normal client behavior. The Backdoor attack showed that malicious behavior can be hidden even when the clean training loss appears stable. These results show that attack impact cannot be evaluated only by checking whether the loss increases or not.

The third research question focused on mitigation methods to reduce the impact of attacks. This was addressed by evaluating several robust aggregation methods, including Trimmed Mean, Multi-Krum, FoolsGold, and the layered aggregator. The results showed that mitigation effectiveness depends on the type of attack. Methods that remove extreme or distant updates were more effective against Gradient Boosting, while FoolsGold was more useful against coordinated and stealthier behavior, such as A Little Is Enough. The layered aggregator showed the most stable behavior across the tested attacks, which indicates that combining several robustness principles can improve resilience compared with relying on a single mitigation method.

Overall, the research questions were answered by showing how FEDn and AICFL behave under normal and adversarial conditions, how different poisoning attacks affect the learning process, and how mitigation methods can reduce the impact of malicious clients. The findings also show that security evaluation in federated battery-management systems should include model performance, training stability, client behavior, and system-level effects.

7.3 Limitations

This thesis was conducted in a controlled experimental environment, which made it possible to compare attacks and mitigation methods under the same conditions. However, this also means that the results should be interpreted within the scope of the implemented setup. The experiments used a limited number of clients, selected attack types, and predefined mitigation methods.
Another limitation is that the evaluation focused mainly on client-side poisoning attacks. Other threats, such as network attacks, server compromise, privacy leakage, or adaptive attackers, were outside the scope of this work. In addition, attacks directly targeting the cluster level were not included. For example, the thesis did not evaluate malicious manipulation of cluster creation, cluster deletion, or compromised cluster-level aggregation components.
The results are also influenced by the battery dataset, the model architecture, and the client partitioning used in the experiments. Since battery data is naturally heterogeneous, honest clients may produce different updates even without malicious behavior. This makes it challenging to clearly separate attacks from normal variation in the data.
Overall, the results provide useful insight into the tested FEDn and AICFL settings, but they should be understood as experimental evidence rather than a general security guarantee for all federated battery-management systems.

7.4 Future Work

Future work could extend the experimental setup to larger federations with more clients, more clusters, and more varied battery datasets. This would make it possible to study whether the observed attack and mitigation behavior remains stable when the system becomes closer to a real deployment. It would also be useful to test the same attacks with different model architectures and battery-management tasks, such as State-of-Charge prediction or remaining useful life estimation.
Another important direction is to study more advanced attackers. In this thesis, the attacks followed predefined strategies and mainly targeted the client update before aggregation. Future work could evaluate adaptive attackers that change their behavior depending on the mitigation method being used. This would give a better understanding of how robust the system is when the attacker actively tries to avoid detection or bypass the aggregation defense.
The AICFL setup also opens the possibility for attacks that are designed specifically for clustered federated learning. Instead of only poisoning the model update, an attacker could try to influence the clustering behavior itself. For example, malicious clients could attempt to affect client assignment, force unnecessary cluster creation, keep weak clusters active, or push honest clients toward less suitable cluster models. Studying these AICFL-specific attacks would make it possible to evaluate whether the adaptive cluster mechanism can be protected against attacks that target the structure of the system, not only the learned model.
Future work could also investigate cluster-level defenses. This may include safer

client reassignment, cluster-level anomaly detection, stronger validation before creating or deleting clusters, and robust aggregation inside each cluster. These mechanisms could help reduce the risk that malicious clients influence both the model training and the cluster lifecycle.
Finally, future work could combine robust aggregation with stronger client-behavior analysis. Loss values alone are not always enough to detect stealthy or backdoor-style attacks. A monitoring approach that observes update behavior, client movement, cluster stability, and suspicious-client scores could provide better support for detecting malicious clients in federated battery-management systems.

Bibliography

[1] Zhong Ren and Changqing Du. A review of machine learning state-of-charge and state-of-health estimation algorithms for lithium-ion batteries. *Energy Reports*, 9:2993–3021, 2023. Accessed: 2026-02-10.

[2] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 2017. Accessed: 2026-02-10.

[3] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2938–2948. PMLR, 2020. Accessed: 2026-02-10.

[4] Bhupathi Hariprasad and Vijayalaxmi Biradar. AI and ML applications in battery management: A technical overview. *Globus An International Journal of Medical Science, Engineering and Technology*, 13(2):67–70, 2024. Accessed: 2026-02-09.

[5] Morgan Ekmefjord, Addi Ait-Mlouk, Sadi Alawadi, Mattias Åkesson, Prashant Singh, Ola Spjuth, Salman Toor, and Andreas Hellander. Scalable federated machine learning with FEDn. *arXiv preprint arXiv:2103.00148*, 2021. Accessed: 2026-02-09.

[6] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 19586–19597. Curran Associates, Inc., 2020. Accessed: 2026-02-10.

[7] Niclas Svensson. Implementing adaptive clustered federated learning in FEDn: A proof-of-concept for battery state-of-health estimation. Master's thesis, Uppsala University, 2025.

[8] Yunhao Feng, Yanming Guo, Yinjian Hou, Yulun Wu, Mingrui Lao, Tianyuan Yu, and Gang Liu. A survey of security threats in federated learning. *Complex & Intelligent Systems*, 11(165), 2025.

[9] Scaleout Systems AB. scaleoutsystems/fedn: An enterprise-ready and vendor-agnostic federated learning framework. GitHub repository, 2026. Accessed: 2026-02-09.

[10] Erin Kenneally and David Dittrich. The menlo report: Ethical principles guiding information and communication technology research. Technical report, U.S. Department of Homeland Security, 2012. Accessed: 2026-02-09.

[11] Elham Tabassi. Artificial intelligence risk management framework (AI RMF 1.0). Technical report, National Institute of Standards and Technology, 2023. Accessed: 2026-02-09.

[12] European Parliament and the Council of the European Union. Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act), 2024. Accessed: 2026-02-09.

[13] U.S. Department of Energy, Alternative Fuels Data Center. Electric vehicle benefits and considerations, 2026. Accessed: 2026-05-22.

[14] Challa Priyanka, Ganta Sai Keerthi, and G. Indira Kishore. A survey on battery management system in electric vehicles. In *AIP Conference Proceedings*, volume 3298, page 030009, 2025. Accessed: 2026-05-22.

[15] Microchip Technology Inc. Fundamental understanding of a battery management system, 2023. Accessed: 2026-05-22.

[16] Abdelrahman O. Ali, Osama Abdelrehim, Mahmoud M. Saafan, Mohamed R. Elmarghany, and Ahmed M. Hamed. Comprehensive review of battery management systems for electric vehicles: Thermal management, charging strategies, and emerging technologies. *Journal of Power Sources*, 658:238269, 2025. Accessed: 2026-05-22.

[17] Nur Banu Altinpulluk, Deniz Altinpulluk, Paritosh Ramanan, Noah Paulson, Feng Qiu, Susan Babinec, and Murat Yildirim. Federated battery diagnosis and prognosis. *arXiv preprint arXiv:2310.09628*, 2023. Accessed: 2026-02-09.

[18] Gilad Baruch, Moran Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019. Accessed: 2026-03-12.

[19] Dong Yin, Yudong Chen, Kannan Ramchandran, and Peter Bartlett. Byzantine-robust distributed learning: Towards optimal statistical rates. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5650–5659. PMLR, 2018. Accessed: 2026-03-20.

[20] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, volume 30, pages 118–128, 2017. Accessed: 2026-03-20.

[21] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[22] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. The limitations of federated learning in sybil settings. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 301–316. USENIX Association, 2020.

CHALMERS
UNIVERSITY OF TECHNOLOGY