

الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
جامعة عين تموشنت - بلحاج بوشعيب
University of Ain Temouchent - Belhadj Bouchaib
Faculty of Science and Technology
Department of Mathematics and Computer Science



Final Year Project

For the award of the degree of **Master** in: **CYSIA**
Field: Mathematics and Computer Science
Specialty: Cybersecurity and Artificial Intelligence

An Adaptive and Behavior-Aware Aggregation Strategy for IOT-Based Hierarchical Federated Learning Systems

Submitted by:

- 1) Aya BOUDAOUD
- 2) Sara BENAMAR

Dr. Hichem Errahmani BOUCHAKOUR	University of Ain Temouchent	President
Dr. Mansour MEDEDJEL	University of Ain Temouchent	Examiner
Dr. Imad Fethi BENARIBI	University of Ain Temouchent	Supervisor
Dr. Khalid SULTAN	American University of Kuwait College of Engineering and Applied Sciences	Co-supervisor
Dr. Samir OUCHANI	CESI Engineering School, Aix-en-Provence, France	Co-supervisor

Academic Year 2025/2026

Dedication

All praise and thanks are due to Allah (SWT), the Most Merciful and the Most Compassionate, for granting us the knowledge, strength, patience, and opportunity to complete this thesis. Without His guidance and blessings, this accomplishment would not have been possible.

*First and foremost, we dedicate this thesis with profound gratitude and respect to our supervisor **Dr. Imad Fethi BENARIBI**, whose guidance, wisdom, and unwavering support have been invaluable throughout this work. More than a teacher, you have been a mentor, a source of encouragement, and a father figure whose kindness and belief in our abilities inspired us to overcome challenges and strive for excellence. Your influence has left a lasting mark on both our academic and personal growth.*

*We dedicate this work with sincere appreciation to **Dr. Samir OUCHANI** and **Dr. Khalid SULTAN**, for your valuable guidance, insightful feedback, and continuous support throughout this journey. This experience of working under your supervision has been truly enriching and inspiring, and we feel deeply honored to have learned under your mentorship.*

*Our sincere thanks go to the members of the jury **Dr. Hichem Errahmani BOUCHAKOUR** and **DR. Mansour MEDEDJEL**, for honoring us with their presence, dedicating their valuable time to the evaluation of this thesis, and providing insightful remarks and suggestions.*

To Mr. Abbas, Mr. Hamdi.M and Mr. Hariri, with sincere gratitude for your kindness and support. May Allah bless your kind hearts and grant you success in all your endeavors.

To our beloved families, words cannot fully express our gratitude. Your unconditional love, endless sacrifices, prayers, and constant support have been the foundation of everything we have achieved. This accomplishment is as much yours as it is ours.

*We dedicate this work to our soulmates, **H.Gh** and **NOR AMINA .Kh**, whose friendship, loyalty, and companionship have brought light to our lives throughout this journey. Thank you for standing by our side through moments of stress, doubt, and success. Your presence has made every challenge easier and every achievement more meaningful.*

*To **ASMA**, **CHAIMAA** and **AMIRA** we are deeply grateful for your constant support and for standing with us through every step of this journey.*

To all of you, this thesis is dedicated with love, gratitude, and deep appreciation.

Acknowledgment

*We would like to express our sincere gratitude to our supervisors, **Dr Imad Fethi BENARIBI**, **Dr Samir OUCHANI**, and **Dr Sultan KHALID**, for their invaluable guidance, continuous support, and encouragement throughout the completion of this thesis.*

Their expertise, insightful feedback, and dedication have greatly contributed to the development and success of this work.

We are deeply thankful for the time they devoted to reviewing our progress, answering our questions, and providing constructive suggestions that helped improve the quality of this research. Their patience, professionalism, and commitment to academic excellence have been a constant source of motivation.

It has been a privilege to work under their supervision, and we sincerely appreciate their trust, kindness, and unwavering support throughout this journey.

Abstract

Federated Learning improves privacy by keeping data decentralized, making it particularly suitable for distributed Internet of Things (IoT) environments where large numbers of edge devices collaboratively train machine learning models. However, the lack of visibility into clients' local training processes creates vulnerabilities to Byzantine attacks that can compromise the global model. Existing robust aggregation methods still face limitations due to their stateless behavior and reliance on limited behavioral indicators, which makes them less effective against adaptive adversaries. Therefore, a dynamic, multi-dimensional, and history-aware trust mechanism is required to achieve more reliable aggregation.

This thesis proposes TWA-HFL, a dynamic trust-aware aggregation framework for Hierarchical Federated Learning that continuously evaluates client reliability throughout the training process. The proposed approach combines multiple behavioral indicators, including update similarity, peer consistency, magnitude stability, and historical behavior, into an adaptive trust score. A temporal regulation mechanism is introduced to monitor trust evolution over time by rewarding consistent clients and reducing the contribution of potentially malicious participants. Furthermore, trust-enhanced clustering and weighted aggregation are applied to select reliable updates and improve resilience against Byzantine attacks.

Extensive experiments were conducted on MNIST, FashionMNIST, and CIFAR-10 datasets under various attack scenarios and malicious client ratios. The experimental results demonstrate that TWA-HFL consistently outperforms traditional and robust aggregation baselines, achieving higher accuracy and stronger robustness against adaptive Byzantine attacks.

Keywords: Federated Learning, Hierarchical Federated Learning, Internet of Things (IoT), Byzantine Attacks, Robust Aggregation, Trust Score, Adversarial Attacks, Gradient Scaling, Label Flipping, IoT Security.

Résumé

L'apprentissage fédéré améliore la confidentialité en conservant les données de manière décentralisée, ce qui le rend particulièrement adapté aux environnements de l'Internet des objets (IoT), où un grand nombre de dispositifs en périphérie collaborent pour entraîner des modèles d'apprentissage automatique. Cependant, l'absence de visibilité sur les processus d'entraînement locaux des clients expose le système à des attaques byzantines susceptibles de compromettre le modèle global. Les méthodes d'agrégation robustes existantes présentent encore des limites dues à leur caractère statique et à leur dépendance à des indicateurs comportementaux restreints, ce qui réduit leur efficacité face à des adversaires adaptatifs. Il devient donc nécessaire de concevoir un mécanisme de confiance dynamique, multidimensionnel et tenant compte de l'historique afin d'assurer une agrégation plus fiable.

Cette thèse propose TWA-HFL, un cadre d'agrégation dynamique basé sur la confiance pour l'apprentissage fédéré hiérarchique, qui évalue en continu la fiabilité des clients tout au long du processus d'entraînement. L'approche proposée combine plusieurs indicateurs comportementaux, notamment la similarité des mises à jour, la cohérence entre pairs, la stabilité des amplitudes et l'historique des comportements, afin de produire un score de confiance adaptatif. Un mécanisme de régulation temporelle est introduit pour suivre l'évolution de la confiance dans le temps, en récompensant les clients cohérents et en réduisant l'influence des participants potentiellement malveillants. En outre, un regroupement basé sur la confiance et une agrégation pondérée sont utilisés pour sélectionner les mises à jour fiables et améliorer la robustesse face aux attaques byzantines.

Des expériences approfondies ont été menées sur les ensembles de données MNIST, FashionMNIST et CIFAR-10 sous différents scénarios d'attaque et différents taux de clients malveillants. Les résultats expérimentaux montrent que TWA-HFL surpasse systématiquement les méthodes d'agrégation classiques et robustes, en obtenant une meilleure précision et une robustesse accrue face aux attaques byzantines adaptatives.

Mots clés : Apprentissage fédéré, apprentissage fédéré hiérarchique, Internet des objets (IoT), attaques byzantines, agrégation robuste, score de confiance, attaques adversariales, mise à l'échelle des gradients, inversion d'étiquettes, sécurité de l'IoT.

الملخص

يُحسّن التعلّم الفيدرالي الخصوصية من خلال إبقاء البيانات موزعة ولا مركزية، مما يجعله مناسباً بشكل خاص لبيئات إنترنت الأشياء (IoT)، حيث تتعاون أعداد كبيرة من الأجهزة الطرفية في تدريب نماذج التعلّم الآلي. إلا أنّ غياب الشفافية في عمليات التدريب المحلي لدى العملاء يجعل النظام عرضة لهجمات بيزنطية قد تُهدد النموذج العالمي. ولا تزال أساليب التجميع المتينة الحالية تعاني من محدودية بسبب اعتمادها على نهج غير سياقي (stateless) واعتمادها على مؤشرات سلوكية محدودة، مما يقلل من فعاليتها أمام المهاجمين المتكيفين. لذلك تبرز الحاجة إلى آلية ثقة ديناميكية متعددة الأبعاد ومرتبطة بالتاريخ لتحقيق تجميع أكثر موثوقية.

تقترح هذه الأطروحة إطار TWA-HFL، وهو إطار تجميع ديناميكي وإع بالثقة للتعلّم الفيدرالي الهرمي، يقوم بتقييم موثوقية العملاء بشكل مستمر طوال عملية التدريب. يعتمد هذا النهج على دمج عدة مؤشرات سلوكية تشمل تشابه التحديثات، واتساق الأقراص، واستقرار المقدار، والسلوك التاريخي، لإنتاج درجة ثقة تكيفية. كما يتم تقديم آلية تنظيم زمني لمتابعة تطور الثقة عبر الزمن، من خلال تعزيز مساهمة العملاء المتسقين وتقليل تأثير المشاركين المحتملين الضارين. إضافةً إلى ذلك، يتم اعتماد التجميع الموزون والمدعوم بالثقة لاختيار التحديثات الموثوقة وتحسين مقاومة النظام للهجمات البيزنطية.

أُجريت تجارب واسعة على مجموعات بيانات MNIST و FashionMNIST و CIFAR-10 ضمن سيناريوهات هجوم مختلفة ونسب متفاوتة من العملاء الضارين. وتُظهر النتائج أن TWA-HFL يتفوق باستمرار على أساليب التجميع التقليدية والمتينة، محققاً دقة أعلى وقدرة أقوى على مقاومة الهجمات البيزنطية التكيفية.

الكلمات المفتاحية:

التعلّم الفيدرالي، التعلّم الفيدرالي الهرمي، إنترنت الأشياء، هجمات بيزنطية، التجميع المتين، درجة الثقة، الهجمات العدائية، تحجيم التدرجات، عكس التصنيفات، أمن إنترنت الأشياء.

Contents

1	General Introduction	1
2	Background of Federated Learning	7
2.1	Introduction	7
2.2	Fundamentals of Machine Learning	7
2.2.1	ML Definition	7
2.2.2	Learning Paradigms	8
2.2.3	Core Training Concepts	10
2.2.4	Deep Learning Fundamentals	14
2.3	Motivation for Decentralized Learning	18
2.4	Federated Learning	18
2.4.1	Definition and Core Concepts	19
2.4.2	FL Training Workflow	19
2.4.3	FL Taxonomies	20
2.4.4	The FedAvg Algorithm	22
2.5	Hierarchical Federated Learning	22
2.5.1	Definition of HFL	22
2.5.2	Motivation for HFL	23
2.5.3	The Three-Tier Architecture	23
2.5.4	HFL Workflow	24
2.6	Applications of Federated and Hierarchical Federated Learning	24
2.7	Fundamental Challenges in FL and HFL	25
2.7.1	Statistical Heterogeneity	25

2.7.2	Systems Heterogeneity	25
2.7.3	Communication Overhead and Efficiency	26
2.7.4	Vulnerabilities in Federated Learning	26
2.8	Conclusion	26
3	Federated Learning Threats and State-of-the-Art Countermeasures	27
3.1	Introduction	27
3.2	Adversarial Threats in Federated and Hierarchical Federated Learning (HFL)	27
3.2.1	Poisoning Attacks	28
3.2.2	Inference Attacks	32
3.2.3	Free-rider Attacks	34
3.2.4	Sybil Attacks	34
3.3	State of the Art Countermeasures	35
3.3.1	Statistical-Based Defenses	35
3.3.2	Accuracy Based Defenses	36
3.3.3	Hybrid Defenses	36
3.3.4	Trust Aware Aggregation Baselines	37
3.4	Conclusion	40
4	Trust Weighted Aggregation for Hierarchical Federated Learning	41
4.1	Introduction	41
4.2	System and Threat Model	41
4.3	The Behavioral Trust Metric	43
4.3.1	Components of the Trust Score	44
4.3.2	Behavioral Regulation: Trend and Volatility	45
4.4	Formulation of the Proposed Approach	45
4.5	Algorithm	66
4.6	Illustrative Example of Trust Evolution	76
4.7	Conclusion	78
5	Implementation and Experimental Results	79

5.1	Introduction	79
5.2	Software Environment	79
5.2.1	Libraries and Dependencies	80
5.2.2	Reproducibility	81
5.3	Simulation Architecture	81
5.4	Datasets	82
5.4.1	MNIST	82
5.4.2	FashionMNIST	83
5.4.3	CIFAR-10	84
5.4.4	Preprocessing	86
5.4.5	NON-IID Data Partitioning	86
5.5	Neural Network Model	86
5.5.1	SimpleCNN - MNIST and FashionMNIST	87
5.5.2	VGG9 - CIFAR-10	88
5.5.3	Training Configuration	90
5.6	Threat Model	91
5.6.1	Adversary Assumptions	91
5.6.2	Attack Taxonomy	92
5.6.3	Attack Descriptions	92
5.6.4	Experimental Coverage	95
5.7	Implementation Details	95
5.7.1	Baseline Aggregators	96
5.7.2	Evaluation Metrics	96
5.8	Experiments and Results	97
5.8.1	Experiment 1 : Clean Baseline	97
5.8.2	Experiment 2 : MNIST Dataset	100
5.8.3	Experiment 3 : FASHION-MNIST Dataset	103
5.8.4	Experiment 4 : CIFAR-10	106
5.9	Comparison	108
5.10	Experiment 5 : Mixed Attack	109

5.11 Discussion	111
5.12 Conclusion	112
6 General Conclusion	113

List of Figures

2.1	Types of machine learning.	8
2.2	Comparison between Logistic Regression and Linear Regression	8
2.3	classification of the digits dataset	9
2.4	K-Means clustering using moon plot [17].	9
2.5	Reinforcement Learning[18].	10
2.6	Dataset preparation and splitting into training, validation, and test sets.	11
2.7	Feature matrix X of size $n \times m$ (n samples, m features) and label vector y of length n [21].	11
2.8	Epochs, Batches, and Iterations Relationship [26].	12
2.9	Learning Rate Scheduling [28].	13
2.10	Underfitting vs. Overfitting [30].	14
2.11	A biological neuron in comparison to an artificial neural network. (a) Brain neuron, (b) Artificial neuron, (c) Neuron & biological synapse, (d) Artificial neural network [33]. . .	15
2.12	Neural Network [35].	16
2.13	CNN architecture [37].	16
2.14	RNN Architecture [38].	17
2.15	federated learning architecture [43].	19
2.16	Categorization of Federated Learning [45].	21
2.17	Classification of FL based on network structures [47].	21
2.18	HFL Architecture [49].	23
3.1	Federated Learning: normal operation vs. adversarial scenario [62].	28
3.2	Poisoning attack in a federated learning environment[62].	28
3.3	Label Flipping Attack	29
3.4	Procedure of typical backdoor attacks on AI models [69].	30

3.5	Byzantine Generals' Problem.	31
3.6	Inference attacks in the federated learning environment	32
3.7	Reconstructed Image (Target 7).	33
3.8	Reconstructed Image (Target 0).	33
3.9	Free-riding attacks in the federated learning environment.	34
3.10	Sybil-Based Untargeted Data Poisoning Attack.	35
5.1	Three-tier hierarchical federated learning architecture. Edge devices transmit locally computed model updates to their assigned fog node, where TWA-HFL performs first-level trust-weighted aggregation. The resulting fog-level aggregates are forwarded to the cloud server for second-level aggregation, and the updated global model is broadcast back through the fog nodes to all edge clients for the next round.	82
5.2	A 10×10 grid of sample images from the MNIST dataset.	83
5.3	A 10×10 grid of sample images from the FashionMNIST dataset.	84
5.4	A 10×10 grid of sample images from the CIFAR-10 dataset.	85
5.5	Simple CNN Architecture	87
5.6	Vgg-9 Architecture	89
5.7	MNIST No Attack	97
5.8	FashionMNIST No Attack	98
5.9	CIFAR-10 No Attack	98
5.10	Label Flip attack results on MNIST under different malicious client ratios	100
5.11	Gradient Scaling attack results on MNIST under different malicious client ratios	100
5.12	ALE attack results on MNIST under different malicious client ratios	100
5.13	ALE (no cluster) attack results on MNIST under different malicious client ratios	101
5.14	Sign Flipping attack results on MNIST under different malicious client ratios	101
5.15	MinMax attack results on MNIST under different malicious client ratios	101
5.16	LIE attack results on MNIST under different malicious client ratios	102
5.17	Label Flip attack results on FashionMNIST under different malicious client ratios	103
5.18	Gradient Scaling attack results on FashionMNIST under different malicious client ratios	103
5.19	ALE attack results on FashionMNIST under different malicious client ratios	103
5.20	ALE (no cluster) attack results on FashionMNIST under different malicious client ratios	104
5.21	Sign Flipping attack results on FashionMNIST under different malicious client ratios	104

5.22	MinMax attack results on FashionMNIST under different malicious client ratios	104
5.23	LIE attack results on FashionMNIST under different malicious client ratios	105
5.24	Label Flip attack results on CIFAR under different malicious client ratios	106
5.25	Gradient Scaling attack results on CIFAR under different malicious client ratios	106
5.26	ALE attack results on CIFAR under different malicious client ratios	106
5.27	ALE (no cluster) attack results on CIFAR under different malicious client ratios	107
5.28	Sign Flipping attack results on CIFAR under different malicious client ratios	107
5.29	MinMax attack results on CIFAR under different malicious client ratios	107
5.30	LIE attack results on CIFAR under different malicious client ratios	108

List of Tables

3.1	Comparison of Robust Aggregation and Defense Mechanisms in Federated Learning . . .	39
4.1	Aggregation weights derived from trust scores.	47
4.2	Client updates and their L2 norms at round $t = 2$	49
4.3	Global direction similarity scores at round $t = 2$	50
4.4	Pairwise cosine similarity matrix.	52
4.5	Magnitude consistency scores	54
4.6	Trust history of C_1 over the observation window.	55
4.7	RawTrust history of C_1 over the sliding window.	57
4.8	Impact of Decay Values on Client Trust	60
4.9	Decay values as a function of consecutive bad rounds.	60
4.10	Trust score evolution for C_1 over three communication rounds.	62
4.11	Client update vectors and their normalized representations	63
4.12	Cosine Similarity Matrix $A_{ij}^{(t)}$ between client updates	63
4.13	Client trust values at round t	64
4.14	Trust Affinity Matrix	64
4.15	Enhanced Affinity Matrix $\overline{A}_{ij}^{(t)}$	64
4.16	Cluster assignments derived from affinity structure	65
4.17	Average trust score per cluster	65
4.18	Computational Complexity Comparison of Aggregation Methods	75
4.19	Behavioral indicator scores and trust values at round $t = 1$	76
4.20	Behavioral indicator scores and trust values at round $t = 2$	77
4.21	Behavioral indicator scores and trust values at round $t = 3$	77
4.22	Behavioral indicator scores and trust values at round $t = 4$	77

5.1	MNIST dataset summary	83
5.2	FashionMNIST dataset summary	84
5.3	CIFAR-10 dataset summary	85
5.4	SimpleCNN summary	88
5.5	VGG9 architecture summary	90
5.6	Attack taxonomy: category, primary objective, and estimated evasion difficulty against similarity-based and magnitude-based defences.	92
5.7	Baseline aggregation algorithms	96
5.8	Average accuracy over the last 5 rounds across all attack scenarios	109
5.9	Test Accuracy (%) under 10% Attack Ratio	110
5.10	Test Accuracy (%) under 40% Attack Ratio	110
5.11	Final 5 Rounds Accuracy Comparison for Different Attack Ratios	110
5.12	Final 5 Rounds Accuracy at 30% Attack	111

Listings

5.1	Automatic device selection.	79
5.2	Seed fixing for reproducibility.	81
5.3	Preprocessing pipeline applied to all datasets.	86
5.4	Non-IID shard-based data partitioning	86
5.5	SimpleCNN architecture for MNIST and FashionMNIST	87
5.6	VGG9 architecture for CIFAR-10	88

Acronyms

TERM	DESCRIPTION
ML	Machine Learning
MAE	Mean Absolute Error
MSE	Mean Squared Error
GD	Gradient Descent
BGD	Batch Gradient Descent
SGD	Stochastic Gradient Descent
MBGD	Mini-Batch Gradient Descent
DL	Deep Learning
CNN	Convolutional Neural Network
RNN	Recurrent Neural Networks
IoT	Internet of Things
GDPR	General Data Protection Regulation
FL	Federated Learning
Non-IID	Non-Independent and Identically Distributed
HIPAA	Health Insurance Portability and Accountability Act
HFL	Hierarchical Federated Learning
WMA	Weighted Median Aggregation
DSGD	Decentralized Stochastic Gradient Descent
SAGE	Spatially Aggregated Gradient Estimation
FIDS	Federated Intrusion Detection Systems
TWA-HFL	Trust-Weighted Adaptive Aggregation for Hierarchical Federated Learning
LIE	A Little Is Enough Attack
ALE	Adaptive Local Ensemble
GPU	Graphics Processing Unit

Chapter 1

General Introduction

The past decade has witnessed an unprecedented expansion of artificial intelligence across virtually every domain of modern life. Healthcare systems leverage machine learning to detect diseases from medical imaging and predict patient outcomes. Autonomous vehicles rely on deep neural networks to perceive their surroundings and make real-time decisions. Financial institutions deploy predictive models for fraud detection, credit scoring, and market analysis. Smart cities use sensor-driven algorithms to optimize traffic flow, energy consumption, and public safety. Across all these domains, the quality and breadth of the training data available to a model directly determine the reliability of its predictions. As a result, the demand for large, diverse, and representative datasets has become one of the central drivers of AI development [1][2].

Traditionally, this demand has been met through centralized data collection, in which raw data from distributed sources is aggregated onto a single server where model training takes place. While this paradigm has enabled many of the landmark achievements in deep learning, it carries fundamental limitations that are increasingly difficult to ignore. Centralizing raw data from millions of devices or organizations creates serious privacy risks: a single breach can expose sensitive personal or institutional information at scale. Legal frameworks such as the European General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA) impose strict constraints on the collection, storage, and cross-border transfer of personal data, creating compliance barriers that centralized learning often cannot satisfy. Beyond privacy, the sheer volume and velocity of data generated at the edge of modern networks makes transferring raw data to a central server increasingly impractical, both in terms of communication bandwidth and latency [3][4].

Federated Learning (FL) emerged as a principled response to these challenges. Rather than moving raw data to the computation, FL brings the computation to the data: clients train models locally on their private datasets and share only model updates, such as gradients or weight differences, with a central server. The server aggregates these updates to produce an improved global model, which is then redistributed to clients for the next training round. This communication protocol preserves data locality by design and has been shown to be applicable across a wide range of domains where data sensitivity is a primary concern[5][6].

Hierarchical Federated Learning (HFL) extends this paradigm by introducing an intermediate aggregation

layer between clients and the cloud server, typically realized as fog or edge nodes. This three-tier architecture reduces the communication overhead associated with transmitting model updates directly to a distant cloud server and is particularly well-suited to Internet-of-Things (IoT) deployments, in which large numbers of resource-constrained devices are physically clustered around edge infrastructure. By performing a first level of aggregation at the fog tier, HFL reduces the number of updates that need to traverse the wide-area network while preserving the privacy guarantees of the original federated paradigm.[7]

Problem Statement

Despite these advantages, federated and hierarchical federated learning introduce a critical security vulnerability that centralized learning does not share: the server has no direct visibility into the local training procedure of any individual client. It can observe only the model updates that clients choose to submit. This constraint, which is inseparable from the privacy guarantee that motivates FL in the first place, creates an opportunity for malicious participants to corrupt the learning process by submitting carefully crafted updates that steer the global model away from its intended objective[8].

Such adversaries, broadly referred to as Byzantine clients, may attempt a wide range of attack strategies. Data poisoning attacks corrupt a client's local training data before training, causing the resulting gradient to reflect a miscalibrated objective. Model poisoning attacks directly manipulate the update before submission, for instance by scaling the gradient magnitude to dominate the aggregation or by flipping gradient signs to perform gradient ascent rather than descent. Sophisticated adaptive attacks, such as the Adaptive Local Ensemble (ALE) attack, are specifically designed to evade common defenses by preserving the directional signature of a benign update while concealing the malicious signal in a different behavioral dimension. In Sybil attack scenarios, a single adversary controls multiple client identities to amplify its influence over the aggregation outcome[9][10].

The straightforward aggregation rule introduced by FedAvg computes a weighted average of all submitted updates, making no distinction between honest and malicious contributions. As a result, even a small number of Byzantine clients can substantially degrade global model performance or introduce targeted misclassifications. A variety of robust aggregation methods have been proposed to address this limitation, including distance-based selection rules (Krum, Multi-Krum), coordinate-wise statistical filters (Trimmed Mean, Median), and trust-based approaches (FLTrust, FoolsGold, reputation-based FL). While these methods offer meaningful improvements over FedAvg in specific threat scenarios, they share two fundamental limitations that reduce their reliability in practice[5][11].

The first limitation is their *stateless* nature. Most existing methods evaluate each communication round in isolation, assigning scores or making filtering decisions based solely on the updates submitted in that round, without consulting any record of each client's past behavior. An adversary that behaves honestly for several rounds before mounting a coordinated attack, or that alternates between benign and malicious rounds to evade detection, can therefore avoid penalization indefinitely. Conversely, an honest client whose update appears statistically anomalous in a single round due to the natural variance introduced by Non-Independent and Identically Distributed (Non-IID) data distributions may be unfairly discarded by

a defense that has no memory of the client’s prior honest contributions[10].

The second limitation is their *single-dimension* assessment criterion. Methods based on directional similarity, such as FLTrust, can be bypassed by attacks that preserve gradient direction while amplifying update magnitude. Methods based on norm filtering can be bypassed by attacks that keep the update norm within an acceptable range while subtly altering its direction. No single behavioral signal is sufficient to characterize the full space of adversarial strategies that a determined attacker can deploy in an adaptive threat model[12][13].

These limitations collectively motivate the need for an aggregation framework that evaluates client trustworthiness across *multiple complementary dimensions* and *over time*, making irreversible exclusion decisions unnecessary by replacing hard filtering with a continuously evolving, history-aware trust score that adapts to each client’s observed behavioral trajectory.

Objectives

The primary objective of this thesis is to design, formalize, and experimentally evaluate TWA-HFL (Trust-Weighted Adaptive Aggregation for Hierarchical Federated Learning), a robust aggregation framework that addresses the shortcomings identified above. The framework is designed to simultaneously achieve four goals:

1. **Reduce the influence of malicious or suspicious clients** on the global model by assigning lower aggregation weights to participants whose behavioral indicators deviate from those expected of honest contributors, without requiring their permanent exclusion.
2. **Preserve the contributions of honest clients** whose updates may occasionally appear anomalous due to data heterogeneity or local training variance, by maintaining a trust history that moderates the impact of isolated anomalies on a client’s long-term reputation.
3. **Detect sophisticated adaptive adversaries** that attempt to evade single-dimension defenses by evaluating client updates simultaneously across four behavioral dimensions: global-direction similarity, peer-cluster consistency, update magnitude, and historical trust stability.
4. **Maintain competitive convergence efficiency** alongside improved robustness, operating within the same asymptotic complexity class as the strongest existing robust aggregators while providing substantially richer behavioral information.

Proposed Approach

To achieve these objectives, TWA-HFL introduces a dynamic per-client trust score that evolves continuously across training rounds. At each communication round, four normalized behavioral indicators are computed for each client. Global-direction similarity measures how well the client’s update aligns with the recent trajectory of the global model. Peer-cluster similarity captures the directional agreement

between the client’s update and those of its peers in the current round. Magnitude consistency detects abnormal update norms by comparing each client against a robust lower-half median reference, making the indicator resistant to a fraction of extreme outliers. Historical trust stability penalizes clients whose accumulated trust scores have fluctuated erratically across previous rounds.

These four indicators are fused into a single RawTrust score via a fixed weighted combination and stored in a per-client sliding window. A temporal regulation mechanism then estimates the linear trend and variance of each client’s recent RawTrust history to distinguish improving behavior from deteriorating or oscillating behavior. Clients demonstrating sustained improvement are rewarded with a small trust bonus, while clients exhibiting persistent degradation are subjected to a progressively amplified decay that reduces the weight of their historical trust in future rounds.

The final trust score is computed as an exponential moving average that balances historical reputation against the current round’s assessment. Prior to aggregation, a trust-enhanced spectral clustering step partitions clients into two groups by combining cosine similarity between updates with pairwise trust affinities, and retains only the higher-trust cluster. The retained updates are norm-clipped to the median of the trusted set and combined via a stabilized softmax weighted average, ensuring that no single client can dominate the aggregation through magnitude amplification while preserving the contribution of every retained participant.

Contributions

The main contributions of this thesis are as follows:

- **A multi-dimensional behavioral trust metric** that simultaneously evaluates client updates along four complementary axes, preventing adversaries from bypassing the defense by manipulating only one behavioral signal.
- **A temporal regulation mechanism** that integrates trend and volatility analysis over a sliding history window, enabling detection of subtle, persistent, or intermittent adversarial strategies that evade single-round evaluations.
- **A trust-enhanced spectral clustering procedure** that leverages both geometric similarity and accumulated trust reputation to isolate suspicious participant groups, providing robustness against coordinated Byzantine and Sybil attacks without permanently excluding any client.
- **A comprehensive empirical evaluation** of TWA-HFL against six established aggregation baselines across seven Byzantine attack types at four malicious client ratios, on three benchmark datasets of increasing visual complexity, totaling 203 independent experimental runs conducted under a fully reproducible setup.

Experimental Evaluation

To assess the robustness and practical effectiveness of the proposed framework, a comprehensive empirical evaluation was conducted on MNIST, FashionMNIST, and CIFAR-10, using a simulated three-tier HFL topology composed of 100 edge clients distributed across five fog nodes. TWA-HFL was benchmarked against FedAvg, Krum, TrimmedMean, FLTrust, FoolsGold, and RFL under seven attack types: Label Flip, Gradient Scale, ALE, ALE without cluster alignment, Sign Flip, MinMax, and LIE, at malicious ratios of 10%, 20%, 30%, and 40%.

The results demonstrated that TWA-HFL achieves the highest average accuracy across all datasets and attack configurations, attaining 94.27% on MNIST, 78.68% on FashionMNIST, and 50.79% on CIFAR-10, averaged over all conditions. Under the most challenging adaptive attack (ALE at 40% malicious clients), all six baseline aggregators collapsed to near-random accuracy on MNIST while TWA-HFL maintained 96–97%. A dynamic mixed-attack scenario, in which each malicious client randomly selected an attack type at every round, was introduced as an additional stress test: on CIFAR-10 under this setting, all baseline methods reached random-guess performance while TWA-HFL held at 42.83% accuracy, validating the robustness of its multi-dimensional and history-aware defense under highly unpredictable adversarial conditions.

Thesis Organization

The remainder of this thesis is organized into four chapters, as follows.

Chapter 2 – Background of Federated Learning. This chapter provides the theoretical foundations necessary to understand the proposed framework. It begins with the fundamentals of machine learning and deep learning, covering supervised, unsupervised, and reinforcement learning paradigms along with core training concepts. It then introduces the motivation for decentralized learning, the architecture and training workflow of federated learning, and the extension to hierarchical federated learning. The chapter concludes by reviewing the principal challenges facing FL and HFL systems, including statistical heterogeneity, systems heterogeneity, communication overhead, and security vulnerabilities.

Chapter 3 – Federated Learning Threats and State-of-the-Art Defense Mechanisms. This chapter surveys the adversarial landscape of federated learning. It presents a taxonomy of attack types targeting FL systems, covering poisoning attacks (data poisoning, model poisoning, backdoor attacks), inference attacks, free-rider attacks, and Sybil attacks. It then critically reviews existing robust aggregation mechanisms, categorizing them as statistical-based defenses, accuracy-based defenses, hybrid defenses, and trust-aware aggregation methods, identifying the limitations that motivate the proposed approach.

Chapter 4 – Trust-Weighted Aggregation for Hierarchical Federated Learning. This chapter presents the TWA-HFL framework in full detail. It defines the system and threat model, describes the multi-dimensional behavioral trust metric and its four components, and provides the complete mathematical formulation of the RawTrust score, temporal regulation mechanisms, trust update rule, trust-enhanced spectral clustering, norm clipping, and trust-weighted aggregation. The algorithmic workflow is presented

alongside a step-by-step computational complexity analysis and a comparison with the state-of-the-art defense mechanisms. The chapter closes with an illustrative trust evolution example tracing five clients across four rounds.

Chapter 5 – Implementation and Experimental Results. This chapter describes the software environment, simulation architecture, datasets, neural network models, training configurations, threat model, and baseline aggregators used in the evaluation. It then presents the full experimental results across all datasets, attack types, and malicious ratios, followed by a comparative analysis and a discussion of the mixed-attack scenario. The chapter concludes by interpreting the results in light of the design objectives.

The thesis closes with a **General Conclusion** that summarizes the contributions, discusses the limitations of the current work, and identifies five directions for future research.

Chapter 2

Background of Federated Learning

2.1 Introduction

Although AI has advanced significantly, its reliance on centralized machine learning causes problems such as privacy risks, limited scalability, and high communication costs. Federated Learning (FL) aims to overcome these limitations through a decentralized approach. In this chapter, we provide a comprehensive background on FL and highlight the transition toward Hierarchical Federated Learning (HFL) in Internet of Things (IoT) environments.

This chapter is organized as follows. Section 2.2 presents the fundamentals of machine learning (ML) and deep learning (DL). Section 2.3 discusses the need for decentralized learning. Section 2.4 introduces FL and its main concepts. Section 2.5 presents HFL. Section 2.6 highlights the main challenges in FL and HFL. Section 2.7 discusses their applications. Finally, Section 2.8 concludes the chapter.

2.2 Fundamentals of Machine Learning

To comprehend the principles and factors underlying FL, it is essential to first review the basic concepts of ML and DL as they form the foundation for more advanced approaches.

2.2.1 ML Definition

Machine learning is a branch of artificial intelligence that studies how computers can mimic human learning processes, acquire knowledge and skills, identify patterns, and continuously improve their performance. It can be defined as a collection of algorithms and techniques used to develop computational systems capable of learning from raw data in order to make predictions and inferences[14].

2.2.2 Learning Paradigms

ML tasks can be categorized into three main groups, as shown in Figure 2.1 [15], based on the nature of the input data and the type of output to be produced.

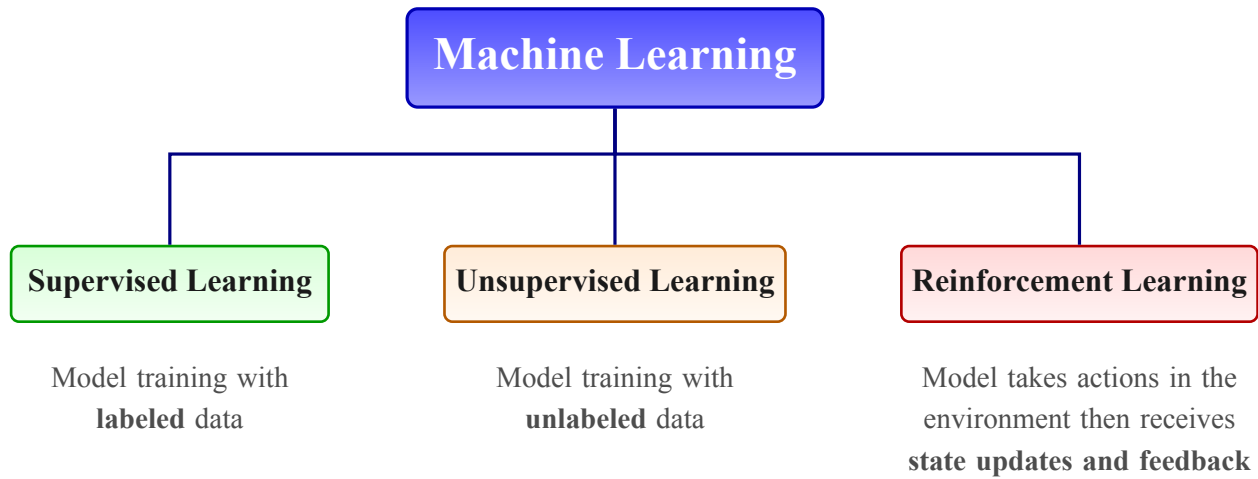
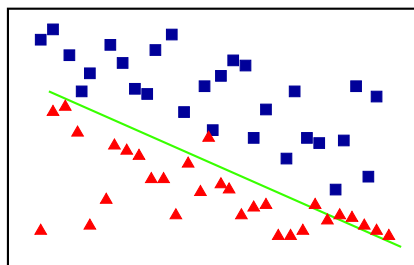


Figure 2.1: Types of machine learning.

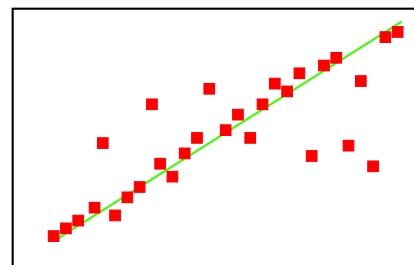
a. Supervised Learning:

Supervised machine learning uses labeled data from past and current examples to make predictions. It works by training models on datasets where the correct outputs are already known. The model learns a function that maps inputs to outputs. Once trained, it can predict results for new data. It also compares its predictions with the real values to measure errors and improve its performance over time [16]. There are two commonly used types of supervised learning algorithms:

Regression: Predicting a continuous numerical value from input data. Linear regression predicts a continuous numerical value (like price or temperature), while logistic regression predicts a probability used for classification (usually between 0 and 1) to decide between categories.



(a) Logistic Regression



(b) Linear Regression

Figure 2.2: Comparison between Logistic Regression and Linear Regression .

Classification: Assigning an input to a specific category or class.

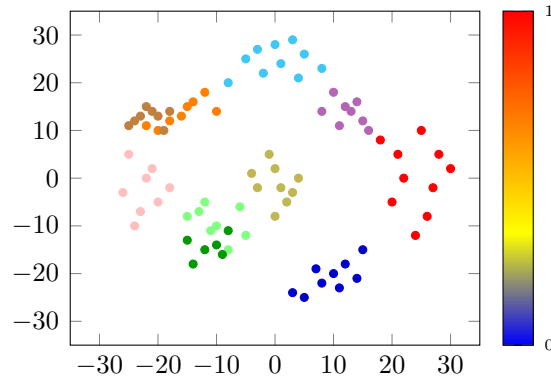


Figure 2.3: classification of the digits dataset .

b. Unsupervised Learning:

Unsupervised ML techniques are employed when the training data is unclassified and not labeled. The system does not rely on predefined outputs, but instead explores the dataset aiming to discover underlying structures or relationships that explain hidden structures in the data[16].

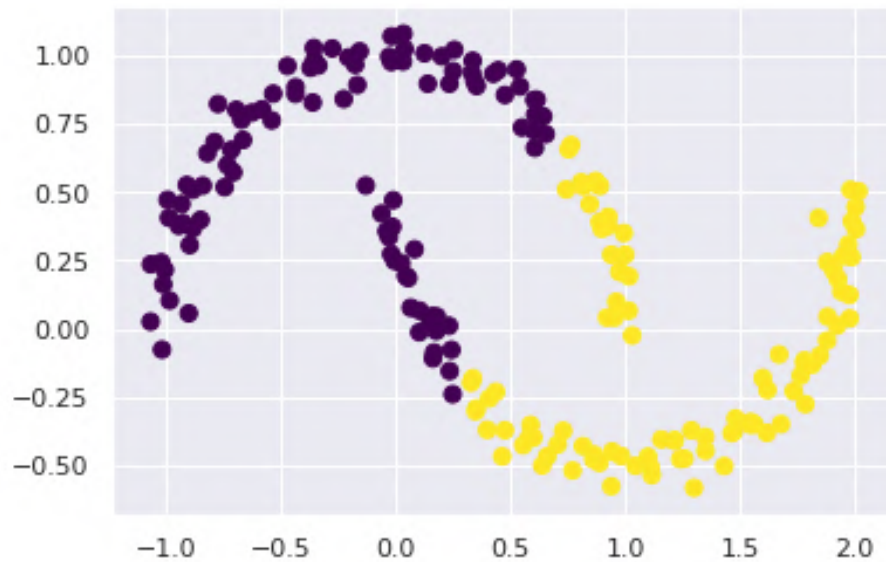


Figure 2.4: K-Means clustering using moon plot [17].

c. Reinforcement Learning:

The goal of reinforcement learning is to make the best decision in a given environment by learning from the rewards and penalties received. While selecting actions, the algorithm should not only consider the immediate reward but also the next and all subsequent rewards. For example, a program designed to play a game or drive a car must constantly interact with a dynamic environment in which it is expected to achieve a specific goal [14].

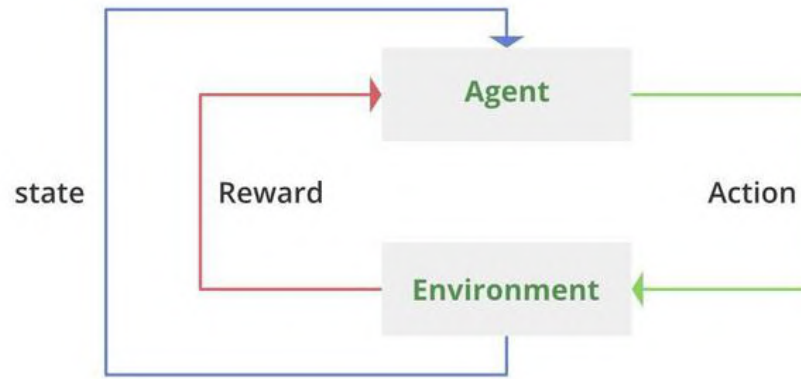


Figure 2.5: Reinforcement Learning[18].

- The agent and the environment produce the basic interaction between them.
- The agent monitors the state of the environment and then performs an action.
- The environment receives the action and produces a new state.

Examples of reinforcement learning techniques include:

- Q-learning
- Temporal difference methods
- Monte Carlo methods

2.2.3 Core Training Concepts

In machine learning training, several criteria are used to reduce errors and improve performance in order to build a model with accurate results. In this section, we discuss some of these criteria.

a. Dataset Splitting:

When the data set is large, it is better to split it into three parts: a training set, a validation set, and a testing set [19].

- **Training set:** This is used to train a machine learning model, it typically represents the largest portion of the dataset, and it usually represents between 70% and 80%.
- **Validation set:** It is a subset from the training set, used to evaluate the prediction error and the performance of different models in order to select the best one by using hyperparameters.
- **Test set:** It is used to evaluate the efficiency and performance of the trained model, and it usually represents between 10% and 20% of the dataset.

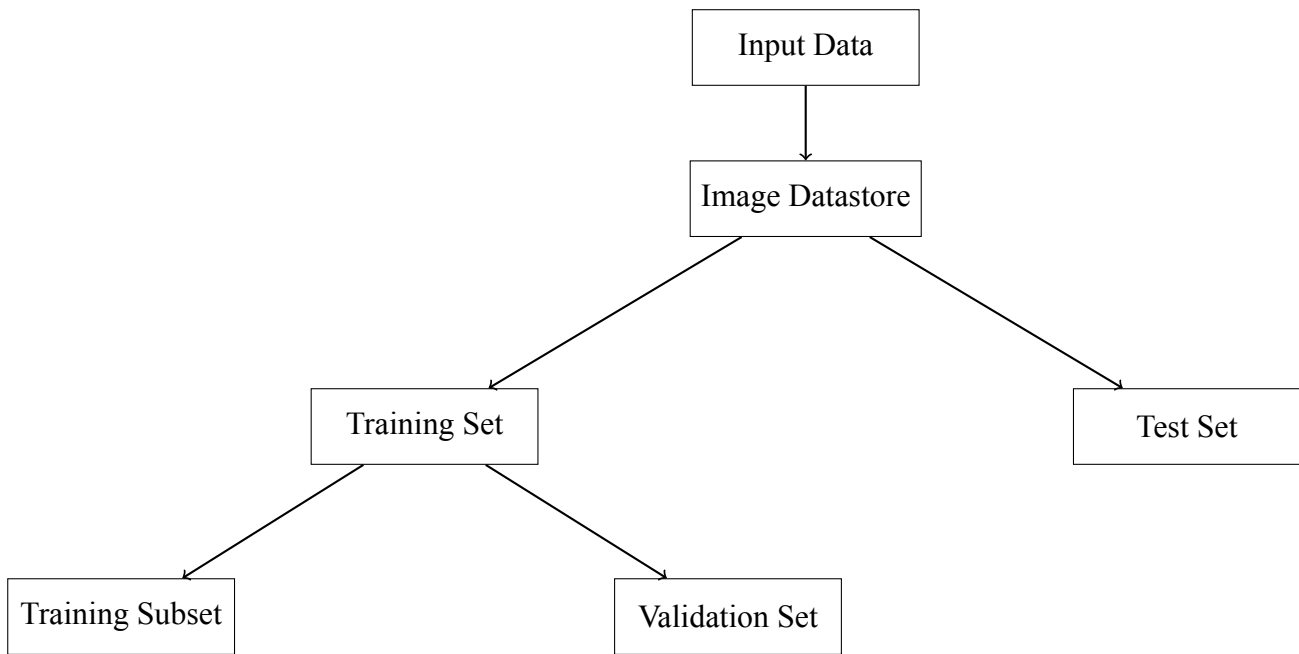


Figure 2.6: Dataset preparation and splitting into training, validation, and test sets.

b. Features and Labels:

In machine learning, models learn from input output data pairs. Features refer to attributes or predictors, which are the input variables used by the model to make predictions. Meanwhile, labels, also referred to as target variables or outcomes, represent the values that the model is trained to predict [20].

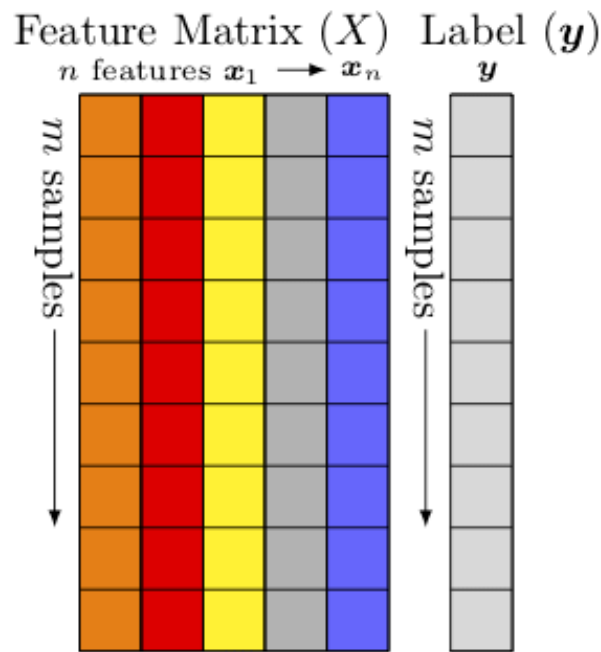


Figure 2.7: Feature matrix X of size $n \times m$ (n samples, m features) and label vector y of length n [21].

c. Model Training:

Model training is the process of training a ML model to improve its performance on a training dataset. It is considered one of the most important steps in ML, as it involves adjusting the model's parameters, measuring the loss, and evaluating performance on validation datasets. These parameters include the weights and biases in the mathematical functions that define the algorithm. This optimization aims to produce more accurate outputs. The process is repeated iteratively until satisfactory results are achieved and may also involve adjusting hyperparameters [22].

d. Epoch and Iteration:

An **epoch** refers to one complete pass through the entire training dataset where every data sample is passed through the model and its parameters are updated based on the calculated error. The training process requires multiple epochs, allowing the model to improve iteratively by adjusting its parameters based on the calculated error [23].

An **iteration** is a single update of the model's parameters during model training. Iterations occur once the model processes one batch of data and updates the weights based on the error computed for that batch. Each epoch consists of multiple iterations because the data are processed in batches [24].

e. Batch Size:

Batch size refers to the number of samples used in training a machine learning model, where the training data is usually divided into smaller batches, each of which is processed independently before updating the model parameters. It can also have a positive or negative impact on the speed and accuracy of machine learning training [25].

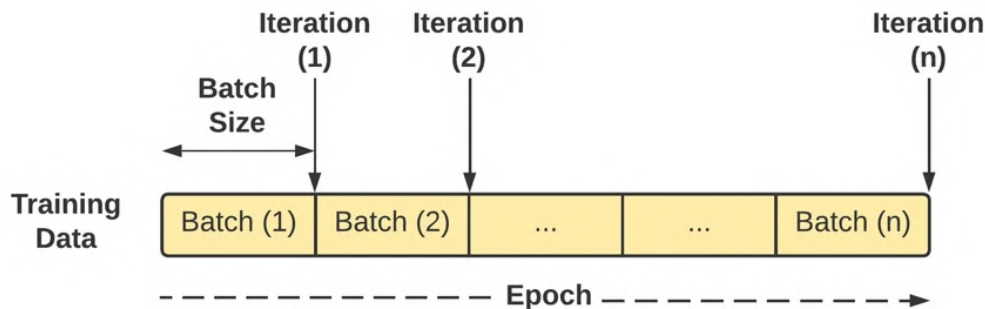


Figure 2.8: Epochs, Batches, and Iterations Relationship [26].

f. Learning Rate:

Choosing an appropriate learning rate is one of the most important criteria in ML. An appropriate value must be selected because a slow learning rate causes the model to take a long time and a large number of iterations to converge, whereas a high learning rate causes the model not to converge at all [27].

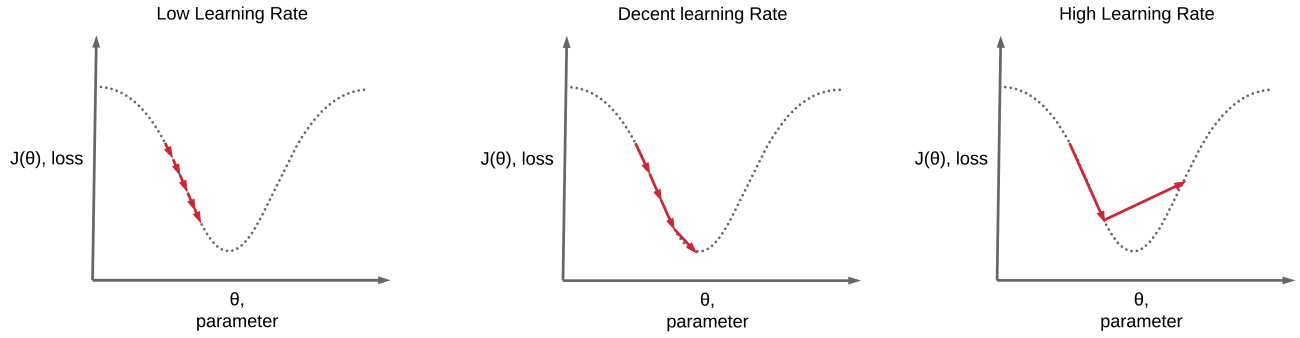


Figure 2.9: Learning Rate Scheduling [28].

g. Loss Functions:

The loss function measures the difference between a prediction and the actual value to evaluate model performance. If the difference is very large, that means the prediction is poor, while a small difference indicates a good prediction. Some of the most commonly used functions are MAE and MSE [29].

- **Mean Absolute Error (MAE):**

MAE, also referred to as the L1 loss, is another common choice for regression tasks. It measures the average of the absolute differences between the predicted values $\hat{Y}_i$ and the target values Y_i :

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |Y_i - \hat{Y}_i|$$

- **Mean Squared Error (MSE):**

MSE, also referred to as the L2 loss, is one of the most widely used regression loss functions. It is defined as the average of the squared differences between the predicted values $\hat{Y}_i$ and the target values Y_i :

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

h. Overfitting and Underfitting:

Overfitting refers to the problem when the model performs well on the training set but poorly on the test set. There could be many reasons causing overfitting, such as high feature dimensions, complex model assumptions, excessive parameters, limited training data, and excessive noise. But essentially, the overfitting problem is mainly caused by the model overfitting the training set without considering the generalization ability of the network. Consequently, the model can predict the training set well but always fails to predict new data [27].

Underfitting refers to a situation where there are too few predictive variables in a statistical machine learning model, resulting in an overly simple model that fails to capture the underlying data distribution.

This problem may also arise when the training dataset is too small or not representative of the overall statistical population. An underfitted model performs poorly on the training data and therefore does not generalize well to unseen data. As a result, its predictive performance on new cases is weak due to insufficient learning of data patterns [19].

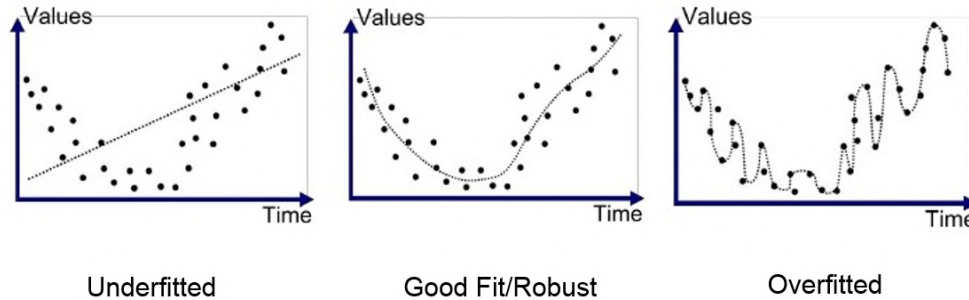


Figure 2.10: Underfitting vs. Overfitting [30].

i. Gradient Descent and Optimization:

Optimization is an important branch that aims to conceptualize, analyze, and solve problems of minimization or maximization of a function on a specific dataset. Several optimization algorithms are discussed in ML and particularly in DL-based systems, such as the gradient descent (GD) algorithm.

GD is one of the most popular optimization algorithms and the most widely used method to optimize parameters for machine learning. It approximates the solution using an iterative approach that adjusts the parameters gradually in order to minimize the cost function. Gradient descent has three main variants, which differ in how much data is used to compute the gradient of the objective (loss) function during each update step. **Batch Gradient Descent (BGD)** computes the gradient using the entire training dataset, which leads to stable but computationally expensive updates. **Stochastic Gradient Descent (SGD)**, on the other hand, updates the model parameters using a single training example at a time, making it much faster but more noisy and less stable. **Mini-Batch Gradient Descent (MBGD)** provides a balance between the two by computing the gradient using a small subset (batch) of the training data, offering a good trade-off between efficiency and stability.

Depending on the amount of data, a trade-off is made between the accuracy of the parameter update and the time required to perform an update. Some of the optimization algorithms available are Momentum, Adagrad, RMSprop, Adam, and Adamax [31].

2.2.4 Deep Learning Fundamentals

Deep learning (DL) is a subfield of machine learning. Unlike traditional machine learning approaches, where feature extraction is often performed manually, deep learning models automatically learn relevant features from raw data without human intervention. Deep learning is based on artificial neural networks (ANNs) and is widely used in applications such as prediction, classification, object detection, and lan-

guage translation [32].

a. Neural Networks:

Generally speaking, DL is based on deep neural networks, namely multi-layer neural networks composed of perceptrons, a perceptron simulates neurons in the human brain.

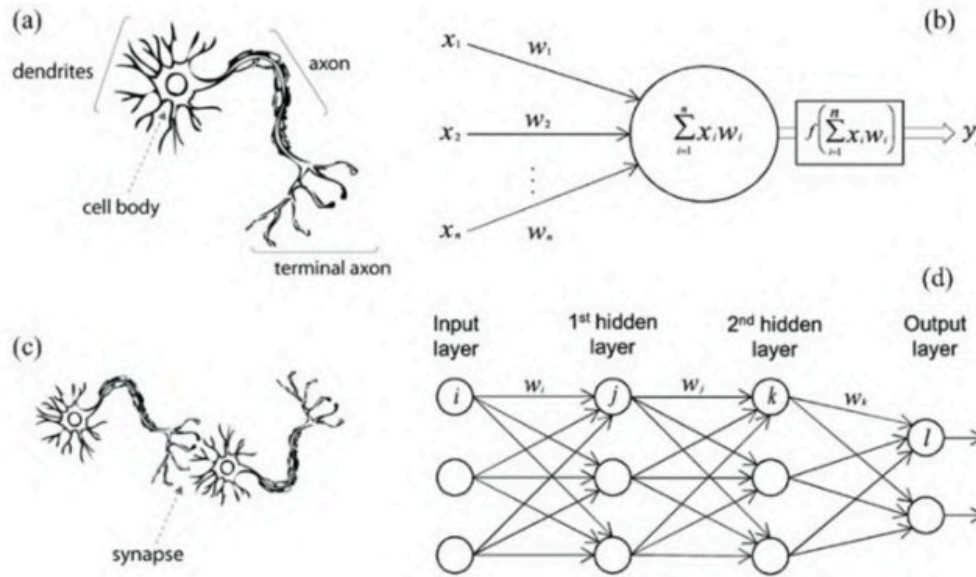


Figure 2.11: A biological neuron in comparison to an artificial neural network. (a) Brain neuron, (b) Artificial neuron, (c) Neuron & biological synapse, (d) Artificial neural network [33].

According to the American neural network scientist Robert Hecht-Nielsen, a neural network is a computing system made up of a number of very simple, highly interconnected processing elements, which are organized in a specific way to process information through their dynamic state response to external inputs [34].

Combining the origins, characteristics, and various interpretations of neural networks, it can be simply described as an information processing system designed to imitate the structure and functions of the human brain. Artificial neural networks exhibit fundamental characteristics of the human brain, such as parallel information processing, learning, association, classification, and memorization.

Neural networks are composed of multiple layers of neurons, including an input layer, one or more hidden layers, and an output layer, as shown in Figure 2.12. Each node has an associated weight used during model training. These networks are widely used in various applications due to their ability to learn from large datasets, such as classification, prediction, face recognition, and translation [32].

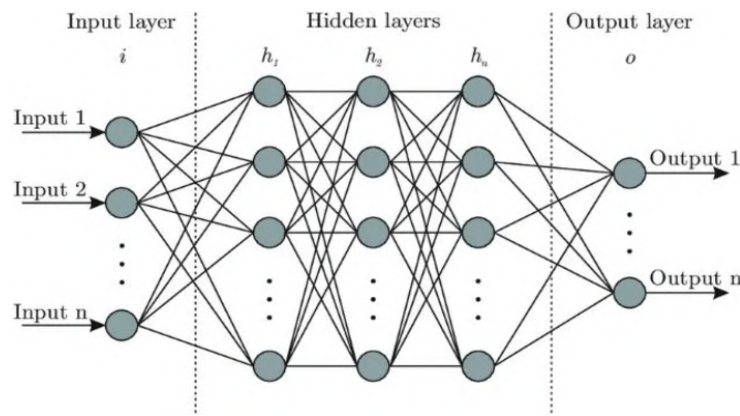


Figure 2.12: Neural Network [35].

b. Deep Learning Models:

DL consists of diverse architectural paradigms, among which Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are particularly prominent. This section provides a concise overview of these architectures [36].

- **Convolutional Neural Network (CNN):**

CNNs have become one of the widely discussed subjects in many scientific fields, particularly in pattern recognition. As the structure of CNNs can avoid complex image preprocessing and can directly take the original image as input, they have been extensively applied in many fields.

CNN contains three types of layers: convolutional, pooling, and fully connected layers, as shown in Figure 2.13. In each CNN, there are two stages in the training process: the feed-forward stage and the backpropagation stage. The most common CNN architectures are ZFNet, GoogLeNet, VGGNet, AlexNet, and ResNet.

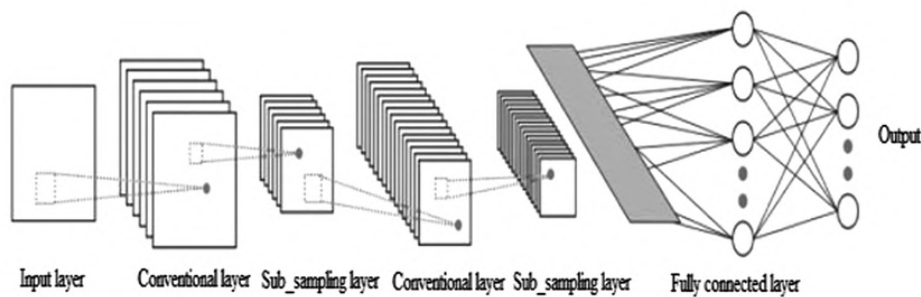


Figure 2.13: CNN architecture [37].

- **Recurrent Neural Networks (RNN):**

RNNs are designed to recognize sequences and patterns such as speech, handwriting, text, and such applications. RNN utilizes cyclic connections in the structure which employ recurrent computations to sequentially process the input data. RNN is basically a standard neural network that has been

extended across time by having edges which feed into the next time step instead of into the next layer in the same time step as shown in Figure 2.14 . Each of the previous input data is kept in a state vector in hidden units, and these state vectors are utilized to compute the outputs.

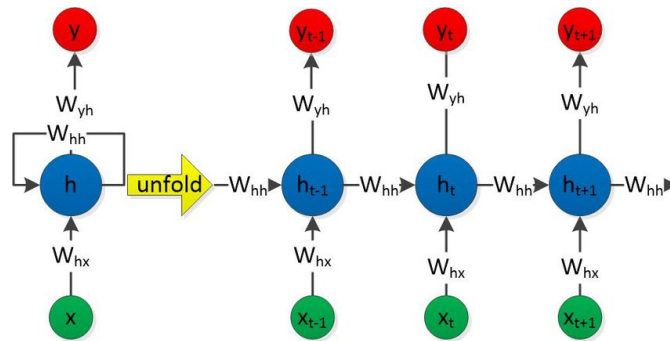


Figure 2.14: RNN Architecture [38].

c. Generalization and Regularization:

In ML, the model faces several common problems such as overfitting, underfitting and generalization. To address these challenges, various regularization techniques are used to improve the model's performance and accuracy.

- **Generalization** is the ability of the model to react as it did during training when fresh data was provided to the model. That fresh data is taken as training data. In simple words, generalization is the ability of a model to provide accurate predictions once it is trained by using the training data. A model that has the ability to generalize is considered a successful model. If we overtrain the model, then the model may not generalize and hence may lose the ability to learn from new data. The over-trained model will make inaccurate predictions when new data is provided to it. Similarly, if the model has an underfitting problem, then again the model will not generalize [27].
- **Regularization** is a rather important and highly effective measure to reduce generalization error in machine learning. Compared to the traditional machine learning model, the capacity of deep learning models is larger, so it is more likely to lead to overfitting. To this end, researchers have worked out several useful techniques to prevent overfitting, including:
 - Parameter norm penalty, which adds norm constraints such as L1 and L2.
 - Dataset expansion, such as adding noise and changing data.
 - Dropout: randomly discarding a part of the output of neurons during the training phase. The parameters of these discarded neurons will not be updated.
 - Early stopping of training should be allowed: by examining the validation dataset regularly, and when the loss function of the validation dataset starts to rise, training can be stopped in advance to avoid overfitting [32].

2.3 Motivation for Decentralized Learning

Decentralized learning facilitates collaborative privacy preserving machine learning without the vulnerabilities and trust issues that centralized systems instantiate, drawing upon distributed protocols to coordinate learning across diverse and autonomous agents.

Centralized learning assumes access to general state information and element paths during training. In this centralized approach, training takes place on a single, powerful central node. IoT nodes share their raw data with this central node, enabling the use of the entire dataset to build the model. However, this centralized learning has many limitations [39]:

- **Privacy Concerns and Data Sensitivity:** Ensuring data privacy and security is of paramount importance, as traditional centralized machine learning methods have long been associated with risks to personal data privacy, including leaks, misuse, and exploitation. In centralized machine learning systems, raw data must be transferred to a central server for processing, exposing personal information to these risks [40].
- **Data Silos and Legal Barriers:** Collecting, aggregating, and integrating heterogeneous data dispersed across various sources, as well as securely managing and processing it, are non-trivial tasks. The challenges arise not only from transporting high-volume, high-velocity, high-veracity, and heterogeneous data across organizations, but also from industry competition, complex administrative procedures, and, fundamentally, data protection regulations and restrictions, such as the EU GDPR [40].
- **Scalability and Communication Bottlenecks:** As the number of clients and the volume of data increase, the challenge of scalability intensifies, since a large number of clients can create bottlenecks and limitations in both computational capacity and network connectivity, which may result in increased latency and reduced efficiency [41].
- **Single Point of Failure:** In centralized learning, the entire training process takes place on a single central server. This presents a serious security vulnerability, as the entire training process could cease if the central server fails or is compromised, leading to significant losses, such as the loss or corruption of training data [41].

2.4 Federated Learning

In the early stages of machine learning, when the amount of data was relatively small, the predominant paradigm was centralized learning. However, with the emergence of big data and advancements in computing power, the limitations of centralized learning became increasingly apparent. Training large-scale models with vast amounts of data on a single server became impractical due to computational constraints. To address these challenges, distributed learning techniques were introduced, enabling the training process to be performed across multiple computing nodes.

More recently, growing concerns about data privacy and security have driven the development of new learning paradigms, among which FL has gained significant attention. In what follows, we present the fundamental concepts of FL.

2.4.1 Definition and Core Concepts

FL is a type of decentralized machine learning where clients can jointly realize a model without sharing their sensitive data with a central server. With FL, in contrast to a centralized learning paradigm where data are gathered and stored on a central server, only model updates, for example, gradients or parameters are transmitted and not the raw data itself, keeping the original data local.

FL is related to distributed learning in that both use a cluster of computing nodes and a parameter server to aggregate results from those nodes. However, they differ in several important aspects. The distribution and quantity of data are not centrally controlled and may be unknown if all training data is kept private. FL also operates under Non-IID conditions, meaning that each client holds its own distinct data. FL offers several advantages. It ensures compliance with data protection regulations, including the GDPR and HIPAA, boosts security and privacy, makes FL applicable to analytics-sensitive applications in banking, healthcare, smart cities etc. while lowering data expenses as well improving personalization [42].

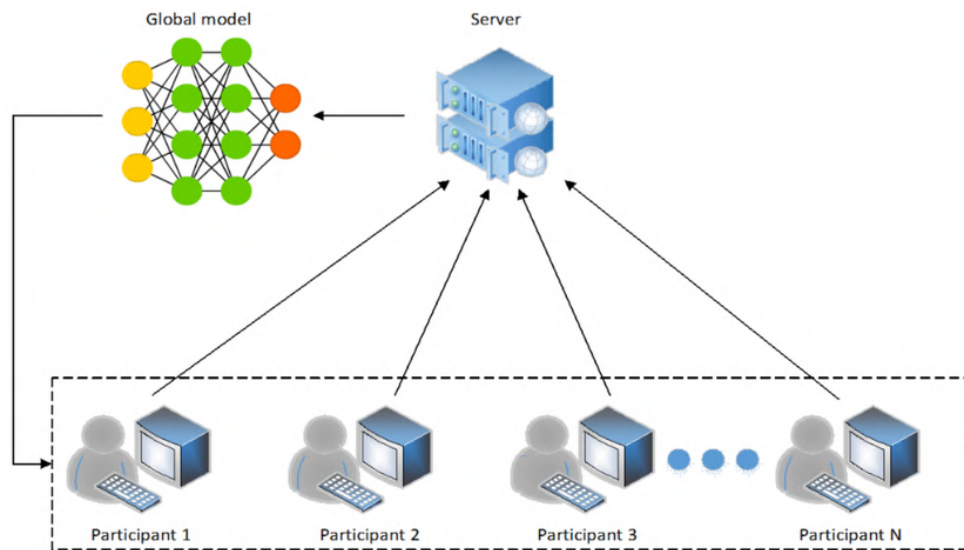


Figure 2.15: federated learning architecture [43].

2.4.2 FL Training Workflow

The training process in the unified learning system occurs in four stages, and they are as follows [44]:

1. **Client Selection and Model Initialization:** At the beginning of each training round, the central server commences each round by randomly selecting a subset of its clients to participate. It then transmits the hyperparameters of the global model's current state to each of the selected clients.

2. **Local Training:** Each selected client receives the current global model and proceeds to train it locally using its own private data. This process is carried out over several local epochs.
3. **Model and Gradient Update Sharing:** Each client transmits the resulting update to the central server, expressed as the difference between the locally updated weights and the initial global model weights.
4. **Aggregation at a Central Server:** The central server receives updates sent from clients and aggregates them using an aggregation algorithm to rebuild an updated global model. The updated model is then redistributed to clients, and this process is repeated until a specified number of rounds is reached.

2.4.3 FL Taxonomies

FL systems can be categorized into different taxonomies depending on data partitioning, system architecture, and scale of participation. These classifications provide a structured understanding of FL settings and their associated challenges.

a. Data Partitioning

Data partitioning in FL describes how data is distributed across multiple clients and is generally classified into horizontal, vertical, and federated transfer learning based on the relationship between feature space and sample space [6].

- **Horizontal Federated Learning**, also referred to as sample-based FL, is applicable in scenarios where datasets share the same feature space while collecting data from different user populations, as shown in Figure 2.16a. For example, data from different universities may share the same feature space, namely, information related to faculties and academic disciplines, yet differ in their sample space, as the data corresponds to distinct students and faculty members.
- **Vertical federated learning**, or feature-based federated learning, is applicable to cases in which two datasets share the same sample ID space but differ in feature space, as shown in Figure 2.16b. For example, suppose there are two different institutions in the same city: one is a bank, and the other is an e-commerce company. Their user base will likely include most of the area's population; therefore, their shared user space is large.
- **Federated Transfer Learning** applies to scenarios where participating clients differ in both their sample space and feature space, as shown in Figure 2.16c.

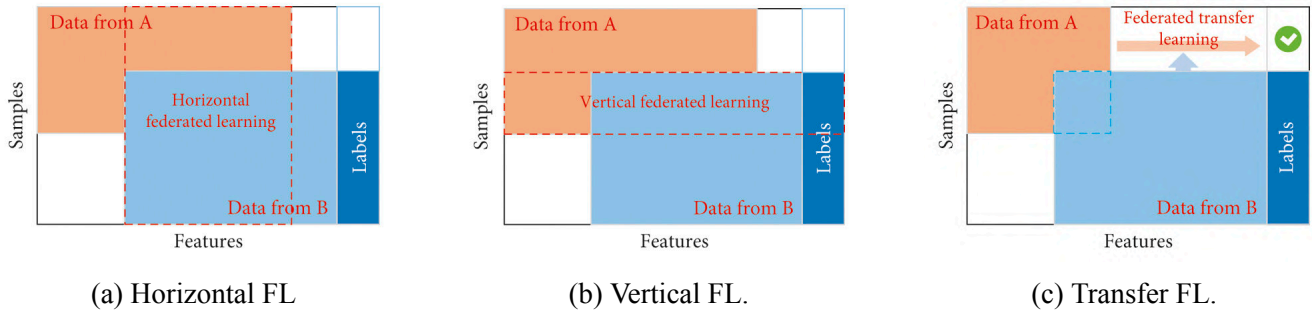


Figure 2.16: Categorization of Federated Learning [45].

b. System Architecture:

Network topology in FL defines the organization of clients and servers and their communication during the training process. It mainly includes centralized and decentralized architectures. as shown in Figure 2.17, each with different communication characteristics and scalability properties [46].

- **Centralized Federated Learning**: The centralized architecture constitutes a prevalent configuration in the majority of FL systems. In this paradigm, a single central node assumes responsibility for communicating with all edge devices, aggregating local models, and deploying the resultant global model. Model transmission within this architecture proceeds seamlessly and elegantly, while the solitary central node incorporates a dedicated system amenable to customization for specific requirements. Expeditionary updates are thereby facilitated, rendering it particularly efficient for small-scale systems, as the central infrastructure demands minimal resources for establishment.
- **Decentralized Federated Learning**: A decentralized federated learning system only contains edge nodes. A decentralized architecture moves the aggregation function to the edge. The system is able to minimize the problem of performance bottlenecks, as the entire computational load is distributed across all nodes.

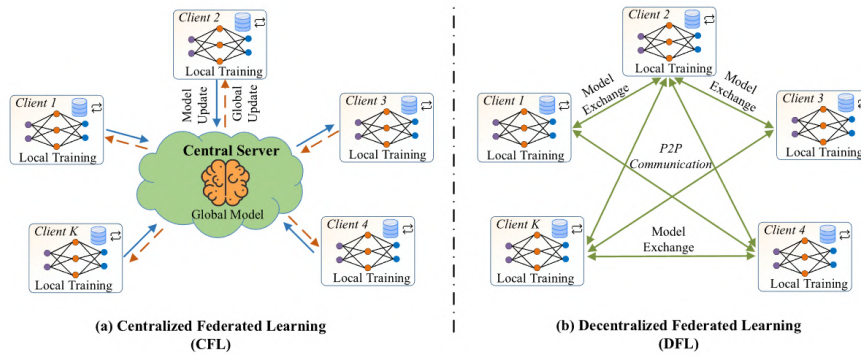


Figure 2.17: Classification of FL based on network structures [47].

c. System Scale:

Federated learning can be classified into two types cross-device and cross-silo, based on the participating clients and the model training scale [48].

- **Cross-Device:** The participating clients are usually edge devices (e.g., smartphones and wearables), and each client is likely to have a relatively small amount of local data. Hence, for cross-device FL to succeed, it usually requires a large number (e.g., up to millions) of edge devices to participate in the training process. The model owner in cross-device FL is the coordinating central server (such as Google).
- **Cross-Silo:** The clients are canonically companies or organizations (e.g., data centers, hospitals, and banks) that own the global model themselves. Clients usually have sufficient computational power and reliable connections (e.g., high-speed wired networks). The number of participants is small (e.g., from two to a hundred), and each can have a relatively large amount of local data, and each client is expected to participate in the entire training process.

2.4.4 The FedAvg Algorithm

McMahan et al. [5] introduced the FedAvg algorithm, which combines local SGD on each client with a central server that aggregates the resulting local models by computing a weighted average based on the number of data samples held by each client. The amount of computation is controlled by three key parameters: C , the fraction of clients that perform computation in each round; E , the number of training passes each client makes over its local dataset in each round; and B , the local minibatch size used for client updates.

2.5 Hierarchical Federated Learning

FL is a paradigm that enables models to be trained collaboratively across distributed clients without breaking privacy. By allowing data to remain on clients' devices and only sharing model updates, FL can preserve privacy while unlocking the potential for collective intelligence for a wide range of applications. Nevertheless, FL encounters several challenges. HFL has emerged as a promising FL extension that introduces multi-level organizational structures to reduce the effect of these challenges.

2.5.1 Definition of HFL

Hierarchical Federated Learning (HFL) is a decentralized machine learning approach that extends the traditional federated learning architecture. It consists of three layers: a cloud server connected to multiple edge servers, which are in turn connected to several clients, each holding its own local datasets, as illustrated in Figure 2.18.

In this architecture, clients perform local training on their private data. Each edge server aggregates the models of its associated clients. Then, the cloud server aggregates the models from all edge servers. The

updated global model is subsequently redistributed to the edge servers and then to the clients [7].

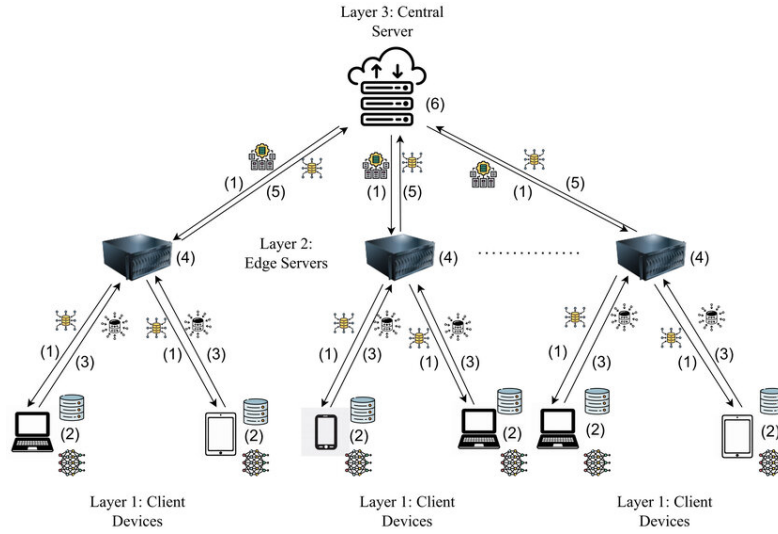


Figure 2.18: HFL Architecture [49].

2.5.2 Motivation for HFL

The motivation behind proposing HFL arises from the problems present in cloud-based FL. Although the number of participating clients may reach millions, providing massive datasets necessary for DL, communication with the cloud server is slow and unstable, for example due to network congestion, which makes the training process inefficient and leads to high communication overhead and long latency. In contrast, the edge server enjoys more efficient communication with clients.

Compared to cloud-based learning, hierarchical learning significantly reduces the communication cost with the cloud, supported by efficient client-to-edge updates, resulting in a reduction in runtime and the number of local iterations, as well as reducing the energy consumption of end devices. On the other hand, since the cloud server can access more data, hierarchical learning outperforms edge-based learning in model training, thus achieving a better balance between computation and communication [7].

2.5.3 The Three-Tier Architecture

The HFL architecture is divided into three levels: IoT/edge, edge, and cloud [50].

- **IoT and Edge Layer:** The lowest layer of HFL, comprising devices such as smartphones, tablets, Internet of Things (IoT) devices, and others. Each client owns its own private data.
- **Edge Layer:** The middle layer of HFL, responsible for aggregating updates from the first and last layers and also reducing the load on the server.
- **Cloud Layer:** The final layer of HFL, a central server that receives all updates sent from the edge servers.

2.5.4 HFL Workflow

The workflow of HFL is inspired by the FedAvg algorithm and extended into a hierarchical version (HierAvg), incorporating an additional aggregation layer. The training process is carried out through the following six stages [7]:

1. **Global model distribution:** The central server sends the global model parameters to the edge servers, which in turn forward them to their associated clients.
2. **Local model training:** Each client receives the model sent by the edge server and trains it locally using its own dataset.
3. **Local model upload:** After training, the clients send the updated model parameters to their respective edge server.
4. **Edge-level aggregation :** Each edge server aggregates the local models received from its clients.
5. **Edge model upload:** The aggregated models from each edge server are sent to the central cloud server.
6. **Cloud-level aggregation:** The central server aggregates the models received from all edge servers to update the global model. It then redistributes the updated parameters to initiate a new training round. These steps are repeated until a predefined number of communication rounds.

2.6 Applications of Federated and Hierarchical Federated Learning

Federated learning systems play an important role in various fields. They are currently applied in a wide range of domains, as illustrated in the following subsections:

- **Healthcare:** In healthcare, FL is used to allow medical institutions to train models independently using their local data, ensuring patient privacy [51]. The combination of FL and the medical industry can build disease detection models with good performance without compromising the private information of patients, which promotes the intelligent transformation of the medical industry [52].
- **Smart cities and homes:** FL offers improvements in smart city services, such as traffic management and energy efficiency, while maintaining data privacy. In smart homes, FL also provides comfort, energy efficiency, and security [53].
- **Finance:** In finance, FL preserves the privacy and security of customers' data while enabling several critical applications, including credit risk management, stock market prediction, and fraudulent transaction detection [54].

- Recommendation systems: FL introduces federated recommendation systems (FedRec), a privacy-preserving approach to recommender systems. It enables personalized recommendations while keeping user data decentralized and protected [55].
- Internet of Things (IoT): Modern IoT networks contain numerous sensors that enable them to collect, react to, and adapt to incoming data in real time. Building aggregated models in these scenarios can be challenging due to the private nature of the data and the limited connectivity of each device. Federated learning methods can help train models that efficiently adapt to changes in these systems while preserving user privacy [56].

2.7 Fundamental Challenges in FL and HFL

Despite the advantages of federated learning, particularly in preserving data privacy and avoiding the sharing of sensitive information with others, it still faces several challenges, among which are:

2.7.1 Statistical Heterogeneity

In FL, each device trains locally using its own private data. This causes the non-IID data problem, which refers to differences in data distribution across users. Four factors contribute to this phenomenon: attribute skew, label skew, temporal skew, and quantity skew [57].

Non-IID data can significantly affect the performance, efficiency, and reliability of FL learning, and these consequences include the following [58]:

- Model bias: The model may favor the data distribution of participants with more prominent datasets.
- Slow convergence: Non-IID data may lead to increased training time, as the global model struggles to reconcile disparate local updates.
- Reduced model accuracy: The global model may perform poorly on specific subsets of underrepresented data.
- Training instability: Non-IID data may cause fluctuations in model performance across training rounds, making it difficult to achieve consistent improvements.
- Communication inefficiency: Reaching convergence may require more communication rounds, increasing network overhead.
- Privacy risks: In some cases, non-IID data may facilitate the inference of information about individual participants' datasets through model updates.

2.7.2 Systems Heterogeneity

System heterogeneity refers to the differences in devices used by clients, their software platforms, communication protocols, and operating environments. These devices differ in processing power, RAM, and

network bandwidth (3G, 4G, 5G, Wi-Fi).

In a federated learning system, the server waits for clients to submit their updated parameters before initiating the aggregation process. Due to the variation in clients' devices, their training efficiency differs, as training time depends on their computational power. This means that high-powered clients and the server may have to wait for slower clients to complete training, leading to wasted resources [59].

2.7.3 Communication Overhead and Efficiency

Communication overhead is a major bottleneck in federated learning, as the communication cost is often significantly higher than the computation cost when numerous edge devices transmit their model parameters to a central server. This high communication cost during the training process leads to reduced training efficiency [52].

2.7.4 Vulnerabilities in Federated Learning

The main vulnerabilities within FL can be classified into three categories: client-side, server-side, and network-level vulnerabilities.

Client-Side Vulnerabilities. Launched on the client portion of the FL process and can be subdivided based on the focus of the attack: training data, local models and the FL channel [60].

Server-Side Vulnerabilities. They target the central server, particularly focusing on the aggregation process and global model in the FL framework. These attacks can be further subdivided based on the attack target: aggregated parameters/gradients and global model [60].

Network-Level Vulnerabilities. The network represents another point of vulnerability that an attacker can exploit to influence the global model learned by the system. The adversary can observe communications sent over the network and prevent the FL algorithm from receiving the information needed to learn a good model [61].

2.8 Conclusion

In this chapter, we presented the fundamentals of ML and DL and discussed the limitations of centralized learning as well as the necessity of adopting decentralized approaches. We also introduced FL and its main concepts, in addition to the multi-level architecture of HFL. Furthermore, we highlighted the key challenges faced by these paradigms, including security vulnerabilities. Finally, we discussed their main application use cases.

Chapter 3

Federated Learning Threats and State-of-the-Art Countermeasures

3.1 Introduction

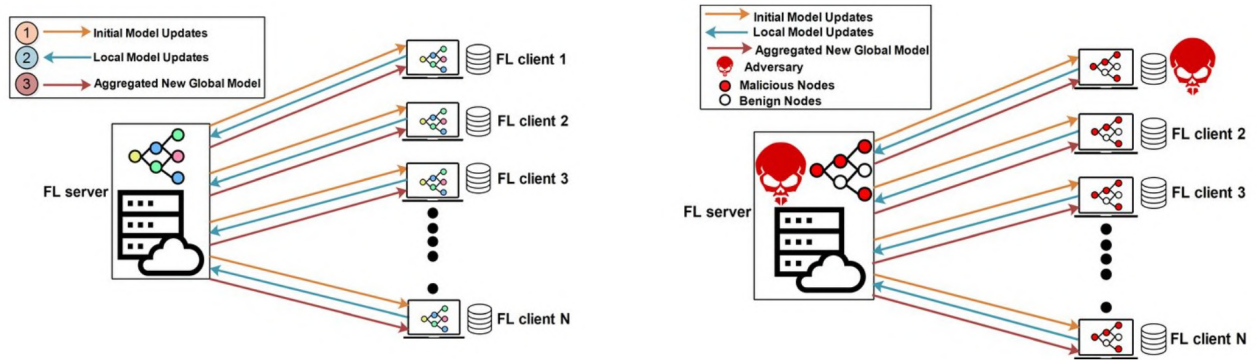
Federated Learning (FL) allows multiple clients to collaboratively train a shared model without exchanging their raw data, making it suitable for distributed systems that are privacy sensitive. However, FL is vulnerable to many adversarial threats, especially when data are non-IID or the system uses multiple hierarchical layers. To address these challenges, researchers have proposed many aggregation mechanisms.

In this chapter we present an overview of the main attack vectors targeting FL systems and we examine multiple state of the art aggregation mechanisms as well as the most used ones that are designed to mitigate these attacks. The objective is not only to review existing contributions, but also to identify their limitations and highlight the research gap that motivated our proposed approach.

This chapter is organized as follows: Section 3.2 provides an overview of the main threats in federated learning, while Section 3.3 reviews the state-of-the-art aggregation mechanisms, categorizing them into different approaches.

3.2 Adversarial Threats in Federated and Hierarchical Federated Learning (HFL)

HFL involves aggregation across multiple layers such as edge, fog, and cloud which increases the system's complexity and expands the attack surface. Understanding these threats is essential in order to design robust aggregation mechanisms that can maintain the model's integrity and reliability even in adversarial environments because each type of attack targets different aspects of the learning process.



(a) Standard Federated Learning setup with benign clients. (b) Federated Learning under adversarial attack with malicious nodes.

Figure 3.1: Federated Learning: normal operation vs. adversarial scenario [62].

Figure 3.1 illustrates the impact of attacks in federated learning by comparing two scenarios: with and without malicious clients. In the benign case (Figure 3.1a), all participants contribute reliable updates, allowing the global model to converge smoothly toward high accuracy. In contrast, when attacks are present (Figure 3.1b), some clients send corrupted or misleading updates, which disrupt the aggregation process and degrade the overall model performance.

3.2.1 Poisoning Attacks

Poisoning attacks seek to compromise the integrity of the global model by injecting malicious data or manipulated updates into the training process. These attacks can take different forms depending on the attacker's goal, typically categorized as targeted or untargeted.

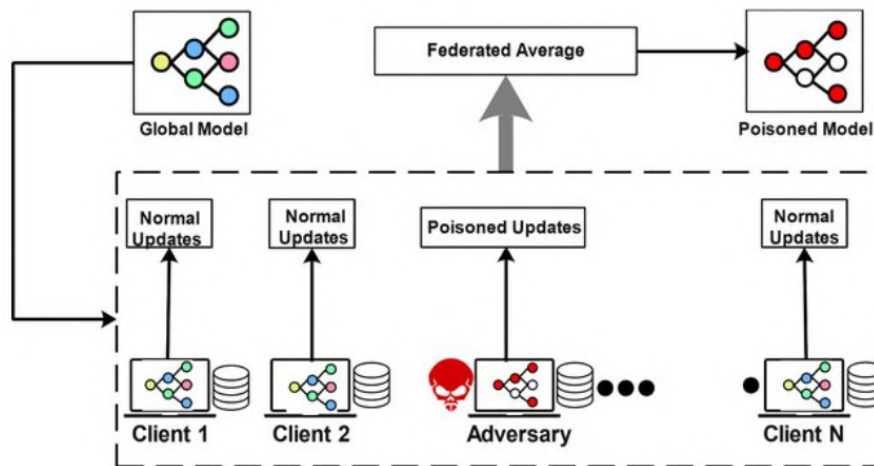


Figure 3.2: Poisoning attack in a federated learning environment[62].

Figure 3.2 illustrates a poisoning attack in an FL environment; adversaries inject malicious updates that cause the poisoning of the global model after aggregation.

a. Targeted Attacks

The attacker aims to make the model behave incorrectly on specific inputs while leaving overall performance mostly unaffected to avoid detection. Targeted attacks can take several forms, including data poisoning, model poisoning, and backdoor attacks [63]. Tolpegin et al. [63] investigated the vulnerability of FL systems to targeted data poisoning attacks, specifically label flipping attacks, where malicious participants change the labels of their local training data from a source class to a target class. Through experiments on CIFAR-10 and Fashion-MNIST, they demonstrated that even a small percentage of malicious participants can significantly degrade global model accuracy and source class recall, while leaving non-target classes largely unaffected which makes the attacks stealthy.

a.1 Data Poisoning Data Poisoning attacks occur when an adversary manipulates the training data by injecting carefully crafted malicious samples. They aim to subtly corrupt the global model or influence its behavior without being detected.

a.1.1 Label Flipping Attacks Adversaries deliberately misclassify training instances by reassigning them to incorrect categories, for example, flipping “spam” emails to “ham”, as illustrated in Figure 3.3. Although this attack is easy to implement, it has a very negative impact on the training process, especially when the number of attackers increases [64]. Lavaur et al. [65] recently highlighted how label flipping attacks can compromise the reliability of Federated Intrusion Detection Systems (FIDS) [66].

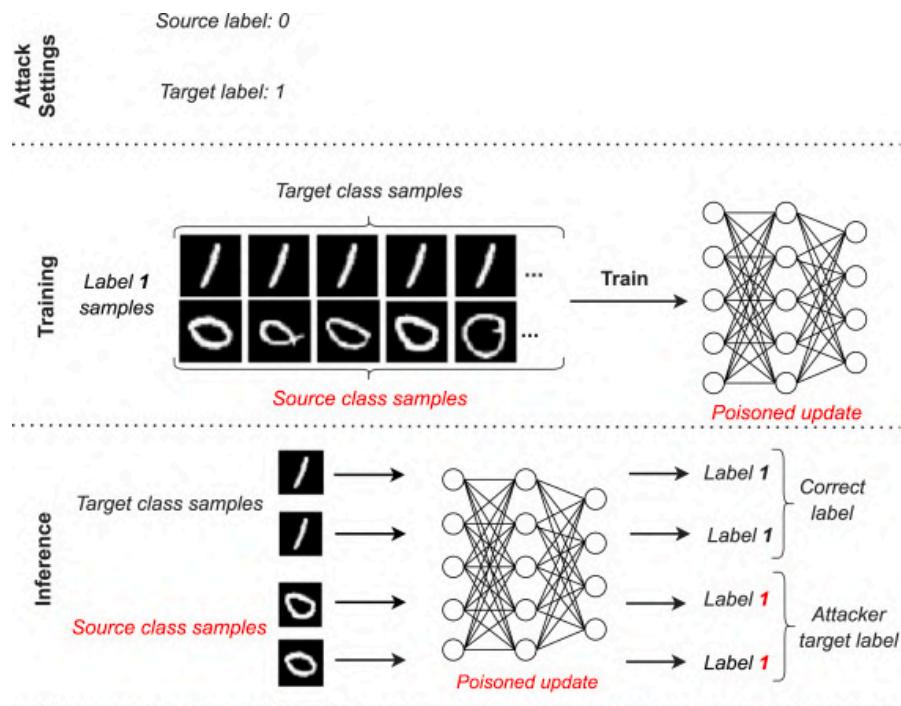


Figure 3.3: Label Flipping Attack

a.2 Backdoor Attacks Backdoor attacks are dangerous and difficult to detect early. The attacker corrupts either a single input such as forcing the model to systematically misidentify a particular individual or a small subset of inputs sharing a common trigger, such as a specific pixel pattern embedded in images. What makes this attack very dangerous is that during training, the model continues to perform normally on clean data, showing no signs of compromise. It is only at inference time, when the specific trigger is present, that the embedded backdoor activates and the model behaves maliciously [67].

Biggio et al. [68] demonstrated that an adversary with knowledge of the training process can craft and inject malicious data points that systematically push the model toward higher classification error. Their work exposed a fundamental vulnerability shared by most learning algorithms: the assumption that training data comes from a clean, uncompromised distribution. This is a critical point in federated learning because the server has no direct visibility into how each client’s data is collected or prepared.

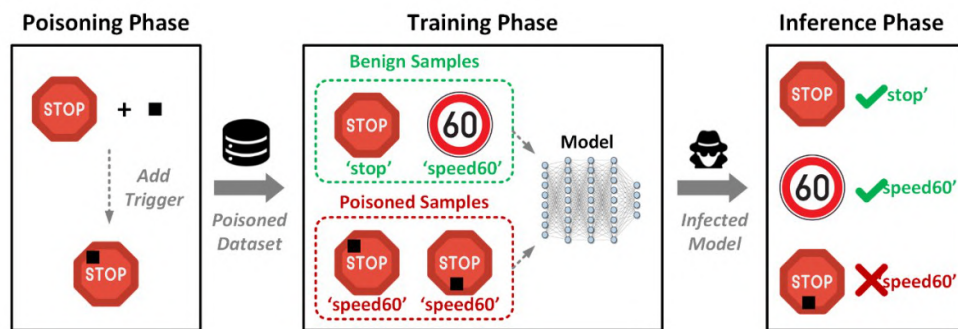


Figure 3.4: Procedure of typical backdoor attacks on AI models [69].

As shown in Figure 3.4, a backdoor attack usually happens in three steps. First, during the poisoning phase, the attacker adds modified data with a hidden trigger into the training set. Then, in the training phase, the model learns to link this trigger to a specific wrong prediction. Finally, during inference, whenever the trigger appears in an input, the model produces the attacker’s desired output instead of the correct one.

a.3 Model Poisoning An attacker deliberately alters the gradients or parameter updates sent to the server, causing the global model to behave incorrectly or to include hidden backdoors.

a.3.1 Attack on Model Replacement In this attack, one malicious client or a group of colluding participants work together to exploit model replacement to add hidden backdoor triggers to the shared global model. This allows them to control how the model acts when it receives certain inputs—for example, manipulating a language model to suggest specific word completions. By scaling their updates, these attackers ensure that their adversarial goals are successfully integrated into the aggregated model during the federated training process [70].

b. Untargeted Attacks

The attacker tries to degrade the model's overall performance, causing general misbehavior across many inputs.

b.1 Gradient Manipulation: Gradients indicate the direction and magnitude of model parameter updates during training. Attackers exploit this reliance by corrupting these signals to compromise the global model:

- **Gradient Scaling:** Attackers scale up their gradient values to amplify the influence of their poisoned updates on the global model [71][66].
- **Sign Flipping Attacks:** Attackers invert the signs of gradient values, instead of contributing to convergence, the attacker forces the model to move in the opposite direction of the optimal solution. This active sabotage is designed to undo the progress made by legitimate users and increase the model's test error [72, 73].
- **Gaussian Noise Injection:** Attackers inject random Gaussian noise into their updates to introduce high variance and entropy into the aggregation to degrade the model's performance [74, 73].

b.2 Byzantine Attacks: The term Byzantine originates from the Byzantine Generals' Problem, introduced by Leslie Lamport, Robert Shostak, and Marshall Pease in 1982. As illustrated in Figure 3.5 [75], the problem describes a scenario in which multiple generals must coordinate a common decision (either attack or retreat) by communicating over unreliable channels. Some of these generals may be traitors who send conflicting or misleading messages to different participants. As a result, loyal generals must reach agreement despite the presence of arbitrary and potentially deceptive behavior [76].

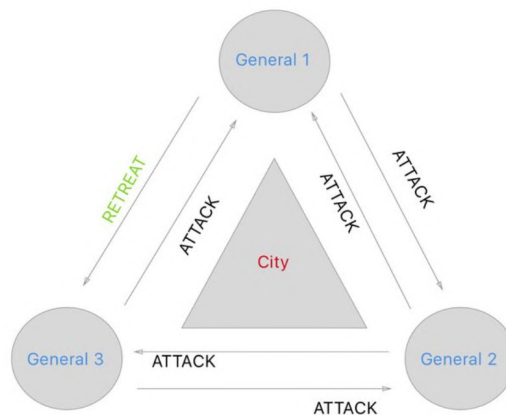


Figure 3.5: Byzantine Generals' Problem.

A Byzantine attack refers to nodes in a distributed system behaving maliciously with the purpose of

disrupting the learning process and preventing the global model from converging toward the optimal solution [77].

Blanchard et al.[11] demonstrated that even a single malicious participant can significantly disrupt the training process, leading to convergence failure and performance degradation. This vulnerability has motivated the development of many defense mechanisms to improve the robustness of federated learning against Byzantine attacks.

What makes Byzantine attacks different from other attacks is their unpredictability. Instead of following a clear goal or strategy, malicious nodes can behave in completely arbitrary ways by sending inconsistent or misleading information at any time. This patternless behavior makes them much harder to anticipate and defend against compared to other structured attacks.

Shi et al.[77] introduced a new Byzantine attack called the *weight attack*, which targets the aggregation step in federated learning. Instead of directly manipulating data or model updates, they exploit how the server assigns weights to client contributions. Malicious clients falsely report the size of their local datasets to gain more influence during aggregation. Since the server cannot verify these claims due to privacy constraints, attackers can easily amplify their impact on the global model.

b.2.1 A Little Is Enough Attack (LIE) LIE's core idea is to craft each malicious update so it appears normal to robust aggregators, while coordinating them to produce a consistent adversarial shift in the global model. It operates by having compromised clients submit small, well-aligned gradient updates that remain close to benign ones in terms of distances or medians, but their aggregate drifts the model in the adversary's direction [78, 79].

3.2.2 Inference Attacks

Inference attacks in federated learning aim to extract sensitive information about the training data from shared model updates or the global model itself, as illustrated in Figure 3.6, despite the absence of direct access to raw data, without disrupting the model's normal training process [80].

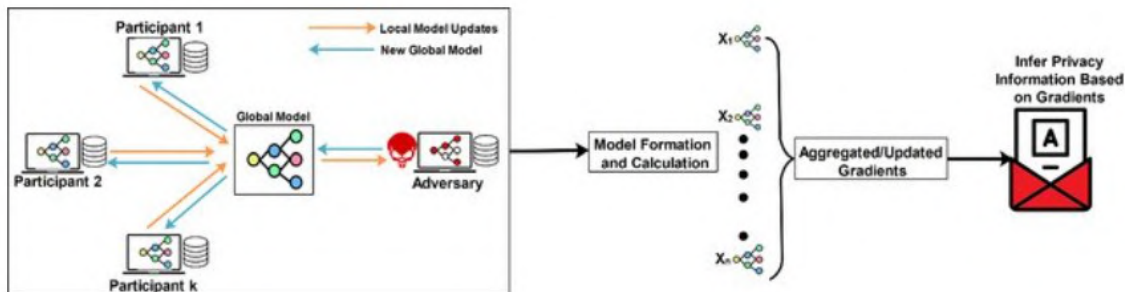


Figure 3.6: Inference attacks in the federated learning environment

a. Model Inversion Attackers exploit their access to trained models via reverse engineering to reconstruct private training data from other clients, helping them reveal private information without direct access to the raw data [81].

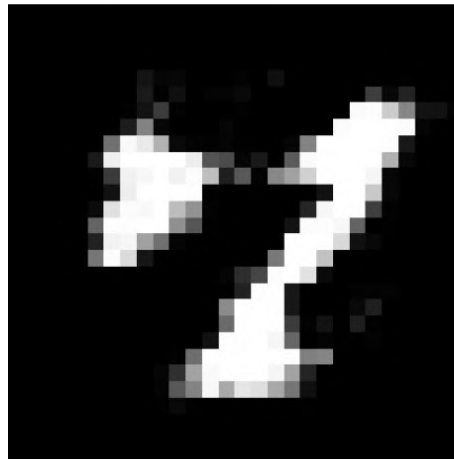


Figure 3.7: Reconstructed Image (Target 7).

Figure 3.7 shows a reconstructed image from the MNIST dataset; although it is slightly noisy, it is recognizable as the handwritten digit "7". The reconstruction achieved a confidence of 0.5391 and a loss of 0.6232. These metrics indicate that the inversion process successfully extracted meaningful visual patterns from the global model. This example demonstrates that MNIST, with its simple and structured patterns, is relatively vulnerable to model inversion.

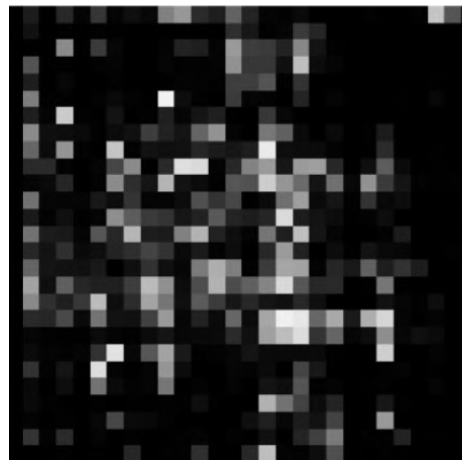


Figure 3.8: Reconstructed Image (Target 0).

Figure 3.8 shows a reconstructed image from class 0 (T-shirt) from the FashionMNIST dataset, which is highly noisy and barely recognizable. The reconstruction achieved a confidence of 0.0210 and a loss of 3.8681, meaning the model failed to recover discriminative features for this class. The higher complexity and variability of FashionMNIST make model inversion much more challenging. The effectiveness of model inversion is dataset-dependent.

3.2.3 Free-rider Attacks

Free rider attack is a new special attack method where malicious clients exploit the fact that they have direct control over their local models or training data to submit fake parameters either because they lack training data or because they want to save local storage space and computation resources, as illustrated in Figure 3.9 [82].

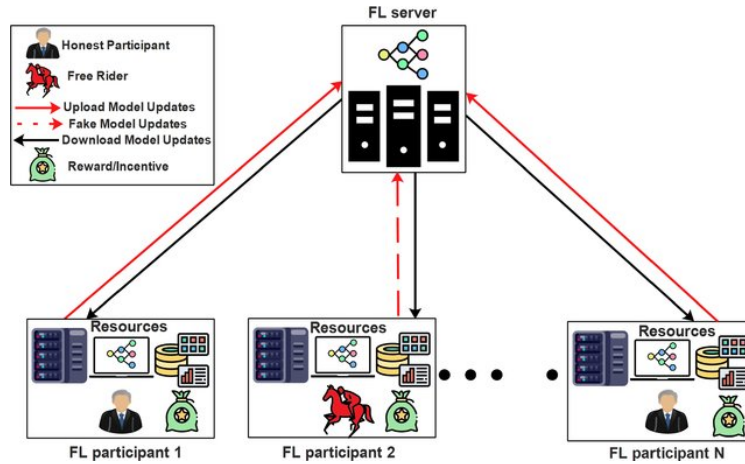


Figure 3.9: Free-riding attacks in the federated learning environment.

[Zhu et al. \[82\]](#) proposed an advanced free-rider attack that uses the history of global models to calculate a scaled delta to ensure convergence. They exploit statistical properties such as the Ornstein-Uhlenbeck process and geometrical relationships (cosine similarity and the L2-norm) to construct a fake update similar to real updates, maintaining the stealth property.

3.2.4 Sybil Attacks

Sybil attacks occur when an adversary controls multiple fake or compromised clients to gain disproportionate influence over the aggregation process. By submitting coordinated malicious updates from these identities, the attacker can bias the global model or even destabilize the training process [83].

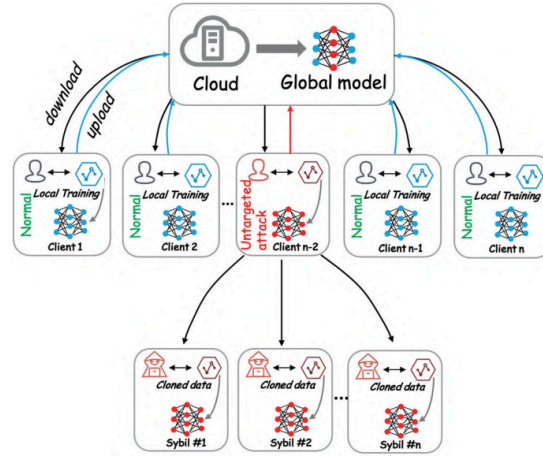


Figure 3.10: Sybil-Based Untargeted Data Poisoning Attack.

Figure 3.10 [84] depicts a Sybil-based untargeted data poisoning attack, where multiple seemingly independent clients, controlled by a single adversary, participate simultaneously in the training process. These clients appear legitimate but collectively introduce subtle perturbations in their updates, leading to a gradual degradation of the global model without targeting a specific outcome.

3.3 State of the Art Countermeasures

After having an overview about the main threats in FL, we explore the classical robust aggregation mechanisms that have been widely adopted to defend against adversarial FL threats.

3.3.1 Statistical-Based Defenses

Blanchard et al. [11] demonstrated that traditional aggregation rules based on linear combinations cannot handle even one malicious client who can manipulate the aggregated update and prevent convergence. To address this limitation, the authors propose *Krum*, a non-linear gradient aggregation rule that selects the update whose distance to its closest $n - f - 2$ neighbors is minimal, where n represents the total number of clients and f is the assumed number of Byzantine ones. Under the condition $2f + 2 < n$, Krum is proven to satisfy a formal Byzantine resilience property, ensuring the convergence of SGD even in the presence of up to f faulty workers. The method has a computational complexity of $O(n^2 \cdot d)$, where d denotes the dimension of the model parameters. The authors also introduce *Multi-Krum*, a variant that averages the m best-scoring vectors instead of selecting a single one. Despite their robustness to Byzantine gradients, selecting only a small subset of gradients may discard useful information from benign clients, which can slow down convergence. Moreover, their effectiveness can degrade in highly heterogeneous environments where client updates naturally differ due to non-IID data distributions.

Unlike simple averaging, which is highly sensitive to extreme or adversarial updates, the median selects the central value among client updates for each model parameter, effectively ignoring unusually large or small contributions which makes it robust against certain types of Byzantine attacks, such as gradi-

ent scaling or random noise injection [85]. However, the classical median treats all clients equally and does not consider their relative reliability or contribution quality, which can limit its effectiveness in heterogeneous and non-IID environments. In this context, Kabbaj et al. [85] proposed a robust aggregation method for federated learning called Weighted Median Aggregation (WMA). In this approach they assign weights to clients based on reliability or contribution. For each model parameter they sort client values then compute a weighted median instead of a mean, this helps limit the influence of malicious update. However, it requires sorting per parameter, the performance depends on quality of assigned weights, and still assumes a bounded number of attackers.

Wang et al. [86] proposed an aggregation method designed to mitigate the impact of malicious or outlier client updates in federated learning where instead of directly averaging all client updates, the server performs a sorting of model updates and removes a fixed proportion of the largest and smallest values before computing the mean of the rest updates. Their aggregator reduces the impact of adversarial clients such as in scaling attacks. This approach is particularly effective when the number of malicious clients is bounded and known in advance. However, its effectiveness depends on the correct choice of the trimming parameter, as overly aggressive trimming may discard useful information from honest clients, especially in non-IID settings where legitimate updates can naturally exhibit high variability. Additionally, since the method operates independently on each parameter dimension, it may fail to capture complex structured attacks that preserve coordinate-wise statistics while still manipulating the global model.

3.3.2 Accuracy Based Defenses

Xie et al. [87] proposed an aggregation rule designed for DSGD under adversarial settings. Unlike the majority of defense mechanisms that assume the number of malicious clients is bounded by a fraction of the total participants, Zeno operates under the assumption of requiring only the presence of at least one honest worker. The method evaluates each submitted gradient through a suspicion scoring mechanism called the *stochastic descendant score*, which estimates how much the candidate update would decrease the loss function while penalizing large update magnitudes. Gradients are ranked according to this score, and the server aggregates the top $m - b$ candidates, where m denotes the number of workers and b represents the number of filtered gradients. This allows the algorithm to suppress harmful updates before aggregation. The authors demonstrated that Zeno maintains stable convergence under several attack scenarios through experiments on benchmark datasets. However, the approach introduces additional computational overhead due to the evaluation of the stochastic descendant score, which requires extra forward passes to estimate the loss variation. Moreover, the performance of Zeno depends on hyperparameters such as the trimming parameter b , the penalty weight ρ , and the auxiliary batch size used for score estimation, which may require careful tuning in practice.

3.3.3 Hybrid Defenses

Savoia et al. [88] propose SAGE, a multi-objective client selection framework that considers residual energy, data diversity, and the use of renewable energy at the same time. They proved that it meets a sustainability condition as long as at least 20% of clients keep a normalized residual energy above 0.2,

which also ensures convergence of the global model even under highly non-IID data across up to 50 heterogeneous devices. The computational cost of SAGE is $O(N \cdot d)$, where N is the number of clients and d is the size of the data divergence vector. They also proposed an adaptive local training variant that reduces local epochs from 5 to 2 for energy-limited clients instead of removing them completely. However, selecting only a subset of clients using a weighted objective can sometimes exclude useful updates from clients that have moderate energy but highly informative data, which may slow down early convergence. In addition, SAGE may perform less well in extreme non-IID settings ($\beta = 0.1$) when the divergence weight is too low, since the method may favor energy-efficient clients while ignoring rare but important data patterns.

3.3.4 Trust Aware Aggregation Baselines

Cao et al. [89] introduced a Byzantine resilient aggregation method that addresses the limitation of the absence of a reliable reference point for evaluating client updates. Instead of relying solely on the submitted gradients, FLTrust establishes a *root of trust* at the server side by maintaining a small clean dataset, referred to as the *root dataset*. Using this dataset, the server computes its own gradient update, called the *server model update* to use as reference to test the reliability of client updates. In each training round, the server assigns a trust score to every client update based on its directional similarity with the server update, measured through a ReLU-clipped cosine similarity. Updates that align more closely with the trusted direction receive higher scores, while those pointing in opposite directions are ignored. In addition, FLTrust normalizes the magnitude of client updates so that they lie on the same hypersphere as the server update, preventing malicious clients from dominating the aggregation through excessively large gradients. The final global update is then computed as a weighted average of the normalized client updates using their corresponding trust scores. Experimental results show that FLTrust can remain robust even when a large fraction of participating clients are malicious. However, the method relies on the availability of a clean root dataset at the server, which may not always be feasible in practice. Moreover, if the distribution of the root dataset differs significantly from that of the clients' data, the trust estimation may become unreliable, potentially reducing the effectiveness of the defense.

Fung et al. [90] introduced an aggregator that focuses on defending the system against sybil attacks. The main idea behind FoolsGold is that malicious clients controlled by the same attacker tend to produce very similar gradient updates, since they share the same objective. To detect this behavior, the method measures the similarity between client updates using cosine similarity and reduces the influence of clients whose updates are too similar to others. Instead of completely removing suspicious clients, FoolsGold adapts the learning rate of each participant, giving less weight to updates that appear highly correlated. This allows the aggregation process to limit the impact of colluding attackers while still keeping honest participants in the training process. Experimental results show that this approach can effectively mitigate different poisoning strategies, including label flipping and backdoor attacks, across several datasets and models

Song et al. [91] introduced a reputation mechanism that evaluates the reliability of each client over multiple training rounds since the central server cannot observe the local training procedure of each client,

such malicious updates can negatively affect the global model. The reputation score of each participant is computed using a Beta distribution model and is updated based on the impact of its local update on the global model performance. Clients that consistently contribute positively receive higher reputation scores, while those whose updates degrade the model performance experience a decrease in their reputation. Based on these scores, the server applies a reputation-based scheduling strategy that prioritizes more trustworthy clients for participation in the aggregation process, thereby reducing the influence of malicious participants. However, the approach relies on evaluating client updates using a validation dataset at the server side, which may not always be effective due to the non-IID nature of data distributed across clients.

Table 3.1: Comparison of Robust Aggregation and Defense Mechanisms in Federated Learning

Contribution	Category	FL Position	Dataset	Architecture	Strengths	Limitations
Blanchard et al. (Krum, Multi-Krum) [11]	Statistical-Based	Cloud-based	MNIST, CIFAR-10	Flat FL	Proven Byzantine resilience and robust gradient selection	Discards useful updates, may slow convergence, sensitive to heterogeneity
Wang et al. (Trimmed Mean) [86]	Statistical-Based	Cloud-based	MNIST, CIFAR-10	Flat FL	Robust against outliers and scaling attacks	Sensitive to trimming parameter, weak against structured attacks
Xie et al. (Zeno) [87]	Accuracy-Based	Cloud-based	MNIST, CIFAR-10	Flat FL	Requires only one honest client, strong theoretical guarantees	High computational cost, sensitive hyperparameters
Savoia et al. (SAGE) [88]	Hybrid	Edge / Fog	IoT data	Hierarchical	Energy-aware and diversity-aware client selection, scalable	May exclude useful clients, sensitive to weight tuning
Cao et al. (FLTrust) [89]	Trust-Based	Cloud-based	MNIST, CIFAR-10	Flat FL	Strong robustness via trusted reference model	Requires clean root dataset, sensitive to distribution mismatch
Fung et al. (Fools-Gold) [90]	Trust-Based	Cloud-based	MNIST, VGGFace2	Flat FL	Effective against Sybil attacks without prior knowledge	Weak against diverse or adaptive attacks
Song et al. (Reputation-Based FL) [91]	Trust-Based	Edge / Wireless	Simulation-based	Flat FL	Captures long-term behavior and distinguishes malicious clients	Depends on validation assumptions, limited real-world evaluation

3.4 Conclusion

In this chapter we reviewed the main security threats in federated learning and analyzed existing robust aggregation methods designed to mitigate them. While current approaches provide varying levels of resilience against adversarial behavior, they often rely on strong assumptions and face limitations in heterogeneous and non-IID settings. These challenges highlight the need for more adaptive and robust aggregation mechanisms.

Chapter 4

Trust Weighted Aggregation for Hierarchical Federated Learning

4.1 Introduction

In previous chapters, we examined the limitations and challenges of FL and reviewed several state-of-the-art defense mechanisms, which allowed us to identify key research gaps. Traditional aggregation methods assume honest clients, some robust aggregation methods aggressively eliminate suspicious clients, such immediate exclusion policies may unfairly penalize honest participants whose updates appear noisy due to non-IID data distributions or training instability, sophisticated adversaries can mimic honest update patterns making detection challenging. These observations motivated us to design a more adaptive and behavior-aware aggregation strategy. In this chapter, we present our proposed approach along with its mathematical formulation. We also provide illustrative examples to facilitate a clear understanding of how client updates are evaluated, weighted, and aggregated.

This chapter is organized as follows: Section 4.2 introduces the system and the threat model describing the HFL architecture and the roles of each layer as well as the adversary assumptions. Section 4.3 presents the proposed behavioral trust metric detailing the trust components used to evaluate client updates. Section 4.4 presents the mathematical formulation of the trust computation, temporal regulation mechanisms and the trust-based aggregation strategy. Sections 4.5 and 4.6 provide the algorithm with an illustrative example. Finally, Section 4.7 concludes the chapter.

4.2 System and Threat Model

To present TWA-HFL framework, we will first define the HFL architecture in which it operates and characterize the adversarial environment it must withstand. We describe the roles of clients, fogs and the cloud server, the assumptions our system operates under, the capabilities and objectives of potential attackers. These elements describe the groundwork for our trust mechanism and specify the threats which we are trying to defend against.

Our HFL system consists of three layers:

a. Edge Clients

- Benign clients perform honest local training while potential sybil clients behave maliciously.
- Each client holds a private non-IID subset of the dataset and, after each training round, transmits their model updates $\Delta w_i^{(t)}$ to their assigned fog node.

b. Fog Nodes

Fog nodes are intermediary servers responsible for aggregating local updates from edge client clusters and computing an intermediate trust score prior to cloud transmission.

- Each fog node receives updates from a subset of clients.
- Performs first-level trust-weighted aggregation using TWA-HFL.
- Forwards aggregated models to the central cloud server.

c. Cloud Server

- Receives aggregated models from all fog nodes.
- Performs second-level aggregation to update the global model.
- Maintains global model $w_g^{(t)}$ and distributes it to all clients at the beginning of each round.

The proposed trust-aware mechanism operates under the following assumptions:

a. Client Assumptions

- Each client possesses sufficient computational resources to perform local training for the assigned number of epochs.
- Clients hold private local datasets and do not share raw data with other clients, fog nodes, or the cloud server.
- At each communication round, participating clients receive the current global model and compute a local update $\Delta w_i^{(t)}$ using their private data.
- Benign clients follow the prescribed training protocol and submit genuine model updates.
- Malicious clients may behave arbitrarily: they can manipulate their local updates, scale them, invert them, or craft adversarial updates with the goal of disrupting the global model.

- Clients do not have direct access to the internal trust scores assigned by the aggregation mechanism.

b. Data Assumptions

- Data is non-IID across clients. The heterogeneity is simulated using a sharding strategy where each client is assigned two shards of the dataset.
- Local datasets may vary in size, reflecting realistic client data imbalance.
- The training and test datasets are drawn from the same underlying distribution to ensure consistent evaluation of global model performance.

c. System and Attack Assumptions

- A fraction of clients may be malicious, and their identities are unknown to the server.
- Malicious clients may collude and coordinate their updates in order to amplify the impact of the attack.
- Fog nodes and the cloud server are assumed to be honest-but-curious and correctly execute the aggregation and trust computation procedures.
- Communication channels between layers are assumed to be reliable, and transmitted model updates are not altered during transmission.

4.3 The Behavioral Trust Metric

Conventional robust aggregation techniques typically rely on a single evaluation criterion, such as gradient similarity, norm deviation, or distance from the mean update. Although they are effective against basic attacks, they remain vulnerable to more adaptive adversaries. For example, in a gradient scaling attack, a malicious client preserves the direction of its update while manipulating its magnitude, such behavior can evade similarity-based detectors that primarily focus on directional alignment. Conversely, adversarial label-exchange (ALE) attacks may maintain a plausible magnitude while subtly altering the update direction, thereby bypassing magnitude-based filtering strategies.

A more fundamental limitation of these approaches is their stateless nature. Most existing methods evaluate client updates independently at each communication round, without considering historical behavior. As a result, intermittent or strategically alternating attacks may avoid detection by remaining within acceptable bounds in isolated rounds.

To address these limitations, the proposed TWA-HFL framework introduces a dynamic behavioral trust metric. Instead of relying on a single signal, each client is assigned a continuously updated trust score that evolves over time. This score is derived from a multi-dimensional behavioral profile capturing complementary characteristics of the client's update patterns.

By incorporating both instantaneous and historical information, the trust mechanism enables the system to identify persistent malicious behavior, detect instability, and reward consistent honesty. The resulting trust score directly influences the aggregation process, reducing the impact of suspicious updates while preserving contributions from reliable clients.

4.3.1 Components of the Trust Score

Each client i is assigned a dynamic trust value that reflects the reliability of its model updates over time. This trust value is derived from an intermediate score, denoted as $\text{RawTrust}_i^{(t)}$, which captures the client's behavioral characteristics at communication round t .

The raw trust score is constructed from four complementary components. Each component measures a distinct aspect of update behavior, ensuring that trust is not determined by a single criterion.

a. Update Similarity

This component evaluates the directional alignment between the client update $\Delta w_i^{(t)}$ and the reference global update direction.

In collaborative training, honest clients are expected to move the model parameters toward a common optimization objective. Consequently, their updates should exhibit high directional similarity with the aggregated global improvement. Malicious clients, in contrast, often introduce directional deviations in order to poison the model. Measuring similarity therefore allows the system to detect updates that significantly diverge from the collective learning direction.

b. Cluster Alignment

In the hierarchical setting, each fog node aggregates updates from a cluster of clients. Cluster alignment measures the consistency between a client's update and the aggregated behavior of its local cluster.

Honest clients operating under similar data distributions within the same cluster tend to produce updates that are mutually coherent. A client whose update deviates strongly from its cluster peers may indicate abnormal or adversarial behavior. This local consistency check strengthens robustness by incorporating cluster-level behavioral information rather than relying solely on global comparison.

c. Update Magnitude

The magnitude component measures the norm of the client's update, that is the size of the update proposed by the client to the model.

Under standard training conditions, clients start from the same global model and train for the same number of local epochs. As a result, honest updates typically have comparable magnitudes. Adversarial behavior often disrupts this pattern. In gradient scaling attacks, malicious clients amplify their updates to dominate aggregation. In contrast, stealthy attackers may shrink their updates to remain undetected while gradually influencing the model. Monitoring update magnitude helps detect both exaggerated and abnormally small

contributions.

d. Historical Consistency

Historical consistency evaluates the stability of a client's past trust values. Instead of focusing solely on the current update, this component incorporates the history of the client's previously computed trust scores.

An honest client should demonstrate relatively stable behavior across rounds. Repeated fluctuations in trust may indicate intermittent or strategically alternating attacks. By integrating historical trust information, the system reduces the impact of one-time anomalies while identifying persistent instability.

4.3.2 Behavioral Regulation: Trend and Volatility

While $\text{RawTrust}_i^{(t)}$ captures instantaneous behavioral signals, it does not directly account for temporal dynamics. To address this, two additional mechanisms are introduced: trend and volatility.

Importantly, these mechanisms operate on the history of RawTrust values rather than the smoothed Trust score. Using raw values allows faster reaction to malicious behavior, as smoothing could otherwise delay detection.

Trend measures the direction of change in RawTrust over a sliding window. A consistently increasing trend indicates improving behavior and leads to trust reinforcement. Conversely, a decreasing trend signals deteriorating behavior and accelerates penalization.

Volatility measures the variance of recent RawTrust values. High volatility reflects unstable or oscillating behavior, which is characteristic of intermittent attacks. Low volatility indicates stable participation.

These two mechanisms are integrated into the final Trust update rule. A positive trend combined with low volatility produces a recovery effect, rewarding sustained honest behavior. A negative trend or high volatility strengthens decay, reducing the influence of suspicious clients.

Through this dual regulation process, the final Trust score becomes both history-aware and responsive, enabling rapid reaction to adversarial deviations while preserving stability for benign clients.

4.4 Formulation of the Proposed Approach

We present the mathematical formulation of the proposed framework. We first define the notation and representation of client updates, then formalize how behavioral indicators are computed and combined. Finally, we describe how these quantities are integrated into the trust-aware aggregation mechanism.

We consider a set of clients participating at communication round t : $\mathcal{S}_t = \{1, \dots, n_t\}$. Let $w^{(t)}$ be the current global model and $w_i^{(t)}$ the local model obtained by client i after local training.

Client update (delta). In this work, each client sends an *update* defined as:

$$\Delta w_i^{(t)} = w_i^{(t)} - w^{(t)}. \quad (4.1)$$

Global update history. The global update direction at round t is defined as :

$$\Delta w_g^{(t)} = w^{(t)} - w^{(t-1)} \quad (4.2)$$

Let:

$$u_i^{(t)} = \text{vec}(\Delta w_i^{(t)}) \in \mathbb{R}^d, \quad (4.3)$$

where $u_i^{(t)}$ is the flattened vector representation of client i 's update at round t , and d is the total number of parameters in the model.

Similarly, let:

$$\Delta u_g^{(t)} = \text{vec}(\Delta w_g^{(t)}) \in \mathbb{R}^d, \quad (4.4)$$

where $\Delta u_g^{(t)}$ is the flattened vector representation of the global update at round t , defined as the difference between consecutive global model states:

$$\Delta u_g^{(t)} = \text{vec}(w^{(t)}) - \text{vec}(w^{(t-1)}).$$

The global update at round t is computed as the trust-weighted aggregation of client updates:

$$u_g^{(t)} = \sum_{j \in \mathcal{S}_t} \epsilon_j^{(t)} u_j^{(t)}. \quad (4.5)$$

where $\epsilon_j^{(t)}$ is the aggregation weight assigned to client j .

a. Trust-Based Aggregation Weights

The goal is to determine how much influence each client should have when updating the final model. We assign each client a continuous weight between 0 and 1 based on their trustworthiness, then use these weights to compute a weighted average of client updates.

Client aggregation weights are computed using the following formula:

$$\hat{\epsilon}_i = \frac{\exp(\text{Trust}_i - m)}{\sum_{j=1}^n \exp(\text{Trust}_j - m)} \quad (4.6)$$

where m represents the maximum trust score among all n clients in the cluster, and n is the set of clients in the chosen trusted cluster $\mathcal{C}^*$:

$$m = \max_{j=1}^n \text{Trust}_j$$

Trust_i is the final trust score of client i .

To avoid completely excluding a client from aggregation, a minimum weight threshold is applied:

$$\hat{\epsilon}_i = \max(\hat{\epsilon}_i, 0.01) \quad (4.7)$$

Weights are then normalized so that $\sum_{j=1}^n \epsilon_j = 1$:

$$\epsilon_i = \frac{\hat{\epsilon}_i}{\sum_{j=1}^n \hat{\epsilon}_j} \quad (4.8)$$

Softmax transforms trust scores into a normalized probability distribution, ensuring that aggregation weights remain positive and sum to one while emphasizing relative differences in reliability among clients.

Now we can calculate the weighted average which will be the value of the first-level aggregation performed by the fog server.

To determine Trust_i we first need to compute RawTrust_i . We compute four normalized indicators in $[0, 1]$:

$$\begin{aligned} S_i^{(t)} &\in [0, 1] \text{ (global-direction similarity)}, & C_i^{(t)} &\in [0, 1] \text{ (cluster alignment)}, \\ M_i^{(t)} &\in [0, 1] \text{ (magnitude consistency)}, & H_i^{(t)} &\in [0, 1] \text{ (historical consistency)}. \end{aligned}$$

Example 1. Assume five clients with trust scores at round t :

$$\text{Trust}_1 = 0.95, \quad \text{Trust}_2 = 0.80, \quad \text{Trust}_3 = 0.60, \quad \text{Trust}_4 = 0.40, \quad \text{Trust}_5 = 0.10.$$

Step 1. Find the maximum trust score $m = \max_j \text{Trust}_j = 0.95$, then evaluate the raw softmax weights:

$$\hat{\epsilon}_i = \frac{\exp(\text{Trust}_i - 0.95)}{\sum_{j=1}^5 \exp(\text{Trust}_j - 0.95)}.$$

$$\hat{\epsilon}_1 = e^0 = 1.000, \quad \hat{\epsilon}_2 = e^{-0.15} \approx 0.861, \quad \hat{\epsilon}_3 = e^{-0.35} \approx 0.705, \quad \hat{\epsilon}_4 = e^{-0.55} \approx 0.577, \quad \hat{\epsilon}_5 = e^{-0.85} \approx 0.427.$$

Step 2. Normalize so that $\sum_{i=1}^5 \epsilon_i = 1$, with $\sum_j \hat{\epsilon}_j = 1.000 + 0.861 + 0.705 + 0.577 + 0.427 = 3.570$:

Client	$\hat{\epsilon}_i$	$\epsilon_i = \hat{\epsilon}_i / 3.570$	Final weight
C_1	1.000	1.000/3.570	28.0%
C_2	0.861	0.861/3.570	24.1%
C_3	0.705	0.705/3.570	19.7%
C_4	0.577	0.577/3.570	16.2%
C_5	0.427	0.427/3.570	12.0%

Table 4.1: Aggregation weights derived from trust scores.

Step 3. Apply the minimum threshold $\hat{\epsilon}_i \leftarrow \max(\hat{\epsilon}_i, 0.01)$. All values already exceed 0.01, so no adjustment is needed. Additionally, no second normalization is needed.

The most-trusted client C_1 receives 28% of the aggregation influence, while C_5 , whose trust score is nine times lower, receives only 12%.

b. Global Direction Similarity

Cosine similarity between two vectors $a, b \in \mathbb{R}^d$ is defined as:

$$\cos(a, b) = \frac{a^\top b}{\|a\|_2 \|b\|_2}$$

where $a^\top b$ denotes the dot product between a and b , and $\|a\|_2$ denotes the Euclidean (ℓ_2) norm of a , defined as:

$$\|a\|_2 = \sqrt{a^\top a}.$$

To measure alignment, we compute the cosine similarity between the client update and recent global update **deltas**, that is, the changes in the global model between consecutive rounds:

$$\cos(u_i^{(t)}, \Delta u_g^{(t-k)}) = \frac{u_i^{(t)} \cdot \Delta u_g^{(t-k)}}{\|u_i^{(t)}\| \|\Delta u_g^{(t-k)}\|}$$

To reduce sensitivity to stochastic fluctuations, we average similarity over the K most recent global deltas:

$$\bar{S}_i^{(t)} = \frac{1}{K} \sum_{k=1}^K \cos(u_i^{(t)}, \Delta u_g^{(t-k)}),$$

where $K = \min(3, t - 1)$, and 3 is a hyperparameter representing the number of recent global rounds used in the averaging process. The selection of $K = 3$ for the temporal averaging window provides a balance between noise reduction and responsiveness. Using a single round measurement ($K = 1$) can be affected by temporary fluctuations caused by mini-batch sampling, non-IID data distributions, and partial client participation, which may lead to incorrect trust evaluations. However, a very large window reduces the ability of $S_i^{(t)}$ to adapt to real changes in the global optimization direction. Therefore, a three round window is sufficient to smooth short-term noise while preserving the ability to follow the current convergence trend.

Since cosine similarity lies in $[-1, 1]$, we normalize it into $[0, 1]$ to obtain a bounded behavioral indicator:

$$S_i^{(t)} = \frac{1 + \bar{S}_i^{(t)}}{2}. \quad (4.9)$$

Thus, $S_i^{(t)} \in [0, 1]$, where:

- $S_i^{(t)} \approx 1$ indicates strong alignment with recent global optimization,
- $S_i^{(t)} \approx 0.5$ indicates weak or neutral alignment,
- $S_i^{(t)} \approx 0$ indicates strong opposition to global progress.

At $t = 1$, or when no global update history is available, we initialize $S_i^{(t)} = 1$ for all clients.

Example 2. Consider five clients at round $t = 2$, with the previous global delta $\Delta u_g^{(1)} = [0.5, 0.75]$ and $\|\Delta u_g^{(1)}\|_2 = 0.901$. At $t = 1$, no global history is available; thus, $S_i^{(1)} = 1$ for all clients. The L2 norm of each client update is computed as:

$$\|u_i^{(t)}\|_2 = \sqrt{\sum_{k=1}^d (u_{i,k}^{(t)})^2}$$

At $t = 2$, the client updates are given as follows:

Client	$\mathbf{u}_i^{(2)}$	$\ \mathbf{u}_i^{(2)}\ _2$
C1	[0.60, 0.80]	1.000
C2	[0.50, 0.70]	0.860
C3	[-0.20, 0.10]	0.224
C4	[0.40, 0.90]	0.985
C5	[-0.50, -0.40]	0.640

Table 4.2: Client updates and their L2 norms at round $t = 2$.

Since $K = \min(3, t - 1) = 1$, the temporal averaging reduces to a single cosine similarity evaluation. The similarity for each client is computed as follows:

$$\begin{aligned} \cos(\mathbf{u}_1^{(2)}, \Delta u_g^{(1)}) &= \frac{0.6 \times 0.5 + 0.8 \times 0.75}{1.000 \times 0.901} = \frac{0.900}{0.901} = 0.999, \\ \cos(\mathbf{u}_2^{(2)}, \Delta u_g^{(1)}) &= \frac{0.5 \times 0.5 + 0.7 \times 0.75}{0.860 \times 0.901} = \frac{0.775}{0.775} \approx 1.000, \\ \cos(\mathbf{u}_3^{(2)}, \Delta u_g^{(1)}) &= \frac{(-0.2)(0.5) + (0.1)(0.75)}{0.224 \times 0.901} = \frac{-0.025}{0.202} = -0.124, \\ \cos(\mathbf{u}_4^{(2)}, \Delta u_g^{(1)}) &= \frac{(0.4)(0.5) + (0.9)(0.75)}{0.985 \times 0.901} = \frac{0.875}{0.887} = 0.987, \\ \cos(\mathbf{u}_5^{(2)}, \Delta u_g^{(1)}) &= \frac{(-0.5)(0.5) + (-0.4)(0.75)}{0.640 \times 0.901} = \frac{-0.550}{0.577} = -0.953. \end{aligned}$$

Normalizing into $[0, 1]$ via $S_i^{(2)} = (1 + \bar{S}_i^{(2)})/2$:

Client	$\cos(\mathbf{u}_i^{(2)}, \Delta u_g^{(1)})$	$S_i^{(2)}$	Interpretation
C_1	0.999	0.9995	Strong alignment
C_2	1.000	1.000	Strong alignment
C_3	-0.124	0.438	Weak / neutral
C_4	0.987	0.993	Strong alignment
C_5	-0.953	0.024	Opposing direction

Table 4.3: Global direction similarity scores at round $t = 2$.

Client C_5 , with $S_5^{(2)} = 0.024$, is strongly opposed to the global optimization direction and will receive a substantially reduced trust contribution from this component.

c. Cluster Alignment Score

For each client i in round t , we compute the cosine similarity between its update and all other clients j :

$$\text{sim}_{ij}^{(t)} = \cos(u_i^{(t)}, u_j^{(t)}), \quad \forall j \neq i \quad (4.10)$$

This produces a similarity matrix that describes the directional agreement between clients. We use cosine similarity because it isolates directional behavior independently of magnitude. This is important since the magnitude is already handled separately by $M_i^{(t)}$. Using cosine prevents double penalization and ensures that each trust component captures a distinct behavioral dimension.

Let $n_t = |\mathcal{S}_t|$ be the number of participating clients at round t .

For each client i , we sort the similarities $\{\text{sim}_{ij}^{(t)} : j \neq i\}$ in descending order.

We define:

$$k = \left\lfloor \frac{n_t}{2} \right\rfloor$$

and let $\mathcal{N}_i^{(t)}$ be the set containing the k clients with the highest similarity to client i .

We compute the average similarity over the selected neighbours:

$$\tilde{C}_i^{(t)} = \frac{1}{|\mathcal{N}_i^{(t)}|} \sum_{j \in \mathcal{N}_i^{(t)}} \text{sim}_{ij}^{(t)} \quad (4.11)$$

Since cosine similarity lies in $[-1, 1]$, we normalize it into $[0, 1]$:

$$C_i^{(t)} = \frac{\tilde{C}_i^{(t)} + 1}{2} \quad (4.12)$$

Thus:

- $C_i^{(t)} = 1$ indicates strong alignment with the majority.
- $C_i^{(t)} \approx 0.5$ indicates weak or neutral alignment.
- $C_i^{(t)} \approx 0$ indicates strong opposition to the majority.

A client whose update direction significantly deviates from the majority as in poisoning or coordinated Sybil attacks will obtain a low cluster alignment score.

Example 3. Consider six clients, four honest and two attackers, with the following update vectors at round t :

Client	Type	$\mathbf{u}_i^{(t)}$	Normalized unit vector
C_1	Honest	[0.52, 0.29]	[0.873, 0.487]
C_2	Honest	[0.48, 0.31]	[0.840, 0.543]
C_3	Honest	[0.51, 0.30]	[0.862, 0.507]
C_4	Honest	[0.49, 0.32]	[0.837, 0.547]
C_5	Attacker	[0.40, 0.45]	[0.664, 0.747]
C_6	Attacker	[0.10, 0.95]	[0.105, 0.995]

For each client, the normalized unit vector is computed using L2 normalization as follows:

$$\mathbf{u}_i^{norm} = \frac{\mathbf{u}_i}{\|\mathbf{u}_i\|_2} = \frac{\mathbf{u}_i}{\sqrt{u_{i1}^2 + u_{i2}^2}}$$

For example, for client C_1 :

$$\|\mathbf{u}\|_2 = \sqrt{0.52^2 + 0.29^2}$$

$$\mathbf{u}_1^{norm} = \frac{[0.52, 0.29]}{\|\mathbf{u}\|_2} = [0.873, 0.487]$$

Because the vectors are normalized to unit length, their dot product equals the cosine similarity directly. The resulting pairwise similarity matrix is:

	C_1	C_2	C_3	C_4	C_5	C_6
C_1	1.000	0.998	0.999	0.997	0.945	0.576
C_2	0.998	1.000	0.999	1.000	0.964	0.627
C_3	0.999	0.999	1.000	0.999	0.952	0.595
C_4	0.997	1.000	0.999	1.000	0.965	0.632
C_5	0.945	0.964	0.952	0.965	1.000	0.813
C_6	0.576	0.627	0.595	0.632	0.813	1.000

Table 4.4: Pairwise cosine similarity matrix.

With $n_t = 6$ clients, the neighbourhood size is $k = \lfloor 6/2 \rfloor = 3$. The cluster alignment scores are derived as follows:

$$\begin{aligned}
\tilde{C}_1^{(t)} &= \frac{1}{3}(1.000 + 0.999 + 0.998) = 0.999 & \Rightarrow C_1^{(t)} &= \frac{0.999+1}{2} = 0.9995, \\
\tilde{C}_5^{(t)} &= \frac{1}{3}(0.965 + 0.964 + 0.952) = 0.960 & \Rightarrow C_5^{(t)} &= \frac{0.960+1}{2} = 0.980, \\
\tilde{C}_6^{(t)} &= \frac{1}{3}(0.813 + 0.632 + 0.627) = 0.691 & \Rightarrow C_6^{(t)} &= \frac{0.691+1}{2} = 0.846.
\end{aligned}$$

The attacker C_6 receives the lowest cluster alignment score (0.845), reflecting its directional deviation from the honest majority.

d. Magnitude Consistency Score

We first compute the Euclidean norm (ℓ_2 magnitude) of each client's flattened update vector:

$$m_i^{(t)} = \|u_i^{(t)}\|_2 = \sqrt{\left(u_i^{(t)}\right)^\top u_i^{(t)}}.$$

The value $m_i^{(t)}$ represents the overall magnitude of the model modification proposed by client i .

Let $\mathcal{M}^{(t)} = \{m_j^{(t)} : j \in \mathcal{S}_t\}$ be the multiset of all update magnitudes at round t . We sort $\mathcal{M}^{(t)}$ in ascending order and define the lower half as:

$$\mathcal{M}_{\text{low}}^{(t)} = \text{smallest } \left\lfloor \frac{n_t}{2} \right\rfloor \text{ values of } \mathcal{M}^{(t)}.$$

We then define the robust reference median as:

$$\text{med}^{(t)} = \text{median}\left(\mathcal{M}_{\text{low}}^{(t)}\right).$$

The lower-half robust median is used instead of the standard median to increase robustness against large malicious updates. When a fraction of clients submit abnormally large updates (e.g., in a gradient scaling

attack), the standard median may shift upward, causing honest clients to appear small by comparison. By restricting the reference to the lower half of the distribution, the robust median remains anchored to the typical honest update magnitude even when up to 50% of clients are malicious.

We measure how far each client deviates from this robust reference:

$$R_i^{(t)} = \frac{m_i^{(t)}}{\text{med}^{(t)} + \varepsilon}, \quad (4.13)$$

where $\varepsilon > 0$ is a small constant to prevent division by zero.

We map $R_i^{(t)}$ into a bounded score $M_i^{(t)} \in [0, 1]$:

$$M_i^{(t)} = \begin{cases} 1, & 0.5 \leq R_i^{(t)} \leq 2, \\ 2R_i^{(t)}, & R_i^{(t)} < 0.5, \\ \frac{2}{R_i^{(t)}}, & R_i^{(t)} > 2. \end{cases} \quad (4.14)$$

We consider three cases:

- If $0.5 \leq R_i^{(t)} \leq 2$, the update magnitude is considered normal and $M_i^{(t)} = 1$.
- If $R_i^{(t)} > 2$, the update is excessively large and may attempt to dominate aggregation.
- If $R_i^{(t)} < 0.5$, the update is unusually small and may indicate gradient hiding or low contribution.

Example 4. Five clients submit updates at round $t = 1$:

Client	$\mathbf{u}_i^{(1)}$	$m_i = \ \mathbf{u}_i\ _2$
C_1	[0.50, 0.75]	$\sqrt{0.25 + 0.5625} = 0.901$
C_2	[0.40, 0.70]	$\sqrt{0.16 + 0.49} = 0.806$
C_3	[0.10, 0.20]	$\sqrt{0.01 + 0.04} = 0.224$
C_4	[0.45, 0.80]	$\sqrt{0.2025 + 0.64} = 0.918$
C_5	[-0.30, -0.40]	$\sqrt{0.09 + 0.16} = 0.500$

Sorting magnitudes in ascending order gives $\{0.224, 0.500, 0.806, 0.901, 0.918\}$. The lower half contains $\lfloor 5/2 \rfloor = 2$ values: $\{0.224, 0.500\}$, so $\text{med}^{(1)} = (0.224 + 0.500)/2 = 0.362$.

The deviation ratios $R_i = m_i/(\text{med} + \varepsilon)$ with $\varepsilon = 0.001$ and the resulting scores M_i are:

Client	m_i	R_i	Case	M_i
C_1	0.901	$0.901/0.363 = 2.482$	$R > 2$	$2/2.482 = 0.806$
C_2	0.806	$0.806/0.363 = 2.220$	$R > 2$	$2/2.220 = 0.901$
C_3	0.224	$0.224/0.363 = 0.617$	$0.5 \leq R \leq 2$	1.000
C_4	0.918	$0.918/0.363 = 2.529$	$R > 2$	$2/2.529 = 0.791$
C_5	0.500	$0.500/0.363 = 1.378$	$0.5 \leq R \leq 2$	1.000

Table 4.5: Magnitude consistency scores

In this example the robust median anchors on the two smallest values, which allows the score to correctly identify magnitude deviations relative to the lower portion of the distribution.

e. Historical Consistency Score

Variance measures how spread out a set of numbers is. We use it to determine whether a client's trust score stays stable (low variance) over time or jumps up and down (high variance).

Let $W_H = \min(5, t - 1)$ denote the observation window for historical consistency.

We define the sample mean of the recent trust values as:

$$\bar{T}_i^{(t)} = \frac{1}{W_H} \sum_{k=1}^{W_H} \text{Trust}_i^{(t-k)}.$$

The variance of the recent trust history is then:

$$\text{Var}_i^{(t)} = \frac{1}{W_H} \sum_{k=1}^{W_H} \left(\text{Trust}_i^{(t-k)} - \bar{T}_i^{(t)} \right)^2.$$

$\text{Trust}_i^{(t-k)}$ denotes the trust value of client i at round $t - k$ (at most 5 previous rounds).

We convert the variance into a bounded score:

$$H_i^{(t)} = \max\left(0, 1 - \min\left(1, \lambda \cdot \text{Var}_i^{(t)}\right)\right), \quad (4.15)$$

where $\lambda = 3$ is a sensitivity parameter that determines the impact of trust variance on the final trust score. Since $\text{Trust}_i^{(t)} \in [0, 1]$, the maximum possible variance is 0.25, which occurs when the trust value alternates between 0 and 1. We set $\lambda = 3$ to apply a strong but non-saturating penalty in this worst-case scenario ($3 \times 0.25 = 0.75$). This value reduces the effect of unstable clients while avoiding excessive penalties for honest clients, whose trust values may naturally vary due to non-IID data.

Thus, $H_i^{(t)} \in [0, 1]$:

- $H_i^{(t)} \approx 1$ indicates stable trust history.

- $H_i^{(t)} \approx 0$ indicates strong instability or behavioral oscillation.

For $t = 1$ or when fewer than 2 observations are available: $H_i^{(t)} = 1$.

Example 5. Client C_1 has the following trust history over the five most recent rounds at $t = 6$, giving $W_H = \min(5, t - 1) = 5$:

Round	$\text{Trust}_1^{(t-k)}$	$\text{Trust}_1^{(t-k)} - \bar{T}_1^{(6)}$	$\left(\text{Trust}_1^{(t-k)} - \bar{T}_1^{(6)}\right)^2$
$t - 5$	0.900	+0.028	0.000784
$t - 4$	0.850	−0.022	0.000484
$t - 3$	0.880	+0.008	0.000064
$t - 2$	0.860	−0.012	0.000144
$t - 1$	0.870	−0.002	0.000004

Table 4.6: Trust history of C_1 over the observation window.

The sample mean is:

$$\bar{T}_1^{(6)} = \frac{0.900 + 0.850 + 0.880 + 0.860 + 0.870}{5} = \frac{4.360}{5} = 0.872.$$

The variance over the window is:

$$\text{Var}_1^{(6)} = \frac{0.000784 + 0.000484 + 0.000064 + 0.000144 + 0.000004}{5} = \frac{0.001480}{5} = 0.000296.$$

Converting to a bounded score with $\lambda = 3$:

$$H_1^{(6)} = \max(0, 1 - \min(1, 3 \times 0.000296)) = 1 - 0.000888 \approx 0.999.$$

The near-unity score confirms that C_1 exhibits highly stable behaviour with negligible fluctuation across the observation window.

f. RawTrust Score

We can now calculate $\text{RawTrust}_i(t)$:

$$\text{RawTrust}_i(t) = w_s S_i(t) + w_c C_i(t) + w_m M_i(t) + w_h H_i(t) \quad (4.16)$$

$$\text{where : } w_s = 0.30, w_c = 0.20, w_m = 0.25, w_h = 0.25$$

This equation means that the instantaneous trust of client i at round t is a weighted sum of four different indicators:

- 30% of the decision is based on the global direction similarity of the client's update $S_i(t)$,
- 20% is based on the client's alignment with other clients $C_i(t)$,
- 25% reflects the magnitude consistency of the client's update $M_i(t)$,
- 25% accounts for the client's historical trust consistency $H_i(t)$.

Larger $\text{RawTrust}_i(t)$ values indicate that the client is more trustworthy, while smaller values indicate that the client is potentially suspicious or malicious.

Example 6. Consider client C_1 at round $t > 1$ with the following behavioral indicators:

$$S_1(t) = 0.95, \quad C_1(t) = 0.90, \quad M_1(t) = 0.96, \quad H_1(t) = 0.92,$$

and prior state $\text{Trust}_1(t-1) = 0.80$, $\text{Decay}_1(t) = 0.90$.

Step 1. RawTrust. Using weights $w_s = 0.30$, $w_c = 0.20$, $w_m = 0.25$, $w_h = 0.25$:

$$\begin{aligned} \text{RawTrust}_1(t) &= 0.30 \times 0.95 + 0.20 \times 0.90 + 0.25 \times 0.96 + 0.25 \times 0.92 \\ &= 0.285 + 0.180 + 0.240 + 0.230 = 0.935. \end{aligned}$$

Step 2. Trust update. With $\alpha = 0.60$ and $\beta = 0.40$:

$$\begin{aligned} \text{Trust}_1(t) &= \alpha \cdot \text{Trust}_1(t-1) \cdot \text{Decay}_1(t) + \beta \cdot \text{RawTrust}_1(t) \\ &= 0.60 \times 0.80 \times 0.90 + 0.40 \times 0.935 \\ &= 0.432 + 0.374 = 0.806. \end{aligned}$$

The updated trust score $\text{Trust}_1(t) = 0.806$ is slightly above the previous value of 0.80, indicating that the client remains reliable across consecutive rounds.

g. Trend Calculation

To capture behavioral dynamics, we introduce a trend component based on the evolution of the RawTrust score over a sliding window of $W_{\text{raw}} = \min(4, t-1)$ rounds.

We estimate the slope of RawTrust using least-squares linear regression over the recent window:

$$\beta_i^{(t)} = \frac{W_{\text{raw}} \sum_{k=1}^{W_{\text{raw}}} k \cdot \text{RawTrust}_i^{(t-k)} - \left(\sum_{k=1}^{W_{\text{raw}}} k \right) \left(\sum_{k=1}^{W_{\text{raw}}} \text{RawTrust}_i^{(t-k)} \right)}{W_{\text{raw}} \sum_{k=1}^{W_{\text{raw}}} k^2 - \left(\sum_{k=1}^{W_{\text{raw}}} k \right)^2} \quad (4.17)$$

where $k = 1$ corresponds to the most recent round and $k = W_{\text{raw}}$ to the oldest round in the window.

- $\beta_i^{(t)} > 0$: improving behavior.

- $\beta_i^{(t)} < 0$: degrading behavior.
- $\beta_i^{(t)} \approx 0$: stable behavior.

Note that the trend and volatility computation uses a window of $W_{\text{raw}} = \min(4, t - 1)$ rounds, while the historical consistency score uses a separate window of $W_H = \min(5, t - 1)$ rounds. These two windows are maintained independently, reflecting the different temporal scales of their respective mechanisms.

Example 7. Consider client C_1 at round $t = 6$, with $W_{\text{raw}} = \min(4, t - 1) = 4$. The RawTrust history over the window is:

k	Round	RawTrust ₁ ^($t-k$)
1	$t - 1$	0.82
2	$t - 2$	0.76
3	$t - 3$	0.70
4	$t - 4$	0.65

Table 4.7: RawTrust history of C_1 over the sliding window.

We apply the least-squares slope formula with $W_{\text{raw}} = 4$. First, compute the required sums:

$$\begin{aligned}
 \sum_{k=1}^4 k &= 1 + 2 + 3 + 4 = 10, \\
 \sum_{k=1}^4 k^2 &= 1 + 4 + 9 + 16 = 30, \\
 \sum_{k=1}^4 \text{RawTrust}_1^{(t-k)} &= 0.82 + 0.76 + 0.70 + 0.65 = 2.93, \\
 \sum_{k=1}^4 k \cdot \text{RawTrust}_1^{(t-k)} &= 1(0.82) + 2(0.76) + 3(0.70) + 4(0.65) \\
 &= 0.82 + 1.52 + 2.10 + 2.60 = 7.04.
 \end{aligned}$$

Substituting into the trend formula:

$$\beta_1^{(6)} = \frac{4 \times 7.04 - 10 \times 2.93}{4 \times 30 - 10^2} = \frac{28.16 - 29.30}{120 - 100} = \frac{-1.14}{20} = -0.057.$$

Since $\beta_1^{(6)} = -0.057 < 0$, the trend is **negative**, indicating that C_1 's RawTrust has been consistently degrading over the last four rounds. Depending on the threshold $\tau_{\text{trend}} = 0.01$, this will trigger the decay mechanism in the current round.

h. Adaptive Volatility

Volatility measures the stability of a client's behavior over recent rounds. In non-IID federated learning, honest clients naturally exhibit some variance due to data heterogeneity, so we use an adaptive threshold based on the population statistics.

A client is considered suspicious only if their variance exceeds one standard deviation above the mean variance of all clients. This prevents unfairly penalizing honest clients with heterogeneous data while still catching erratic attackers.

Trend captures direction, but not stability. An attacker performing intermittent attacks may avoid producing a clear negative slope while still generating unstable behavior. To detect such oscillations, we measure the variance of RawTrust over the same sliding window W_{raw} .

First, we compute the mean RawTrust over the window:

$$\hat{\mu}_i^{(t)} = \frac{1}{W_{\text{raw}}} \sum_{k=1}^{W_{\text{raw}}} \text{RawTrust}_i^{(t-k)}$$

Then, we compute the variance:

$$\sigma_i^2(t) = \frac{1}{W_{\text{raw}}} \sum_{k=1}^{W_{\text{raw}}} \left(\text{RawTrust}_i^{(t-k)} - \hat{\mu}_i^{(t)} \right)^2 \quad (4.18)$$

Low variance indicates stable behavior, while high variance suggests oscillating or erratic participation.

The adaptive volatility threshold is computed from the population of all client volatilities at round t :

$$\tau_{\text{vol}} = \mu_{\sigma^2}^{(t)} + \sigma_{\sigma^2}^{(t)}$$

where $\mu_{\sigma^2}^{(t)}$ and $\sigma_{\sigma^2}^{(t)}$ are the mean and standard deviation of $\{\sigma_j^2(t)\}$ across all clients j with sufficient history.

Example 8. Consider three clients with computed volatility values:

$$\sigma_1^2 = 0.002, \quad \sigma_2^2 = 0.010, \quad \sigma_3^2 = 0.030.$$

Assume the population statistics are:

$$\mu_{\sigma^2}^{(t)} = 0.015, \quad \sigma_{\sigma^2}^{(t)} = 0.008.$$

Thus, the adaptive threshold is:

$$\tau_{\text{vol}} = 0.015 + 0.008 = 0.023.$$

We classify clients as follows:

$$\sigma_1^2 < \tau_{vol} \Rightarrow \text{stable}, \quad \sigma_2^2 < \tau_{vol} \Rightarrow \text{stable}, \quad \sigma_3^2 > \tau_{vol} \Rightarrow \text{suspicious}.$$

Client C_3 is flagged due to high volatility indicating unstable behavior.

i. Trust Recovery Mechanism

Let good_i denote the number of consecutive rounds in which client i has exhibited improving behavior, defined by:

$$\text{if } \beta_i^{(t)} > \tau_{\text{trend}} \text{ and } \sigma_i^2(t) \leq \tau_{\text{vol}} : \quad \text{good}_i \leftarrow \text{good}_i + 1, \quad \text{bad}_i \leftarrow 0$$

where $\tau_{\text{trend}} = 0.01$.

When a client demonstrates sustained improving behavior ($\text{good}_i \geq 3$), a small bonus is added to its raw trust score before the final trust update:

$$\overline{\text{RawTrust}}_i^{(t)} = \begin{cases} \min(1, \text{RawTrust}_i^{(t)} + \rho \cdot \text{good}_i), & \text{if } \text{good}_i \geq 3, \\ \text{RawTrust}_i^{(t)}, & \text{otherwise,} \end{cases} \quad (4.19)$$

with $\rho = 0.05$.

j. Decay Mechanism

Let bad_i denote the number of consecutive rounds with degrading or unstable behavior:

$$\text{if } \beta_i^{(t)} < -\tau_{\text{trend}} \text{ or } \sigma_i^2(t) > \tau_{\text{vol}} : \quad \text{bad}_i \leftarrow \text{bad}_i + 1, \quad \text{good}_i \leftarrow 0$$

$$\text{Decay}_i(t) = \begin{cases} \gamma \cdot \delta^{\text{bad}_i}, & \text{if } \text{bad}_i \geq 2, \\ \gamma, & \text{otherwise,} \end{cases} \quad (4.20)$$

where $\gamma = 0.9$ and $\delta = 0.9$.

$\text{Decay}_i(t)$ is the retention ratio of a client's previous trust when updating its trust score in the current round. Decay values lie between 0 and 1, allowing past trust to be fully or partially retained. Occasional errors slightly reduce Decay, while repeated misbehavior progressively diminishes it, ensuring gradual rather than abrupt penalization of historical trust.

γ represents the maximum possible value of the decay factor, i.e., the upper limit of how much past trust can be retained. We chose $\gamma = 0.9$ because it is close to 1, meaning that minor mistakes do not cause a significant loss of trust. This choice retains reliable clients while gradually penalizing clients with

repeated misbehavior over successive rounds.

δ is a factor used to gradually reduce a client's previous trust when they behave badly in consecutive rounds. We chose $\delta = 0.9$ so that the reduction of previous trust occurs gradually in response to repeated misbehavior, rather than dropping abruptly.

Decay Value	Effect
1.0	No penalty
0.9	Light penalty
0.5	Moderate penalty
0.1	Strong penalty
0.0	History reset

Table 4.8: Impact of Decay Values on Client Trust

Example 9. The decay factor $\text{Decay}_i(t)$ is governed by the consecutive-bad-round counter bad_i , with parameters $\gamma = 0.9$ and $\delta = 0.9$:

$$\text{Decay}_i(t) = \begin{cases} \gamma \cdot \delta^{\text{bad}_i}, & \text{if } \text{bad}_i \geq 2, \\ \gamma, & \text{otherwise.} \end{cases}$$

The following table illustrates the progressive effect of repeated misbehaviour:

bad_i	$\text{Decay} = 0.9 \times 0.9^{\text{bad}_i}$	Effect on prior trust
0 or 1	0.900	Light penalty
2	$0.9 \times 0.81 = 0.729$	Moderate penalty
3	$0.9 \times 0.729 = 0.656$	Significant penalty
4	$0.9 \times 0.656 = 0.590$	Strong penalty
5	$0.9 \times 0.590 = 0.531$	Very strong penalty

Table 4.9: Decay values as a function of consecutive bad rounds.

The gradual decrease ensures that isolated mistakes do not cause a disproportionate loss of trust, while persistent misbehaviour across multiple rounds leads to progressively stronger penalization of accumulated historical trust.

k. Trust Score Update Mechanism

The trust of client i at communication round t is defined as:

$$\text{Trust}_i(t) = \begin{cases} \overline{\text{RawTrust}_i}(t), & \text{if } t = 1, \\ \alpha \cdot \text{Trust}_i(t-1) \cdot \text{Decay}_i(t) + \beta \cdot \overline{\text{RawTrust}_i}(t), & \text{if } t > 1, \end{cases} \quad (4.21)$$

where $\alpha = 0.6, \beta = 0.4$.

Finally, the trust score is bounded to $[0, 1]$:

$$\text{Trust}_i(t) = \min(1, \max(0, \text{Trust}_i(t)))$$

$\text{Trust}_i(t)$ is the trust score of client i at communication round t .

- A high value indicates that the client is consistent, reliable, and aligned with the global model.
- A low value indicates that the client may be faulty, noisy, or potentially malicious.

$\overline{\text{RawTrust}}_i(t)$ is the instantaneous trust score of client i at round t , boosted if the client has demonstrated sustained good behavior.

$\text{Trust}_i(t - 1)$ denotes the trust score of the previous round.

$\text{Decay}_i(t)$ represents the proportion of the client's previous trust that is retained when updating trust in the current round. For $t > 1$, 60% of the previous round's trust, adjusted by the retention factor $\text{Decay}_i(t)$, is combined with 40% of the current round's trust. The weights of 60% and 40% are chosen to prevent excessive penalization: even if a client had high trust in previous rounds, a minor mistake in the current round should not cause a disproportionate drop.

Example 10. We trace client C_1 across three consecutive rounds to illustrate the full update mechanism, with $\alpha = 0.60, \beta = 0.40, \gamma = 0.9, \delta = 0.9$, and $\tau_{\text{trend}} = 0.01$.

Round $t = 1$. No prior history is available, so the trust score is initialized directly:

$$\text{Trust}_1(1) = \overline{\text{RawTrust}}_1(1) = 0.935.$$

Round $t = 2$. The behavioral trend $\beta_1^{(2)} > \tau_{\text{trend}}$ and volatility $\sigma_1^2(2) \leq \tau_{\text{vol}}$, so $\text{bad}_1 = 0$ and $\text{Decay}_1(2) = \gamma = 0.90$:

$$\begin{aligned} \text{Trust}_1(2) &= 0.60 \times \text{Trust}_1(1) \times 0.90 + 0.40 \times \overline{\text{RawTrust}}_1(2) \\ &= 0.60 \times 0.935 \times 0.90 + 0.40 \times 0.970 \\ &= 0.505 + 0.388 = 0.893. \end{aligned}$$

Round $t = 3$. The client exhibits degrading behaviour for two consecutive rounds ($\text{bad}_1 = 2$), so $\text{Decay}_1(3) = 0.9 \times 0.9^2 = 0.729$:

$$\begin{aligned} \text{Trust}_1(3) &= 0.60 \times \text{Trust}_1(2) \times 0.729 + 0.40 \times \overline{\text{RawTrust}}_1(3) \\ &= 0.60 \times 0.893 \times 0.729 + 0.40 \times 0.820 \\ &= 0.391 + 0.328 = 0.719. \end{aligned}$$

Round t	RawTrust	bad_i	Decay	$\alpha \cdot \text{Trust}(t-1) \cdot \text{Decay}$	Trust(t)
1	0.935	—	—	—	0.935
2	0.970	0	0.900	0.505	0.893
3	0.820	2	0.729	0.391	0.719

Table 4.10: Trust score evolution for C_1 over three communication rounds.

The decay triggered at $t = 3$ reduces the trust score from 0.893 to 0.719, reflecting the client's degrading behavioral pattern while still preserving a portion of its accumulated historical trust.

I. Trust-Enhanced Spectral Clustering Formulation

before the final aggregation, to improve robustness against coordinated and Byzantine attacks, we perform clustering of client updates before aggregation. Unlike classical clustering, we incorporate trust scores directly into the similarity structure.

Cosine similarity matrix : For all clients i, j in the fog, we compute the pairwise cosine similarity of their normalized update vectors:

$$A_{ij}^{(t)} = \frac{\hat{u}_i^{(t)} \cdot \hat{u}_j^{(t)}}{\|\hat{u}_i^{(t)}\| \|\hat{u}_j^{(t)}\|}, \quad (4.22)$$

$$\hat{u}_i^{(t)} = \frac{u_i^{(t)}}{\|u_i^{(t)}\| + \varepsilon}, \quad (4.23)$$

with $A_{ii}^{(t)} = 0$.

Trust affinity matrix : We then construct a trust outer-product matrix:

$$T_{ij}^{(t)} = \text{Trust}_i^{(t)} \cdot \text{Trust}_j^{(t)} \quad (4.24)$$

Enhanced affinity matrix : We combine geometric similarity with trust-based affinity:

$$\tilde{A}_{ij}^{(t)} = \lambda_s \cdot A_{ij}^{(t)} + \lambda_t \cdot T_{ij}^{(t)} \quad (4.25)$$

where $\lambda_s = 0.6$ and $\lambda_t = 0.4$.

Spectral clustering : Apply spectral clustering with $K = 2$ clusters using $\tilde{A}^{(t)}$ as the precomputed affinity matrix, producing cluster labels $\ell_i \in \{0, 1\}$.

Trusted cluster selection : We select the cluster with the highest mean trust score as the most reliable cluster :

$$c^* = \arg \max_{c \in \{0,1\}} \frac{1}{|\mathcal{C}_c|} \sum_{i \in \mathcal{C}_c} \text{Trust}_i^{(t)}, \quad \mathcal{C}_c = \{i : \ell_i = c\} \quad (4.26)$$

The final trusted client set is defined as:

$$\mathcal{C}^* = \mathcal{C}_{c^*} \quad (4.27)$$

Example 11. To illustrate the proposed trust-enhanced spectral clustering mechanism, consider a fog node receiving updates from 5 clients at round t .

Each client i provides a flattened update vector $u_i^{(t)}$, while the normalized update vector $\hat{u}_i^{(t)}$ is computed as

$$\hat{u}_i^{(t)} = \frac{u_i^{(t)}}{\|u_i^{(t)}\|_2 + \varepsilon}.$$

Client	$u_i^{(t)}$	$\hat{u}_i^{(t)}$
C_1	[0.60, 0.80]	[0.60, 0.80]
C_2	[0.55, 0.75]	[0.592, 0.806]
C_3	[0.58, 0.78]	[0.597, 0.802]
C_4	[-0.40, -0.30]	[-0.800, -0.600]
C_5	[-0.35, -0.45]	[-0.614, -0.789]

Table 4.11: Client update vectors and their normalized representations

Step 1: We compute the pairwise cosine similarity $A_{ij}^{(t)}$ of their normalized update vectors:

	C_1	C_2	C_3	C_4	C_5
C_1	0	1.000	1.000	-0.960	-1.000
C_2	1.000	0	1.000	-0.957	-1.000
C_3	1.000	1.000	0	-0.959	-1.000
C_4	-0.960	-0.957	-0.959	0	0.965
C_5	-1.000	-1.000	-1.000	0.965	0

Table 4.12: Cosine Similarity Matrix $A_{ij}^{(t)}$ between client updates

Step 2: Each client is assigned a normalized trust score:

Client	$Trust_i^{(t)}$
C_1	0.95
C_2	0.92
C_3	0.94
C_4	0.30
C_5	0.25

Table 4.13: Client trust values at round t

We construct the trust affinity matrix $T_{ij}^{(t)}$

	C_1	C_2	C_3	C_4	C_5
C_1	0.9025	0.8740	0.8930	0.2850	0.2375
C_2	0.8740	0.8464	0.8648	0.2760	0.2300
C_3	0.8930	0.8648	0.8836	0.2820	0.2350
C_4	0.2850	0.2760	0.2820	0.0900	0.0750
C_5	0.2375	0.2300	0.2350	0.0750	0.0625

Table 4.14: Trust Affinity Matrix

Step 3 : We combine geometric similarity and trust-based affinity to obtain the final enhanced matrix $\overline{A}_{ij}^{(t)}$.

	C_1	C_2	C_3	C_4	C_5
C_1	0	0.9496	0.9499	-0.4620	-0.6000
C_2	0.9496	0	0.9499	-0.4562	-0.6000
C_3	0.9499	0.9499	0	-0.4584	-0.6000
C_4	-0.4620	-0.4562	-0.4584	0	0.6090
C_5	-0.6000	-0.6000	-0.6000	0.6090	0

Table 4.15: Enhanced Affinity Matrix $\overline{A}_{ij}^{(t)}$

Step 4: To determine the cluster assignments, we analyze the structure of the enhanced affinity matrix $\overline{A}_{ij}^{(t)}$.

We observe that clients C_1 , C_2 , and C_3 have strong positive similarities (close to 1) between each other, indicating highly consistent updates.

In contrast, their similarities with C_4 and C_5 are negative (around -0.5 to -0.6), suggesting strong dissimilarity.

On the other hand, clients C_4 and C_5 exhibit a positive similarity (0.6090), meaning they form a separate group.

Based on this similarity pattern, we identify two distinct clusters:

$$\mathcal{C}_0 = \{C_1, C_2, C_3\}, \quad \mathcal{C}_1 = \{C_4, C_5\}$$

We then assign cluster labels accordingly:

Client	Cluster Label ℓ_i
C_1	0
C_2	0
C_3	0
C_4	1
C_5	1

Table 4.16: Cluster assignments derived from affinity structure

Step 5: We compute the average trust score of each cluster in order to identify the most reliable group of clients.

For cluster $\mathcal{C}_0 = \{C_1, C_2, C_3\}$:

$$\frac{1}{3}(0.95 + 0.92 + 0.94) = \frac{2.81}{3} = 0.9367$$

For cluster $\mathcal{C}_1 = \{C_4, C_5\}$:

$$\frac{1}{2}(0.30 + 0.25) = \frac{0.55}{2} = 0.2750$$

Cluster	Mean Trust Score
$\mathcal{C}_0$	0.9367
$\mathcal{C}_1$	0.2750

Table 4.17: Average trust score per cluster

We select the cluster with the highest mean trust score:

$$c^* = 0$$

Thus, the final trusted client set is:

$$\mathcal{C}^* = \mathcal{C}_0 = \{C_1, C_2, C_3\}$$

Clients C_1 , C_2 , and C_3 are identified as reliable since they exhibit both high mutual similarity and high trust scores. In contrast, clients C_4 and C_5 have low trust values and form a separate cluster, indicating potentially unreliable or malicious behavior.

4.5 Algorithm

The TWA-HFL aggregation procedure is presented across three algorithm blocks due to its length. The first block (**Algorithm 1 : Update preprocessing and behavioral indicators.**) vectorizes the raw client updates and computes three of the four behavioral indicators (global-direction similarity S_i , peer-cluster similarity C_i , and magnitude consistency M_i) leaving historical trust stability H_i for the next block. The second block (**Algorithm 2 : RawTrust aggregation and temporal behaviour analysis.**) completes the indicator stage with H_i , fuses all four indicators into a RawTrust score $RT_i^{(t)}$, and performs temporal behavior analysis to distinguish sustained trends from momentary noise, producing an adjusted score $\overline{RT}_i^{(t)}$ and decay factor $\text{Decay}_i^{(t)}$. The third block (**Algorithm 3 : Trust consolidation, clustering, clipping, and aggregation.**) consolidates the final trust score $T_i^{(t)}$, applies trust-enhanced spectral clustering to isolate suspicious participants, clips update norms, and computes the trust-weighted global aggregation $u_g^{(t)}$. Together, the three blocks form a single coherent aggregation round executed entirely on the server side.

Algorithm 1: TWA-HFL Framework

Input : Client updates $\{\Delta w_i^{(t)}\}_{i \in \mathcal{S}_t}$,
 global model $w^{(t-1)}$,
 previous global model $w^{(t-2)}$,
 raw-trust history window of size $W_r=4$,
 trust history window of size $W_h=5$
Output: Fog weighted aggregation $u_g^{(t)}$

// Step 1: Compute flattened updates

1 **for** each client $i \in \mathcal{S}_t$ **do**

2 | $u_i \leftarrow \text{vec}(\Delta w_i^{(t)}) \in \mathbb{R}^d$

3 **end**

// Step 2: behavioral indicators

4 **(S) Global-direction similarity**

5 $K \leftarrow \min(3, t-1)$.

6 If $K=0$: $S_i \leftarrow 1.0$.

7 Else:

8

$$S_i = \frac{1}{2} \left(1 + \frac{1}{K} \sum_{k=1}^K \frac{u_i \cdot \Delta u_g^{(t-k)}}{\|u_i\| \|\Delta u_g^{(t-k)}\| + \varepsilon} \right)$$

9 **(C) Peer-cluster similarity**

10 If $|\mathcal{S}_t| < 3$: $C_i \leftarrow 1.0$.

11 Else: $k \leftarrow \lfloor |\mathcal{S}_t|/2 \rfloor$;

12 select top- k peers $\mathcal{N}_i^{(t)}$ by cosine similarity:

13

$$C_i = \frac{1}{2} \left(1 + \frac{1}{|\mathcal{N}_i^{(t)}|} \sum_{j \in \mathcal{N}_i^{(t)}} \cos(u_i, u_j) \right)$$

14 **(M) Magnitude consistency:**

15 Let $m_i = \sqrt{\left(u_i^{(t)}\right)^\top u_i^{(t)}}$ and $m_{\text{ref}} = \text{median}(\{\|u_j\|\}_{j: \|u_j\| \leq \text{median}(\|u_j\|)})$ (lower-half median).

16 $R_i \leftarrow m_i / (m_{\text{ref}} + \varepsilon)$.

17

$$M_i = \begin{cases} 1.0 & \text{if } 0.5 \leq R_i \leq 2.0, \\ 2 R_i & \text{if } R_i < 0.5, \\ \min(1, 2/R_i) & \text{otherwise.} \end{cases}$$

Algorithm 2: TWA-HFL Framework (*cont.*)

18 **(H) Historical trust stability:**

19 If $|\mathcal{H}_i| \geq 2$: let $\sigma_H^2 = \text{Var}(\mathcal{H}_i)$:

$$H_i = \max(0, 1 - \min(1, \lambda_{\text{var}} \sigma_H^2)), \quad \lambda_{\text{var}} = 3$$

Else $H_i \leftarrow 1.0$.

// Step 3: RawTrust score

20 **for each client** $i \in \mathcal{S}_t$ **do**

21 | $\text{RT}_i^{(t)} \leftarrow 0.30 S_i + 0.20 C_i + 0.25 M_i + 0.25 H_i$

22 | Append $\text{RT}_i^{(t)}$ to sliding window $\mathcal{R}_i$

23 **end**

// Step 4: Temporal behaviour analysis

24 Compute population volatility threshold: $\tau_{\text{vol}} \leftarrow \mu_{\sigma^2}^{(t)} + \sigma_{\sigma^2}^{(t)}$ over all i with $|\mathcal{R}_i| \geq 2$

25 **for each client** $i \in \mathcal{S}_t$ **do**

26 | **if** $|\mathcal{R}_i| < 2$ **then**

27 | | $\overline{\text{RT}}_i^{(t)} \leftarrow \text{RT}_i^{(t)}$; $\text{Decay}_i^{(t)} \leftarrow \gamma$; **continue**

28 | **end**

29 | $W_r \leftarrow \min(4, |\mathcal{R}_i|)$; index $\mathcal{R}_i^{(w)}$ as r_k with $k=1$ oldest, $k=W_r$ most recent

30 |

$$\beta_i = \frac{W_r \sum_{k=1}^{W_r} k r_k - (\sum k)(\sum r_k)}{W_r \sum_{k=1}^{W_r} k^2 - (\sum k)^2 + \varepsilon}$$

Volatility: $\sigma_i^2(t) = \text{Var}(\mathcal{R}_i)$

31 | **if** $\beta_i > \tau_{\text{trend}}$ **and** $\sigma_i^2(t) \leq \tau_{\text{vol}}$ **then**

32 | | $\text{good}_i \leftarrow \text{good}_i + 1$; $\text{bad}_i \leftarrow 0$

33 | | **if** $\text{good}_i \geq 3$ **then**

34 | | | $\overline{\text{RT}}_i^{(t)} \leftarrow \min(1, \text{RT}_i^{(t)} + \rho \cdot \text{good}_i)$ ($\rho = 0.05$)

35 | | **else**

36 | | | $\overline{\text{RT}}_i^{(t)} \leftarrow \text{RT}_i^{(t)}$

37 | | **end**

38 | | $\text{Decay}_i^{(t)} \leftarrow \gamma$

39 | **else if** $\beta_i < -\tau_{\text{trend}}$ **or** $\sigma_i^2(t) > \tau_{\text{vol}}$ **then**

40 | | $\text{bad}_i \leftarrow \text{bad}_i + 1$; $\text{good}_i \leftarrow 0$

41 | | $\overline{\text{RT}}_i^{(t)} \leftarrow \text{RT}_i^{(t)}$

42 | | **if** $\text{bad}_i \geq 2$ **then**

43 | | | $\text{Decay}_i^{(t)} \leftarrow \gamma \cdot \delta^{\text{bad}_i}$ ($\gamma=0.9, \delta=0.9$)

44 | | **else**

45 | | | $\text{Decay}_i^{(t)} \leftarrow \gamma$

46 | | **end**

47 | **else**

48 | | $\overline{\text{RT}}_i^{(t)} \leftarrow \text{RT}_i^{(t)}$; $\text{Decay}_i^{(t)} \leftarrow \gamma$

49 | **end**

50 **end**

Algorithm 3: TWA-HFL Framework (*cont.*)

```

// Step 5: Trust update
51 for each client  $i \in \mathcal{S}_t$  do
52   if  $t = 1$  then
53      $T_i^{(t)} \leftarrow \overline{\text{RT}}_i^{(t)}$ 
54   else
55      $T_i^{(t)} \leftarrow \alpha \cdot T_i^{(t-1)} \cdot \text{Decay}_i^{(t)} + \beta \cdot \overline{\text{RT}}_i^{(t)} \quad (\alpha=0.6, \beta=0.4)$ 
56   end
57    $T_i^{(t)} \leftarrow \min(1, \max(0, T_i^{(t)}))$ ; Append  $T_i^{(t)}$  to  $\mathcal{H}_i$ 
58 end

// Step 6: Trust-enhanced spectral clustering
59 if  $|\mathcal{S}_t| \geq 4$  then
60    $\hat{u}_i \leftarrow u_i / (\|u_i\| + \varepsilon)$  for all  $i$ 
61    $A_{ij} \leftarrow \hat{u}_i \cdot \hat{u}_j, \quad A_{ii} \leftarrow 0$  (cosine similarity matrix)
62    $T_{ij} \leftarrow T_i^{(t)} \cdot T_j^{(t)}$  (trust affinity matrix)
63    $\tilde{A}_{ij} \leftarrow 0.6 A_{ij} + 0.4 T_{ij}$  (enhanced affinity matrix)
64   Apply spectral clustering with  $K=2$  on  $\tilde{A}$ ; obtain labels  $\ell_i \in \{0, 1\}$ 
65   
$$c^* = \arg \max_{c \in \{0,1\}} \frac{1}{|\mathcal{C}_c|} \sum_{i \in \mathcal{C}_c} T_i^{(t)}, \quad \mathcal{C}_c = \{i : \ell_i = c\}$$

66    $\mathcal{S}_t^* \leftarrow \mathcal{C}_{c^*}$ 
67 else
68    $\mathcal{S}_t^* \leftarrow \mathcal{S}_t$ 
69 end

// Step 7: Norm clipping
69  $m_{\text{ref}} \leftarrow \text{median}(\{\|u_i\| : i \in \mathcal{S}_t^*\})$ 
70 for each client  $i \in \mathcal{S}_t^*$  do
71    $\tilde{u}_i \leftarrow u_i \cdot \min\left(1, \frac{m_{\text{ref}}}{\|u_i\| + \varepsilon}\right)$ 
72 end

// Step 8: Trust-weighted aggregation
73 Stabilized softmax weights (max-shift + floor):  $m \leftarrow \max_{j \in \mathcal{S}_t^*} T_j^{(t)}$ 
74 for each client  $i \in \mathcal{S}_t^*$  do
75    $\hat{\epsilon}_i \leftarrow \max(\exp(T_i^{(t)} - m), 0.01)$ 
76 end
77   
$$\epsilon_i \leftarrow \frac{\hat{\epsilon}_i}{\sum_{j \in \mathcal{S}_t^*} \hat{\epsilon}_j} \quad \forall i \in \mathcal{S}_t^* \quad u_g^{(t)} \leftarrow \sum_{i \in \mathcal{S}_t^*} \epsilon_i \cdot \tilde{u}_i$$

78 return  $u_g^{(t)}$ 

```

a. Algorithmic Workflow

- **Lines 1–3: Compute client updates**

Each participating client computes its local model update $\Delta w_i^{(t)}$. The server then flattens each update into a vector representation u_i so that all updates can be compared mathematically using cosine similarity and vector norms.

- **Lines 4–19: Compute behavioral indicators**

For every client, the algorithm evaluates four trust-related indicators:

- **Global-direction similarity (Lines 4–8)**

This indicator measures how consistently a client’s current update aligns with the recent trajectory of the global model. The cosine similarity between the client’s update u_i and each of the last $K \leq 3$ global model differences $\Delta u_g^{(t-k)}$ is computed and averaged. The result is shifted into the range $[0, 1]$ so that a perfect alignment yields $S_i = 1.0$. Clients whose updates deviate from the established global direction receive a lower S_i , flagging them as potential adversaries or stragglers. When no previous global update is available ($K = 0$), a neutral score of $S_i = 1.0$ is assigned.

- **Peer-cluster similarity (Lines 9–13)**

This indicator assesses how well a client’s update agrees with the majority of its peers within the current round. The top- k most similar peers $\mathcal{N}_i^{(t)}$ are selected using cosine similarity, where $k = \lfloor |\mathcal{S}_t|/2 \rfloor$. The average pairwise cosine similarity is then shifted into $[0, 1]$ to produce C_i . A high C_i indicates that the client’s update is consistent with the dominant cluster of updates, while a low C_i suggests the client may be an outlier or running a divergent local objective. When fewer than three clients participate, all receive $C_i = 1.0$ to avoid unreliable pairwise comparisons.

- **Magnitude consistency (Lines 14–17)**

This indicator detects abnormal update magnitudes that could indicate model poisoning or gradient amplification attacks. The norm m_i of the client’s update is compared against a robust reference m_{ref} defined as the median of the lower half of all client norms, making it resistant to a small fraction of extreme outliers. The ratio $R_i = m_i / (m_{\text{ref}} + \varepsilon)$ is then mapped to a score $M_i \in [0, 1]$: clients within a factor of two of the reference ($0.5 \leq R_i \leq 2.0$) receive the maximum score, while those with abnormally small or large updates are penalized proportionally.

- **Historical trust stability (Lines 18–19)**

This indicator rewards clients that have demonstrated consistent trust behaviour over previous rounds. The variance σ_H^2 of the trust history $\mathcal{H}_i$ is computed, and H_i is derived by penalizing high variance using a sensitivity parameter $\lambda_{\text{var}} = 3$. A client whose trust scores have been stable across rounds receives $H_i \approx 1.0$, while erratic behaviour — such as alternating between

high and low trust — yields a lower H_i . When fewer than two historical observations are available, a neutral value $H_i = 1.0$ is used.

• **Lines 20–23: Compute RawTrust score**

The four indicators are combined into a weighted score called *RawTrust*:

$$RT_i^{(t)} = 0.30 S_i + 0.20 C_i + 0.25 M_i + 0.25 H_i$$

Global-direction similarity is assigned the highest weight (0.30) as alignment with the global learning trajectory is the most direct signal of benign intent. Peer-cluster similarity receives the lowest weight (0.20) since its reliability depends on having a sufficiently large and uncompromised peer group. Magnitude consistency and historical stability each contribute equally (0.25), reflecting their complementary roles in detecting current anomalies and capturing long-term behaviour. The obtained value is stored inside a sliding history window $\mathcal{R}_i$ for future behavioral analysis.

• **Lines 24–49: Temporal behaviour analysis**

The algorithm studies the behaviour of each client over time instead of relying only on the current round.

- A trend score β_i is computed as the ordinary-least-squares slope of the client’s recent RawTrust values within the sliding window $\mathcal{R}_i$, capturing whether the client’s trustworthiness is improving or deteriorating.
- A volatility score $\sigma_i^2(t)$ is computed as the variance of the same window and compared against a population-level threshold τ_{vol} , detecting clients whose behaviour fluctuates erratically across rounds.

Clients showing a positive trend and low volatility over at least three consecutive rounds are rewarded through a small trust bonus $\rho \cdot \text{good}_i$ (with $\rho = 0.05$), incentivizing consistently honest participation.

Clients showing a negative trend or high volatility have their decay factor reduced as $\gamma \cdot \delta^{\text{bad}_i}$ (with $\gamma = 0.9$, $\delta = 0.9$), progressively dampening their past trust contributions and reducing their future influence gradually rather than removing them abruptly.

• **Lines 51–58: Trust update**

The final trust score $T_i^{(t)}$ is updated by combining the decayed historical trust with the adjusted RawTrust score:

$$T_i^{(t)} = \alpha \cdot T_i^{(t-1)} \cdot \text{Decay}_i^{(t)} + \beta \cdot \overline{\text{RT}}_i^{(t)}, \quad \alpha = 0.6, \beta = 0.4.$$

The momentum coefficient $\alpha = 0.6$ ensures that accumulated historical evidence outweighs any single round’s observation, preventing sudden trust fluctuations caused by isolated anomalies. The

freshness coefficient $\beta = 0.4$ ensures the score remains responsive to genuine behavioral changes. The result is clipped to $[0, 1]$ and appended to the trust history $\mathcal{H}_i$ for use in subsequent rounds.

• **Lines 59–68: Trust-enhanced spectral clustering**

The algorithm builds an enhanced affinity matrix $\tilde{A}$ combining:

- the cosine similarity matrix A_{ij} capturing geometric agreement between client updates, and
- the trust affinity matrix $T_{ij} = T_i^{(t)} \cdot T_j^{(t)}$ encoding mutual trustworthiness between pairs of clients.

The two matrices are blended as $\tilde{A}_{ij} = 0.6 A_{ij} + 0.4 T_{ij}$, weighting update geometry more heavily while still incorporating trust reputation. Spectral clustering is then applied to divide clients into two groups. The cluster with the highest average trust score is selected as the trusted set $\mathcal{S}_t^*$, while clients in the lower-trust cluster are excluded from aggregation. When fewer than four clients participate, clustering is bypassed and all clients are retained.

• **Lines 69–72: Norm clipping**

The magnitude of each trusted client’s update is clipped using the median norm of $\mathcal{S}_t^*$ as the reference threshold:

$$\tilde{u}_i = u_i \cdot \min\left(1, \frac{m_{\text{ref}}}{\|u_i\| + \varepsilon}\right).$$

Updates whose norm exceeds the median are scaled down to match it, while smaller updates are left unchanged. This prevents any single client from dominating the aggregation through excessively large gradients — a common strategy in scaling and boosting attacks — while preserving the update direction intact.

• **Lines 73–77: Trust-weighted aggregation**

The remaining trusted clients receive aggregation weights derived from their trust scores through a numerically stabilized softmax. A max-shift ($\exp(T_i^{(t)} - m)$ where $m = \max_j T_j^{(t)}$) prevents floating-point overflow, and a floor of 0.01 ensures that no client is assigned a zero weight, preserving a minimal contribution from all retained participants. Clients with higher trust scores therefore obtain proportionally larger aggregation weights, and the global update is computed as:

$$u_g^{(t)} = \sum_{i \in \mathcal{S}_t^*} \epsilon_i \tilde{u}_i,$$

where ϵ_i represents the trust-based aggregation weight normalized to sum to one over $\mathcal{S}_t^*$.

b. Computational Complexity Analysis

Let:

- $n = |\mathcal{S}_t|$ be the number of participating clients,
- d be the dimension of the model update vector.

The computational complexity of TWA-HFL can be analyzed step by step as follows:

- **Step 1: Flattened updates** Each client update is vectorized once:

$$\mathcal{O}(nd)$$

- **Step 2: behavioral indicators**

- **Global-direction similarity:** Computing cosine similarity with up to $K = 3$ previous global updates:

$$\mathcal{O}(nd)$$

since K is constant.

- **Peer-cluster similarity:** Requires pairwise cosine similarities between client updates:

$$\mathcal{O}(n^2d)$$

- **Magnitude consistency:** Computing norms and the robust median:

$$\mathcal{O}(nd) + \mathcal{O}(n \log n)$$

- **Historical trust stability:** Variance computation over fixed sliding windows:

$$\mathcal{O}(n)$$

Therefore, Step 2 is dominated by:

$$\mathcal{O}(n^2d)$$

- **Step 3: RawTrust computation** Weighted score computation for all clients:

$$\mathcal{O}(n)$$

- **Step 4: Temporal behaviour analysis** Trend and variance calculations use fixed-size windows ($W_r = 4$), therefore:

$$\mathcal{O}(n)$$

- **Step 5: Trust update** Updating trust values for all clients:

$$\mathcal{O}(n)$$

- **Step 6: Trust-enhanced spectral clustering**

- Construction of the enhanced affinity matrix reuses the pairwise cosine similarities already computed in Step 2. The trust affinity matrix $T_{ij} = T_i^{(t)} \cdot T_j^{(t)}$ requires only $\mathcal{O}(n^2)$, and the weighted combination of both matrices is likewise $\mathcal{O}(n^2)$.
- Spectral clustering on the $n \times n$ enhanced affinity matrix requires eigendecomposition:

$$\mathcal{O}(n^3)$$

Thus, Step 6 contributes:

$$\mathcal{O}(n^3)$$

with no additional n^2d term beyond what was already paid in Step 2.

- **Step 7: Norm clipping** Norm computation and clipping for trusted clients:

$$\mathcal{O}(nd)$$

- **Step 8: Trust-weighted aggregation** Softmax weight computation and weighted aggregation:

$$\mathcal{O}(nd)$$

Therefore, the overall computational complexity of the proposed TWA-HFL aggregation algorithm, after accounting for the reuse of pairwise similarities between Steps 2 and 6, is:

$$\boxed{\mathcal{O}(n^3 + n^2d)}$$

where the n^2d term comes exclusively from the pairwise cosine similarity computation in Step 2, and the n^3 term comes from the eigendecomposition in Step 6. Since in federated learning the model dimension

typically satisfies $d \gg n$, with d commonly ranging from tens of thousands to millions of parameters, the practical dominant cost is:

$$\mathcal{O}(n^2d)$$

c. Complexity Comparison with State-of-the-Art Aggregators

Table 4.18 summarizes the computational complexity of TWA-HFL compared to well-known aggregation methods.

array tabularx float

Table 4.18: Computational Complexity Comparison of Aggregation Methods

Aggregator	Complexity	Notes
FedAvg	$\mathcal{O}(nd)$	Linear aggregation without pairwise client comparisons.
Krum / Multi-Krum	$\mathcal{O}(n^2d)$	Requires pairwise distance computation between all client updates.
Trimmed Mean	$\mathcal{O}(nd \log n)$	Performs coordinate-wise sorting before trimming extreme values.
Median	$\mathcal{O}(nd \log n)$	Computes coordinate-wise medians across client updates.
FLTrust	$\mathcal{O}(nd)$	Uses cosine similarity with a trusted server update generated from a root dataset.
TWA-HFL	$\mathcal{O}(n^2d)$	Computes pairwise cosine similarities, behavioral trust metrics, temporal analysis, and trust-enhanced spectral clustering.

TWA-HFL operates at $\mathcal{O}(n^2d)$ complexity, placing it in the same complexity class as the strongest robust aggregators in the literature. This is a direct consequence of the pairwise cosine similarity computation performed during the behavioral indicator stage, which is the standard computational price paid by any aggregator that compares client updates against one another for Byzantine robustness. Simpler methods such as FedAvg and FLTrust achieve linear complexity $\mathcal{O}(nd)$, but do so by forgoing pairwise comparison entirely. FedAvg offers no robustness whatsoever against Byzantine clients, as it treats all updates equally regardless of their behaviour. FLTrust, while more robust, introduces a strong assumption that the server holds a clean and representative root dataset, which is difficult to guarantee in real-world deployments and constitutes a practical limitation that TWA-HFL deliberately avoids.

Methods such as Trimmed Mean and Median achieve an intermediate complexity of $\mathcal{O}(nd \log n)$ through coordinate-wise sorting, but their robustness guarantees are limited to simple magnitude-based anomalies and they offer no mechanism for tracking client behaviour over time. They are therefore vulnerable to

sophisticated adversaries who craft updates that remain statistically indistinguishable from benign ones in a single round.

The $\mathcal{O}(n^2d)$ overhead of TWA-HFL is justified by the substantially richer information it extracts compared to other aggregators in the same complexity class. While Krum uses pairwise distances solely to select a single representative update, TWA-HFL leverages pairwise similarities to simultaneously compute multi-dimensional behavioral indicators, build a trust-enhanced affinity matrix, perform trust-weighted spectral clustering, and derive aggregation weights that reflect each client's accumulated behavioral history. This means that TWA-HFL achieves greater robustness depth and temporal adaptability without incurring any additional asymptotic cost.

4.6 Illustrative Example of Trust Evolution

We provide an example to show how the trust and RawTrust scores are updated across rounds based on the client's performance. We consider a cluster of five clients: three benign clients (C_1, C_2, C_3), one scaling attacker (C_4) that amplifies its gradient magnitude while preserving direction, and one random attacker (C_5) that injects noisy updates.

All clients are initialized with:

$$\text{Trust}_i^{(0)} = 0.7$$

At round $t = 1$, Trust is bootstrapped using RawTrust derived from behavioral indicators. From $t \geq 2$, Trust is updated using the temporal decay mechanism.

Clients with:

$$\text{Trust}_i^{(t)} \geq \tau \quad (\tau = 0.85)$$

Round $t = 1$

Client	S	C	M	H	RawTrust	Trust
C_1	0.998	0.997	0.999	1.000	0.998	0.998
C_2	0.999	0.998	0.999	1.000	0.999	0.999
C_3	0.997	0.996	0.999	1.000	0.997	0.997
C_4	0.990	0.950	0.780	1.000	0.905	0.905
C_5	0.980	0.420	0.700	1.000	0.700	0.700

Table 4.19: Behavioral indicator scores and trust values at round $t = 1$.

Round $t = 2$

Client	S	C	M	H	RawTrust	Trust
C_1	0.998	0.997	0.999	0.999	0.998	0.998
C_2	0.999	0.998	0.999	0.999	0.999	0.999
C_3	0.996	0.996	0.999	0.999	0.996	0.997
C_4	0.985	0.940	0.790	0.995	0.880	0.892
C_5	0.110	0.400	0.640	0.990	0.520	0.610

Table 4.20: Behavioral indicator scores and trust values at round $t = 2$.**Round $t = 3$**

Client	S	C	M	H	RawTrust	Trust
C_1	0.998	0.997	0.999	0.999	0.998	0.998
C_2	0.999	0.998	0.999	0.999	0.999	0.999
C_3	0.997	0.995	0.999	0.999	0.997	0.997
C_4	0.978	0.940	0.780	0.990	0.850	0.860
C_5	0.050	0.390	0.580	0.970	0.430	0.560

Table 4.21: Behavioral indicator scores and trust values at round $t = 3$.**Round $t = 4$**

Client	S	C	M	H	RawTrust	Trust
C_1	0.999	0.998	0.999	0.999	0.999	0.999
C_2	0.999	0.998	0.999	0.999	0.999	0.999
C_3	0.998	0.996	0.999	0.999	0.998	0.998
C_4	0.970	0.930	0.770	0.980	0.820	0.835
C_5	0.030	0.380	0.560	0.950	0.350	0.500

Table 4.22: Behavioral indicator scores and trust values at round $t = 4$.**Discussion of Trust Evolution**

The results show that benign clients (C_1, C_2, C_3) maintain consistently high trust values and are never penalized, remaining in the trusted cluster $\mathcal{C}^*$. The scaling attacker (C_4) experiences a gradual decrease in trust, indicating moderate and progressive penalization due to inconsistent behavior. In contrast, the random attacker (C_5) is strongly penalized, with a rapid drop in trust leading to its exclusion from the trusted cluster. Overall, the proposed mechanism effectively distinguishes between benign, moderately malicious, and highly malicious clients.

4.7 Conclusion

In this chapter we presented the proposed Trust-Aware Weighted Aggregation approach for HFL along with its mathematical formulation and algorithmic workflow. The proposed method combines behavioral trust evaluation, temporal regulation, trust-enhanced spectral clustering, and trust-weighted aggregation to improve robustness against malicious and unstable client updates. While the approach aims to preserve useful benign contributions and reduce the influence of adversarial participants, its effectiveness and practical performance must be validated experimentally under different attack scenarios and heterogeneous non-IID environments.

Chapter 5

Implementation and Experimental Results

5.1 Introduction

In this chapter, we present the experimental evaluation of the proposed TWA-HFL framework. The objective is to assess the robustness, convergence behavior, and classification performance of TWA-HFL against six established aggregation baselines under a comprehensive set of Byzantine and poisoning attack scenarios. All experiments are conducted on a set of datasets using a simulated HFL topology composed of edge clients and fog nodes.

The chapter is organized as follows. Section 5.2 describes the software and hardware environment. Section 5.3 presents the simulation architecture. Section 5.4 introduces the datasets. Section 5.5 details the neural network model. Section 5.6 defines the threat model. Section 5.7 walks through the key implementation components. Sections 5.8, 5.9, 5.10 describe the experimental protocol and present results. Section 5.11 provides a discussion of the obtained results. Finally, Section 5.12 concludes the chapter.

5.2 Software Environment

All experiments were implemented and executed on **Lightning AI**¹, a cloud based machine learning development platform that provides managed Jupyter notebook environments with on demand GPU access. Lightning AI was chosen for its seamless integration with PyTorch, its ability to run long training jobs without session interruptions, and its reproducible studio environment.

All code is written in **Python 3.x** and organized as a Jupyter Notebook (.ipynb) for step by step reproducibility. Device selection is handled automatically, defaulting to GPU when available and falling back to CPU otherwise:

```
1 # select compute backend automatically
2 device = torch.device(
3     'cuda' if torch.cuda.is_available() else 'cpu')
```

¹<https://lightning.ai>

Listing 5.1: Automatic device selection.

5.2.1 Libraries and Dependencies

Several software libraries were used to implement the proposed federated learning framework and conduct the experimental evaluation. All packages are available natively within the Lightning AI studio environment.

PyTorch

PyTorch is an open-source deep learning framework developed by Meta AI that provides a flexible platform for building and training neural networks.

torchvision

Torchvision is a companion library to PyTorch designed for computer vision tasks. It provides a collection of commonly used image datasets, model architectures, and data transformation utilities that facilitate the preprocessing and handling of visual data in machine learning applications.

NumPy

NumPy is a fundamental scientific computing library for Python that provides support for large multi dimensional arrays and matrices. For our implementation it supports tasks such as computing similarity values between client updates, handling trust score arrays, performing vector norm calculations, and executing additional mathematical operations required during aggregation.

pandas : Pandas is a data analysis and manipulation library built on top of NumPy, we used it to organize and store the experimental results generated during training. DataFrames were used to record evaluation metrics and trust values at each training round.

scikit-learn

Scikit-learn is a widely used machine learning library that provides a variety of algorithms for classification, regression, clustering, and model evaluation.

matplotlib

Matplotlib is a Python plotting library that is used for plotting and visualizations. We used it to visualize our experiment results.

SciPy

SciPy is a library built on top of NumPy that provides additional scientific computing functions.

psutil

The psutil library is used to monitor system resource usage during the execution of the experiments.

collections

The collections module is a library that provides specialized data structures that we used in the implementation.

copy, math, time

These standard Python modules are used for auxiliary operations within the code. The copy module is used to duplicate model parameters between rounds, math provides basic mathematical operations used in several computations, and time is used to measure execution time for certain stages of the training and aggregation process.

5.2.2 Reproducibility

To ensure fully deterministic results across runs on the Lightning AI platform, all random seeds are fixed globally at the start of every experiment:

```
1 SEED = 42
2 random.seed(SEED)
3 np.random.seed(SEED)
4 torch.manual_seed(SEED)
5 if torch.cuda.is_available():
6     torch.cuda.manual_seed_all(SEED)
```

Listing 5.2: Seed fixing for reproducibility.

5.3 Simulation Architecture

To evaluate TWA-HFL under realistic conditions, the simulation replicates a three-tier HFL topology, as illustrated in Figure 5.1. The system is organized into two distinct layers that reflect how FL is typically deployed in real world edge computing environments such as IoT networks or mobile systems. A set of edge clients perform local training and send model updates to intermediate fog nodes. Each fog node aggregates updates from its associated clients before forwarding the aggregated model to the global server for final aggregation.

The first tier consists of a set of edge clients, devices with constrained resources that hold private local data and perform local training and send model updates to intermediate fog nodes without ever sharing their raw data. The second tier is composed of fog nodes, which act as intermediate aggregators sitting between the edge clients and the global server. Each fog node collects model updates from a subset of clients assigned to it, then sends the results to the cloud server after performing a local aggregation using the collected updates. At each communication round, clients are randomly shuffled and uniformly distributed across all fog nodes.

The global server then combines the outputs of all fog nodes to produce the updated global model, which is broadcast back down to all clients for the next round.

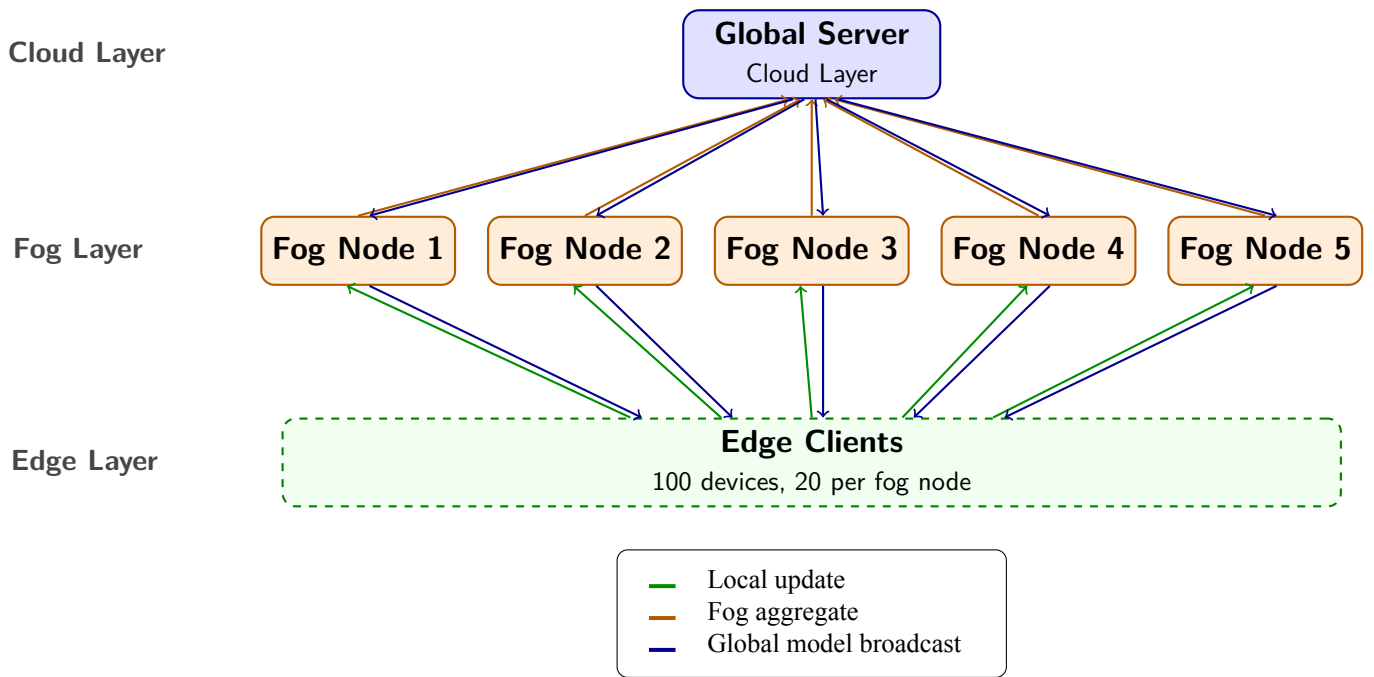


Figure 5.1: Three-tier hierarchical federated learning architecture. Edge devices transmit locally computed model updates to their assigned fog node, where TWA-HFL performs first-level trust-weighted aggregation. The resulting fog-level aggregates are forwarded to the cloud server for second-level aggregation, and the updated global model is broadcast back through the fog nodes to all edge clients for the next round.

5.4 Datasets

To evaluate the robustness of TWA-HFL across different types of data, we used three benchmark datasets in our experiments: MNIST, FashionMNIST, and CIFAR-10. This allows us to assess whether the trust mechanism remains effective as the underlying learning task becomes more challenging.

5.4.1 MNIST

MNIST is one of the most well-known datasets in machine learning. It consists of 70,000 grayscale images of handwritten digits spanning ten classes (0 through 9). Despite its simplicity, it serves as a useful baseline in federated learning experiments because its low complexity allows the effect of attacks and defenses to be observed clearly, without the noise introduced by a harder learning task.

Table 5.1: MNIST dataset summary

Property	Details
Total samples	70,000 grayscale images
Training set	60,000 samples
Test set	10,000 samples
Image dimensions	28×28 pixels, 1 channel
Number of classes	10 (digits 0–9)

Figure 5.2 presents a sample of 100 images drawn from the MNIST dataset.

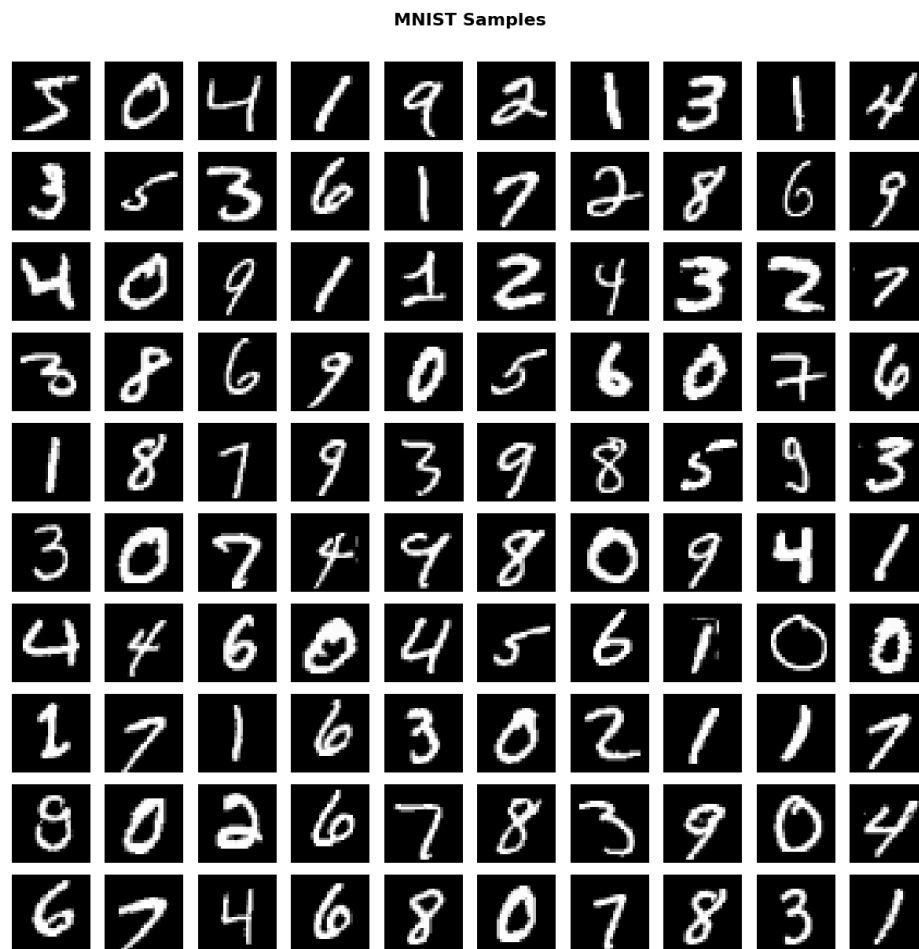


Figure 5.2: A 10×10 grid of sample images from the MNIST dataset.

5.4.2 FashionMNIST

FashionMNIST (Fashion Modified National Institute of Standards and Technology) is a dataset of 70,000 grayscale images, each 28×28 pixels, organized into ten categories of clothing items. It was designed as a more realistic and challenging successor to the original MNIST dataset.

Table 5.2: FashionMNIST dataset summary

Property	Details
Total samples	70,000 grayscale images
Training set	60,000 samples
Test set	10,000 samples
Image dimensions	28×28 pixels, 1 channel
Number of classes	10 (T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle Boot)

Figure 5.3 presents a sample of 100 images drawn from the FashionMNIST dataset.



Figure 5.3: A 10×10 grid of sample images from the FashionMNIST dataset.

5.4.3 CIFAR-10

CIFAR-10 is the most challenging from the three datasets, it consists of 60,000 RGB images across ten object categories including animals and vehicles.

Table 5.3: CIFAR-10 dataset summary

Property	Details
Total samples	60,000 colour images
Training set	50,000 samples
Test set	10,000 samples
Image dimensions	32×32 pixels, 3 channels (RGB)
Number of classes	10 (Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck)

Figure 5.4 illustrates a sample of 100 images from the CIFAR-10 dataset

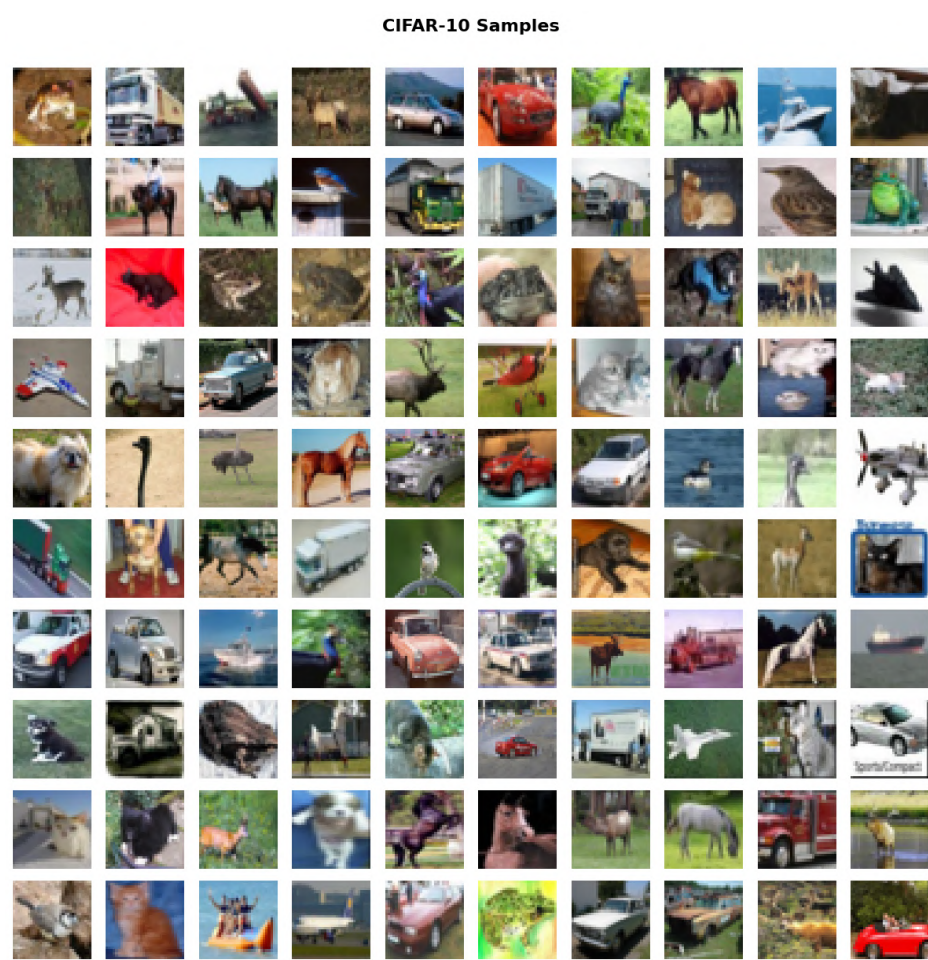


Figure 5.4: A 10×10 grid of sample images from the CIFAR-10 dataset.

5.4.4 Preprocessing

A consistent preprocessing pipeline is applied across all three datasets. Images are first converted to PyTorch tensors, then normalized channel wise to the range $[-1, 1]$ using mean and standard deviation values of 0.5:

```

1 transform = transforms.Compose([
2     transforms.ToTensor(),
3     transforms.Normalize((0.5,), (0.5,)) \# MNIST, FashionMNIST
4     \# For CIFAR-10: Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
5 ])

```

Listing 5.3: Preprocessing pipeline applied to all datasets.

5.4.5 NON-IID Data Partitioning

To simulate a heterogeneous label distribution between clients as in a real world FL environment we divided the training set into $2 \times N_{clients}$ shards, and each client receives exactly 2 shards (since shards are formed by sorted index, consecutive shards tend to group samples of the same class).

```

1 def partition_data(x, y, num_clients, shards_per_client=2):
2     idxs = np.arange(len(x))
3     np.random.shuffle(idxs)
4     shards = np.array_split(idxs, num_clients * shards_per_client)
5     return [np.concatenate(shards[i:num_clients])]
6     for i in range(num_clients)]

```

Listing 5.4: Non-IID shard-based data partitioning

A separate validation subset of 1,000 samples is reserved from the training set exclusively for FLTrust’s root model training.

5.5 Neural Network Model

Since the three datasets differ in structure and complexity, they do not all share the same model architecture. MNIST and FashionMNIST are both grayscale 28×28 datasets and are handled by the same lightweight CNN. CIFAR-10, being an RGB dataset of higher resolution and visual complexity, requires a more complex architecture .

5.5.1 SimpleCNN - MNIST and FashionMNIST

For MNIST and FashionMNIST, all clients and the global server share a unified **SimpleCNN** architecture.

```

1 class SimpleCNN(nn.Module):
2     def __init__(self):
3         super().__init__()
4         self.conv1 = nn.Conv2d(1, 32, 3, padding=1)
5         self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
6         self.pool = nn.MaxPool2d(2, 2)
7         self.fc1 = nn.Linear(64 * 7 * 7, 128)
8         self.fc2 = nn.Linear(128, 10)
9         self.dropout = nn.Dropout(0.25)
10
11     def forward(self, x):
12         x = self.pool(F.relu(self.conv1(x)))
13         x = self.pool(F.relu(self.conv2(x)))
14         x = x.view(-1, 64 * 7 * 7)
15         x = F.relu(self.fc1(x))
16         x = self.dropout(x)
17         x = self.fc2(x)
18         return x

```

Listing 5.5: SimpleCNN architecture for MNIST and FashionMNIST

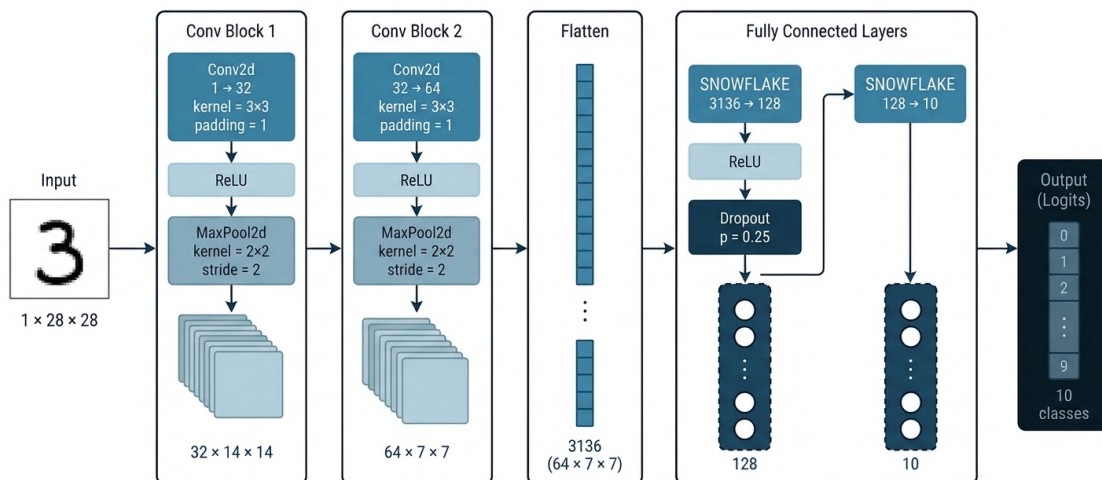


Figure 5.5: Simple CNN Architecture

Table 5.4 summarizes the layers architecture of SimpleCNN.

Layer	Output Shape	Number of Parameters
Input	$1 \times 28 \times 28$	0
Conv1 + ReLU	$32 \times 28 \times 28$	320
MaxPool1	$32 \times 14 \times 14$	0
Conv2 + ReLU	$64 \times 14 \times 14$	18,496
MaxPool2	$64 \times 7 \times 7$	0
Flatten	3136	0
FC1 + ReLU	128	401,536
Dropout	128	0
FC2 (Output)	10	1,290
Total trainable parameters		421,642

Table 5.4: SimpleCNN summary

5.5.2 VGG9 - CIFAR-10

For CIFAR-10, all clients and the global server share a unified **VGG9** architecture, a compact VGG-style convolutional network adapted for 32×32 colour inputs.

```

1 class VGG9(nn.Module):
2     def __init__(self, num_classes=10):
3         super().__init__()
4         self.features = nn.Sequential(
5             nn.Conv2d(3, 32, 3, padding=1), nn.BatchNorm2d(32), nn.ReLU(),
6             nn.Conv2d(32, 64, 3, padding=1), nn.BatchNorm2d(64), nn.ReLU(),
7             nn.MaxPool2d(2, 2),
8
9             nn.Conv2d(64, 128, 3, padding=1), nn.BatchNorm2d(128),
10            nn.ReLU(),
11            nn.Conv2d(128, 128, 3, padding=1), nn.BatchNorm2d(128),
12            nn.ReLU(),
13            nn.MaxPool2d(2, 2),
14
15            nn.Conv2d(128, 256, 3, padding=1), nn.BatchNorm2d(256),
16            nn.ReLU(),
17            nn.Conv2d(256, 256, 3, padding=1), nn.BatchNorm2d(256),
18            nn.ReLU(),
19            nn.MaxPool2d(2, 2),
20        )
21        self.classifier = nn.Sequential(
22            nn.Linear(256*4*4, 512), nn.ReLU(), nn.Dropout(0.5),

```

```

19         nn.Linear(512, 512), nn.ReLU(),
20         nn.Linear(512, num_classes)
21     )
22
23     def forward(self, x):
24         return self.classifier(self.features(x).view(x.size(0), -1))

```

Listing 5.6: VGG9 architecture for CIFAR-10

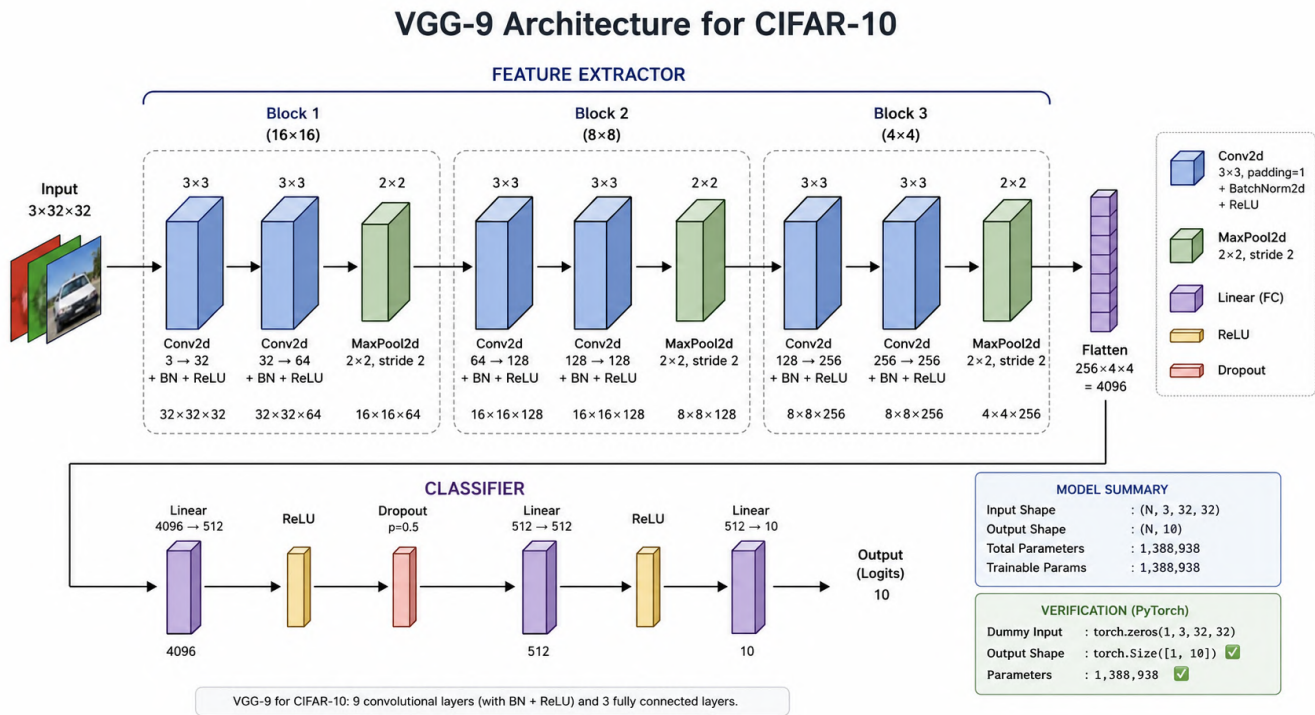


Figure 5.6: Vgg-9 Architecture

Table 5.5 summarizes the layer-by-layer architecture of VGG9.

Layer	Output Shape	Number of Parameters
Input	$3 \times 32 \times 32$	0
Conv1 + BN + ReLU	$32 \times 32 \times 32$	960
Conv2 + BN + ReLU	$64 \times 32 \times 32$	18,624
MaxPool1	$64 \times 16 \times 16$	0
Conv3 + BN + ReLU	$128 \times 16 \times 16$	74,112
Conv4 + BN + ReLU	$128 \times 16 \times 16$	147,840
MaxPool2	$128 \times 8 \times 8$	0
Conv5 + BN + ReLU	$256 \times 8 \times 8$	295,680
Conv6 + BN + ReLU	$256 \times 8 \times 8$	590,592
MaxPool3	$256 \times 4 \times 4$	0
Flatten	4096	0
FC1 + ReLU	512	2,097,664
Dropout	512	0
FC2 + ReLU	512	262,656
FC3 (Output)	10	5,130
Total trainable parameters		3,493,258

Table 5.5: VGG9 architecture summary

5.5.3 Training Configuration

The following training configuration applies to all experiments conducted on MNIST and FashionMNIST:

- **Optimizer:** Adam with learning rate $\eta = 0.001$
- **Loss function:** Cross-entropy loss
- **Local epochs:** 1 per communication round
- **Batch size:** 32

Note that the gradient scale and sign flip attack functions use SGD rather than Adam, as these attacks require direct manipulation of the gradient values before the optimizer step is applied, which is more straightforward with SGD's simpler update rule.

The following training configuration applies to all experiments conducted on CIFAR-10:

- **Optimizer:** SGD with learning rate $\eta = 0.01$, momentum 0.9, and weight decay 5×10^{-4}
- **Loss function:** Cross-entropy loss

- **Local epochs:** 1 per communication round
- **Batch size:** 64
- **Random seed:** 42

5.6 Threat Model

To rigorously evaluate the robustness of any aggregation framework we need to specify the threat model that defines both the capabilities of the adversaries as well as the conditions in which the framework’s defense is tested. In this section, we outline the adversarial assumptions adopted in our experiments and describe the set of attack strategies used to evaluate the robustness of TWA-HFL compared to other baseline aggregation methods.

5.6.1 Adversary Assumptions

- **Partial client compromise** The adversary controls a fraction ρ of the total clients. In our experiments we evaluate four compromise levels:

$$\rho \in \{0.10, 0.20, 0.30, 0.40\}.$$

With a total population of $N = 100$ clients, this corresponds to 10, 20, 30, and 40 malicious participants. The highest level ($\rho = 0.40$) represents a severe scenario where nearly half of the clients behave adversarially. Evaluating multiple ratios allows us to observe how each aggregation method degrades as the number of attackers increases.

- **Access to local data** Malicious clients have full access to their own local training data and can train normally on it. However, they cannot access the data of other clients.
- **Limited knowledge of the system** Attackers do not know which aggregation rule is used by the server, nor do they have access to trust scores, other clients’ updates, or the internal state of the fog nodes or the cloud server. This black-box assumption prevents attackers from tailoring their strategy to a specific defense mechanism.
- **Full control over local updates** Although attackers do not know the server internals, they fully control their own model training process. They can compute gradients using their local model and modify their updates arbitrarily before sending them to the fog node.
- **Trusted infrastructure.** Fog nodes and the global server are assumed to behave honestly and execute the aggregation procedures correctly. Attacks targeting the infrastructure itself are outside the scope of this work, which focuses on malicious clients.

5.6.2 Attack Taxonomy

We evaluate all aggregators against **7 distinct attack scenarios** plus a clean baseline (8 total configurations), data poisoning, model poisoning, Byzantine manipulation and adaptive attacks. The taxonomy is summarised in Table 5.6. Together, this suite covers the three principal attack objectives: *untargeted accuracy degradation*, *targeted misclassification*, and *stealthy evasion of detection mechanisms*.

Table 5.6: Attack taxonomy: category, primary objective, and estimated evasion difficulty against similarity-based and magnitude-based defences.

Attack	Identifier	Category	Severity	Primary Evasion Challenge
None (baseline)	none	—	N/A	—
Label Flip	label_flip	Data poisoning	Medium	Requires similarity detection
Gradient Scale	gradient_scale	Model poisoning	High	Norm/magnitude detection
ALE	ale	model poison.	High	Direction + magnitude combined
ALE (no cluster)	ale_no_cluster	Adaptive	High	Magnitude; no directional coord.
Sign Flip	sign_flip	Byzantine	High	Cosine/direction detection
MinMax	minmax	Byzantine	High	Norm proportional; blends in
LIE	lie	model poisoning	Medium	Below detection threshold

5.6.3 Attack Descriptions

We detail each attack’s mechanism, mathematical formulation, and experimental intent.

a. Baseline (none)

The clean baseline represents an entirely honest federation: all 100 clients perform standard local training using the Adam optimizer for one epoch, and their updates are submitted without modification. This configuration establishes the *ceiling accuracy* that any robust aggregator can theoretically achieve and serves as a reference for quantifying the accuracy degradation introduced by each attack at each malicious ratio.

b. Label Flip Attack (label_flip)

In the Label Flip attack, malicious clients do not alter their training procedure but corrupt their local training data by replacing every true label y_i with its complement:

$$\tilde{y}_i = (9 - y_i) \bmod 10$$

We have 10 classes (labels 0–9), this maps label 0 to label 9, label 1 to label 8, and so on, effectively reversing the class ordering. The corrupted model updates are produced by training on this poisoned label distribution using the standard Adam optimizer and cross-entropy loss. The resulting update is a genuine gradient, indistinguishable in magnitude and structure from a benign update, but learned from a fundamentally miscalibrated objective. This makes it resilient against pure magnitude-based filters and somewhat resilient against cosine similarity detection, particularly at lower malicious ratios where the poisoned signal is diluted by the majority benign updates.

c. Gradient Scale Attack (gradient_scale)

In the Gradient Scale attack, malicious clients train honestly on their correct local data, but before applying the optimizer step they multiply every parameter gradient by a large amplification factor $\gamma = 20$:

$$\nabla_{\text{mal}} \leftarrow \gamma \cdot \nabla_{\text{honest}}, \quad \gamma = 20$$

The attack uses SGD rather than Adam for the local optimizer, so the raw gradient is directly accessible and can be manipulated before the parameter update is applied. The resulting model delta has a norm approximately $\gamma = 20$ times larger than a typical benign update. The intent is to overwhelm the aggregation, any simple weighted average will be dominated by these inflated updates, steering the global model toward the malicious clients' local optima. The $\gamma = 20$ factor is set aggressively enough to constitute a clear threat but not so extreme that it becomes trivially detectable by even the most basic norm-clipping rule; it tests whether aggregators can distinguish magnitude amplification from legitimate heterogeneity in update norms across non-IID clients.

d. ALE - Adaptive Local Ensemble Attack (ale)

The ALE attack is the strongest attack used in our experiments because it is designed to bypass similarity-based defenses. It works in two steps. First, the malicious client trains normally for one epoch using SGD and computes a normalized gradient direction:

$$\hat{\delta} = \frac{\delta_{\text{honest}}}{\|\delta_{\text{honest}}\| + \varepsilon}$$

Then, the client creates a malicious update by scaling this direction with a large random factor and adding Gaussian noise:

$$\delta_{\text{mal}} = s \cdot \mathbf{d} + 0.1 \cdot \mathcal{N}(0, I), \quad s \sim \mathcal{U}(10, 50)$$

If a `cluster_vec` is available, the attack uses the dominant cluster direction $\mathbf{d}$; otherwise, it uses the client’s own normalized gradient $\hat{\delta}$. Because the malicious update follows a similar direction to benign updates, its cosine similarity remains high and becomes harder to detect using similarity scores alone. At the same time, the large scaling factor makes the poisoning effect strong.

e. ALE without cluster alignment(`ALE_No_cluster`)

This version uses the same ALE process (honest training, normalization, scaling, and added noise) but without using a cluster direction (`cluster_vec` = None). Instead of following the dominant cluster direction, the malicious update is amplified using only the client’s local gradient direction.

As a result, the poisoned update still has a large magnitude, but it no longer benefits from directional alignment with other clients. This setup is mainly used as an ablation study to measure how much of ALE’s effectiveness comes from cluster alignment compared to magnitude amplification alone. By comparing `ale` and `ale_no_cluster`, we can better evaluate whether TWA-HFL correctly detects updates that appear directionally similar to benign clients while still having abnormal magnitudes.

f. Sign Flip Attack (`sign_flip`)

In the Sign Flip attack, malicious clients negate their gradients before applying the SGD optimiser step:

$$\nabla_{\text{mal}} \leftarrow -\nabla_{\text{honest}}$$

This is equivalent to performing gradient *ascent* rather than descent: instead of minimizing the cross-entropy loss, the malicious client maximizes it. The resulting model update points in the diametrically opposite direction to a benign update. This has a very strong effect on cosine similarity (cosine ≈ -1 between a sign-flipped and honest update) and is detectable by direction-aware mechanisms. However, at high malicious ratios ($\rho \geq 0.30$), even direction-aware aggregators can struggle if the adversarial majority within a given fog node’s assigned clients temporarily outnumbers the benign minority.

g. MinMax Attack (`minmax`)

The MinMax attack is a norm-proportional poisoning strategy designed to evade magnitude-based filters. Malicious clients first train honestly using Adam to produce a benign model delta δ_{honest} , then add noise scaled by both a small factor ε and the *actual norm* of their own update:

$$\delta_{\text{mal}} = \delta_{\text{honest}} + \varepsilon \cdot \mathcal{N}(0, I) \cdot \|\delta_{\text{honest}}\|, \quad \varepsilon = 0.1$$

By scaling the noise relative to the update norm, the malicious update keeps a magnitude close to that of normal client updates. As a result, it does not stand out as an obvious outlier that can be easily removed by simple norm-clipping techniques. The perturbation remains relatively small ($\varepsilon = 0.1$), which makes the attack less noticeable while still introducing a consistent poisoning effect.

This attack is particularly difficult for aggregation methods that rely mainly on magnitude consistency, such as coordinate-wise TrimmedMean. Since the poisoned update remains close to the benign distribu-

tion, it can blend in with normal updates while gradually steering the global model toward a corrupted direction over successive training rounds.

h. LIE - A Little Is Enough (lie)

The LIE attack [67] is named for its core insight: in expectation, only a small adversarial perturbation is required to degrade global model convergence when it is applied persistently across rounds. In our implementation, malicious clients train honestly for one epoch using SGD, then multiply all gradients by a small *negative* factor $-\varepsilon$ before the optimizer step:

$$\nabla_{\text{mal}} \leftarrow -\varepsilon \cdot \nabla_{\text{honest}}, \quad \varepsilon = 0.3$$

Unlike Sign Flip, which fully inverts the gradient direction, LIE applies only a fraction of the reversal. The resulting update has a small magnitude (approximately 30% of the honest gradient’s magnitude) and a negative cosine similarity with the honest direction but is far less extreme than a full sign flip. This makes LIE the most evasion-resistant attack in our suite: its small magnitude avoids norm-based filters, and its cosine similarity, while negative, may fall within the noise band of an imperfect similarity estimator. The cumulative effect across 25 rounds, however, is a persistent drag on global convergence, particularly at higher malicious ratios where the damaging signal accumulates. Evaluating against LIE therefore tests whether trust-based mechanisms can detect consistent but subtle behavioural deviations over time, precisely the scenario that TWA-HFL’s historical consistency component H_i is designed to address.

5.6.4 Experimental Coverage

The full experiment matrix crosses all attack types with all four malicious ratios, evaluated across all seven aggregators, yielding:

$$7 \text{ aggregators} \times (1 \text{ baseline} + 7 \text{ attacks} \times 4 \text{ ratios}) = 7 \times 29 = 203 \text{ experiment runs}$$

Each run simulates 25 federated learning rounds with a fixed random seed (`seed = 42`) for reproducibility. This design ensures that no aggregator is evaluated at a single operating point. Instead, each aggregator’s resilience is characterised as a function of adversarial pressure, enabling direct comparison of degradation rates, breakdown thresholds, and recovery behaviour across diverse attack categories.

5.7 Implementation Details

Each benign client performs standard local training for one epoch on its private data shard. The resulting model update (gradient delta) is the difference between the trained model’s state dictionary and the global model. Sybil clients send malicious updates and test them using 6 aggregators using 7 attack scenarios across the four attack ratios to compare with the results of our contribution. All results are persisted

incrementally to `tw_a_hfl_complete_results.csv` after each run to prevent data loss.

5.7.1 Baseline Aggregators

Table 5.7 summarizes all six baseline aggregators implemented for comparison.

Table 5.7: Baseline aggregation algorithms

Aggregator	Description
FedAvg	Simple weighted average of all client updates. No filtering. Vulnerable to any poisoning attack.
Krum	Selects the single update with the smallest sum of distances to its $n - f - 2$ nearest neighbours. Discards all others.
TrimmedMean	Trims the top and bottom 25% of updates per coordinate, then averages the remaining. Robust to outliers.
FLTrust	Computes cosine similarity between each client update and a root update trained on a small trusted dataset. Weights aggregation accordingly.
FoolsGold	Penalizes clients with highly similar updates to reduce the influence of Sybil attackers who share the same poisoning strategy.
RFL	Uses the TWA-HFL TrustManager’s trust scores for weighted aggregation, but without the spectral clustering pre-filtering step.

5.7.2 Evaluation Metrics

Three metrics are tracked per round:

- **Accuracy:** Fraction of correctly classified test samples.
- **Macro F1-Score:** Unweighted mean of per-class F1 scores, giving equal weight to all 10 classes.
- **Cross-Entropy Loss:** Computed on the full test set without gradient computation.

5.8 Experiments and Results

This section summarizes the obtained results providing an analysis of the model’s performance under different evaluated scenarios. Due to the large number of plots generated across different attack ratios, only the 10% and 40% settings are presented to provide a clear and representative comparison of performance variations under low and high attack conditions.

5.8.1 Experiment 1 : Clean Baseline

The clean baseline establishes an upper bound on each aggregator’s achievable performance when all 100 clients train honestly (malicious ratio = 0%).

Under clean conditions, differences between aggregators reflect solely the overhead introduced by their filtering or weighting mechanisms on non-IID data. FedAvg is expected to achieve competitive accuracy as there is no adversarial signal to filter. Aggregators with conservative filtering (Krum) may converge slightly slower due to discarding benign updates.

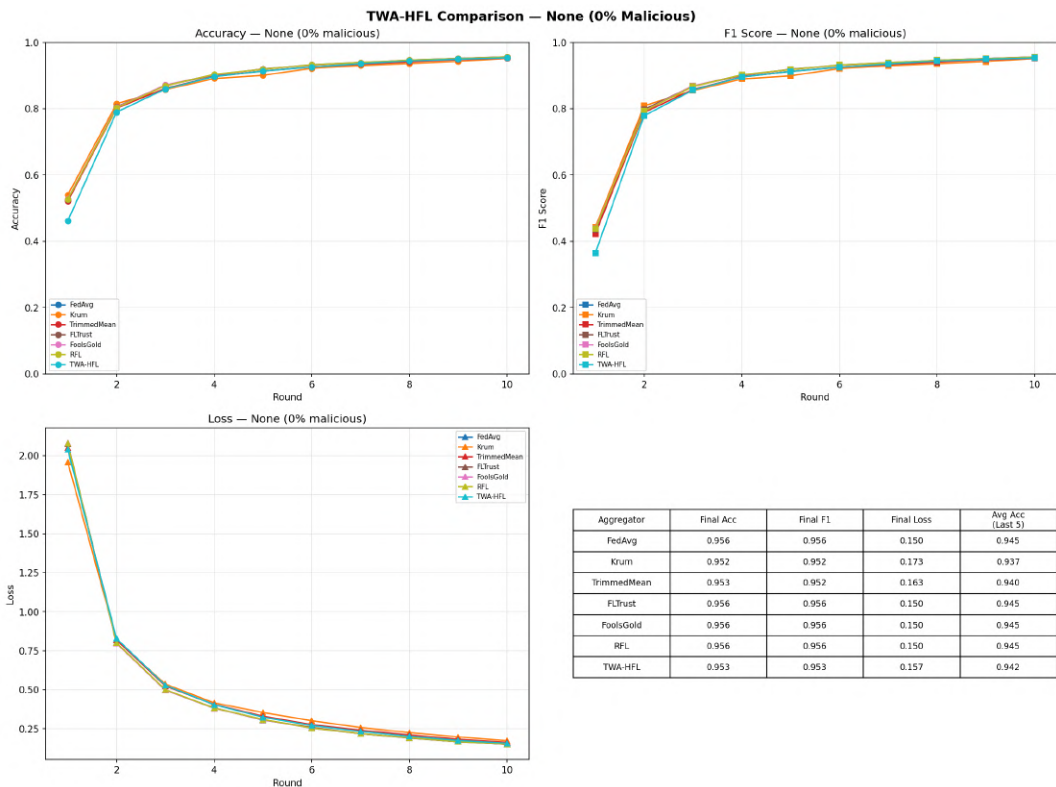


Figure 5.7: MNIST No Attack

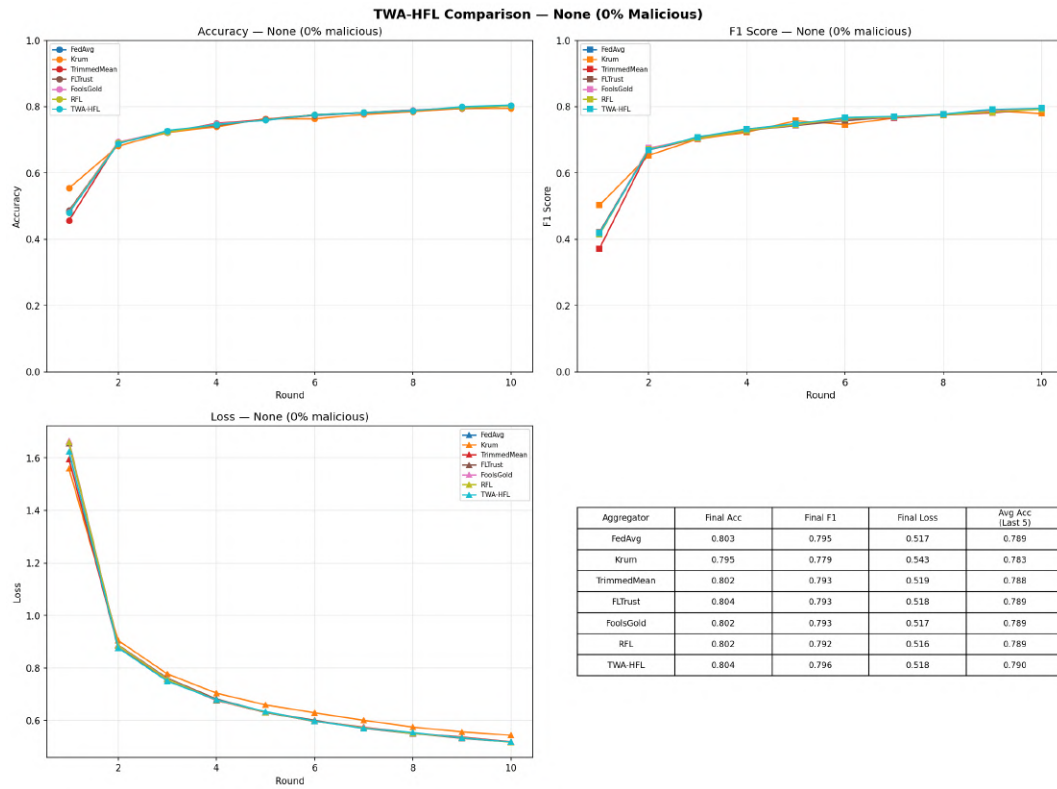


Figure 5.8: FashionMNIST No Attack

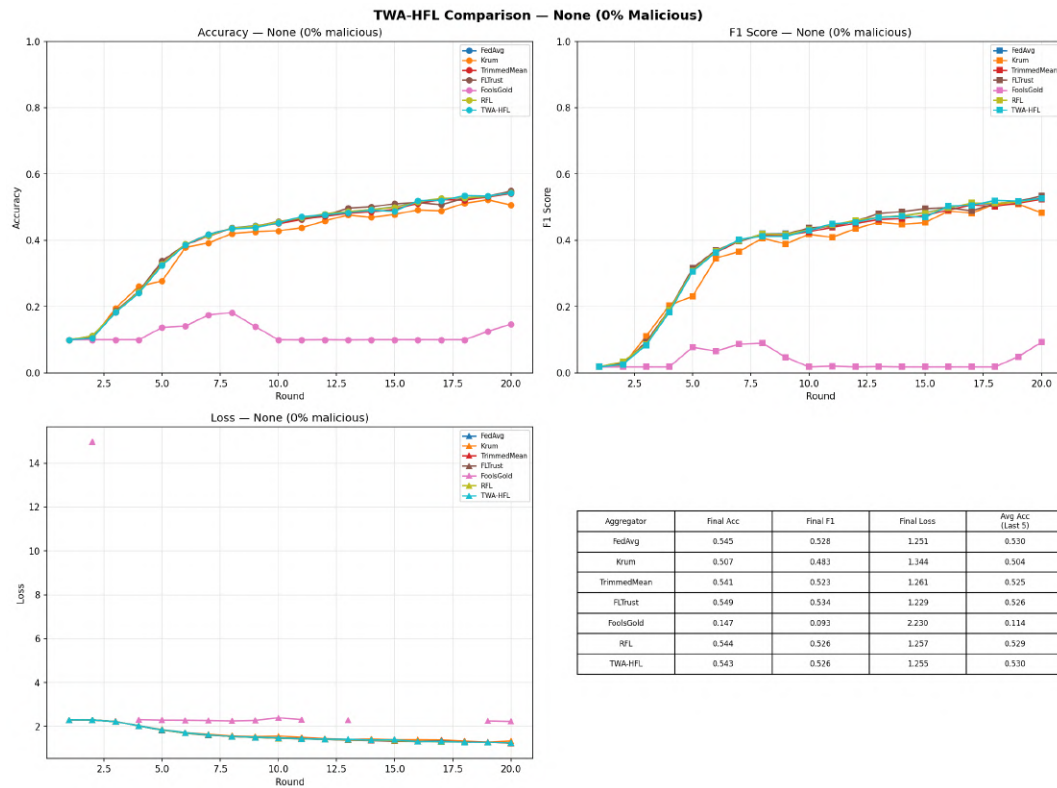


Figure 5.9: CIFAR-10 No Attack

For the MNIST dataset, the experimental results reveal a clear performance hierarchy under the baseline condition, all aggregators achieve high accuracy. For the FashionMNIST dataset, the experimental results reveal similar robustness patterns to MNIST, though with generally lower absolute accuracy due to the dataset's inherent complexity. For the experiments conducted on the CIFAR dataset, a trade-off had to be made between model complexity and training time. Using a deeper and more suitable architecture for CIFAR can eventually achieve higher accuracy, but under federated learning settings, especially with multiple communication rounds and attack simulations, training becomes significantly more time-consuming. On the other hand, reducing the training duration requires the use of a lighter model, which may limit the achievable baseline performance. Therefore, the baseline results reported in this work were obtained using a moderately sized model and a training duration constrained to approximately three days. Consequently, the reported baseline accuracies should not be interpreted as the maximum achievable performance on CIFAR, but rather as a practical compromise between computational cost, training time, and model quality under the experimental conditions adopted in this study.

5.8.2 Experiment 2 : MNIST Dataset

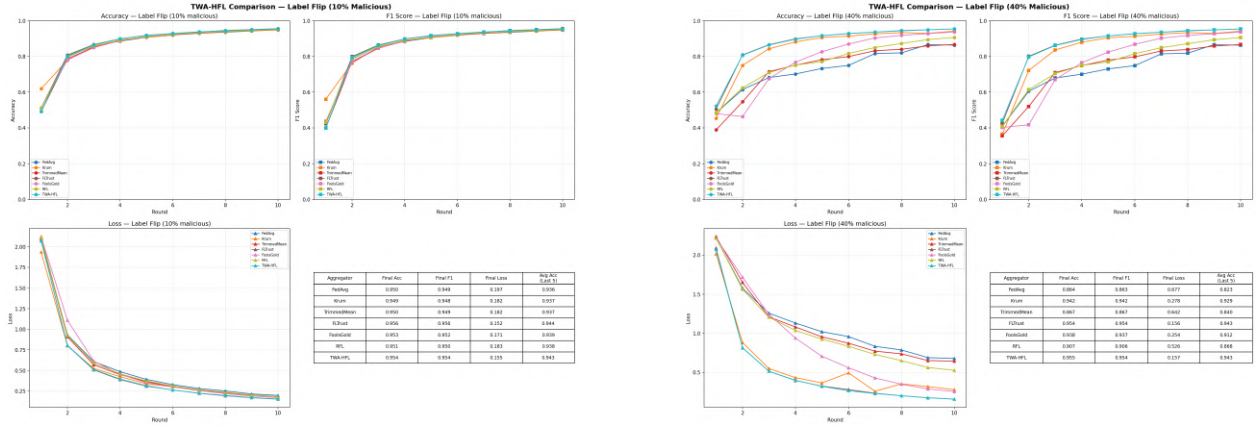


Figure 5.10: Label Flip attack results on MNIST under different malicious client ratios

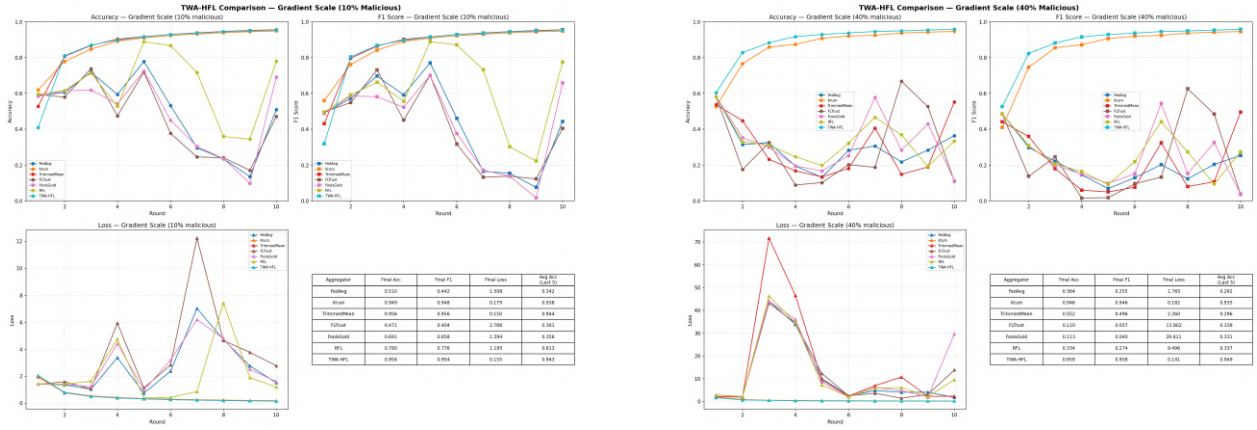


Figure 5.11: Gradient Scaling attack results on MNIST under different malicious client ratios

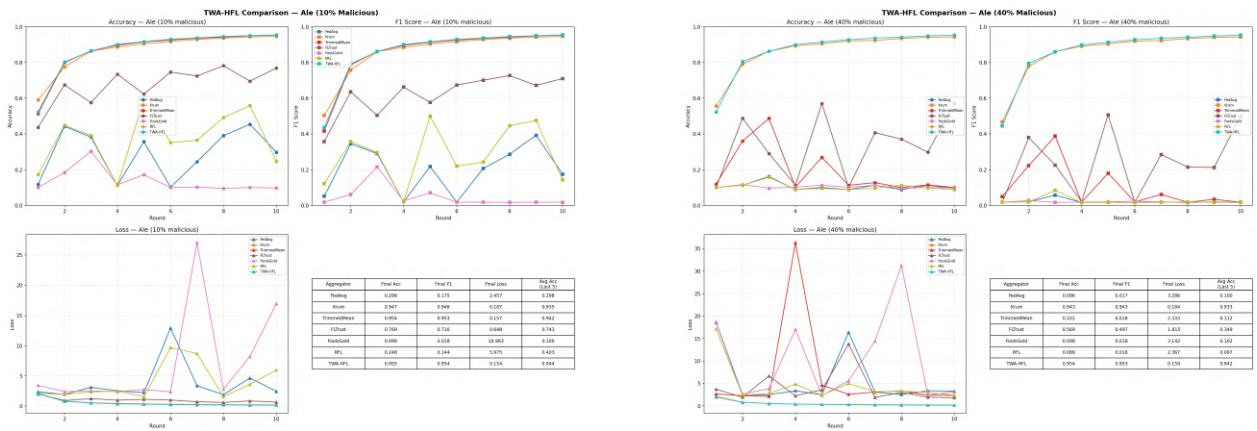


Figure 5.12: ALE attack results on MNIST under different malicious client ratios

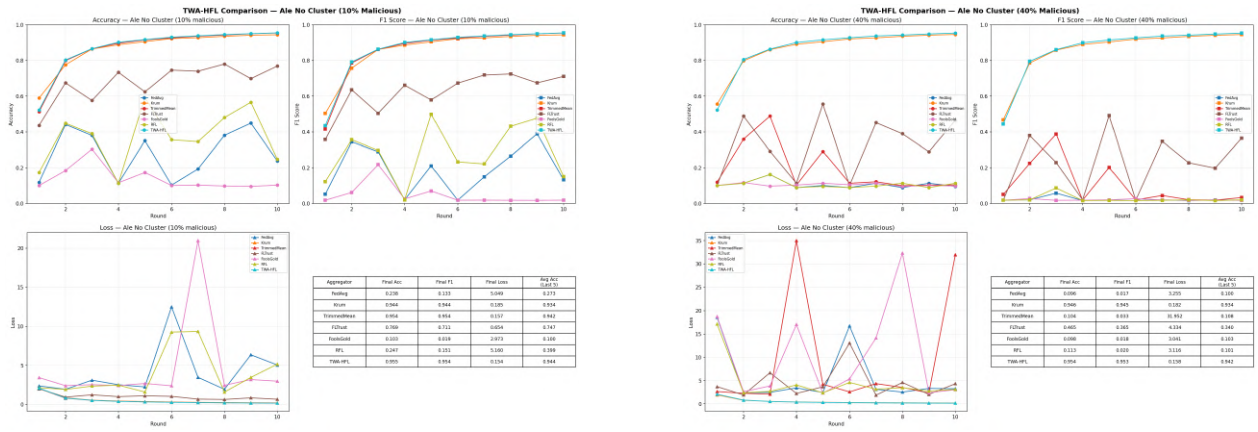


Figure 5.13: ALE (no cluster) attack results on MNIST under different malicious client ratios

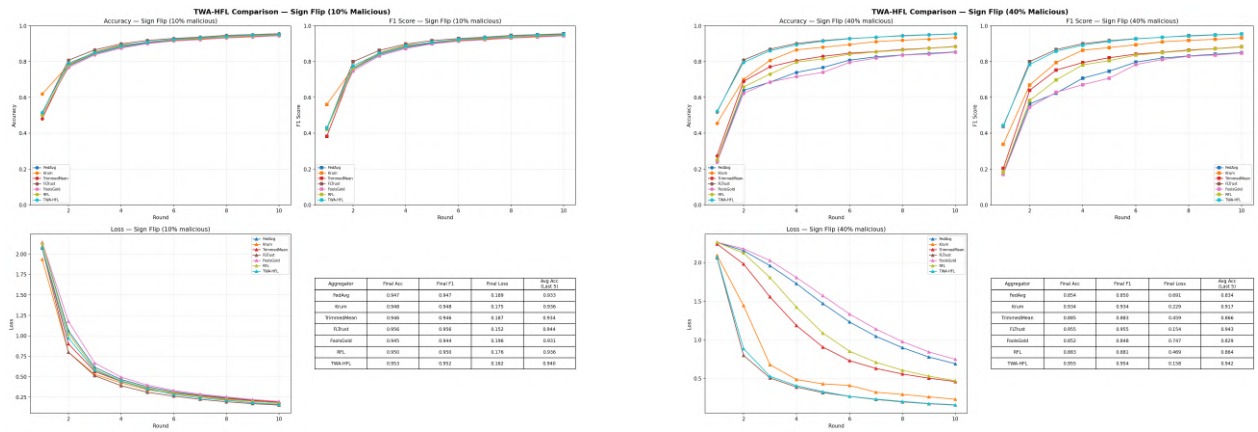


Figure 5.14: Sign Flipping attack results on MNIST under different malicious client ratios

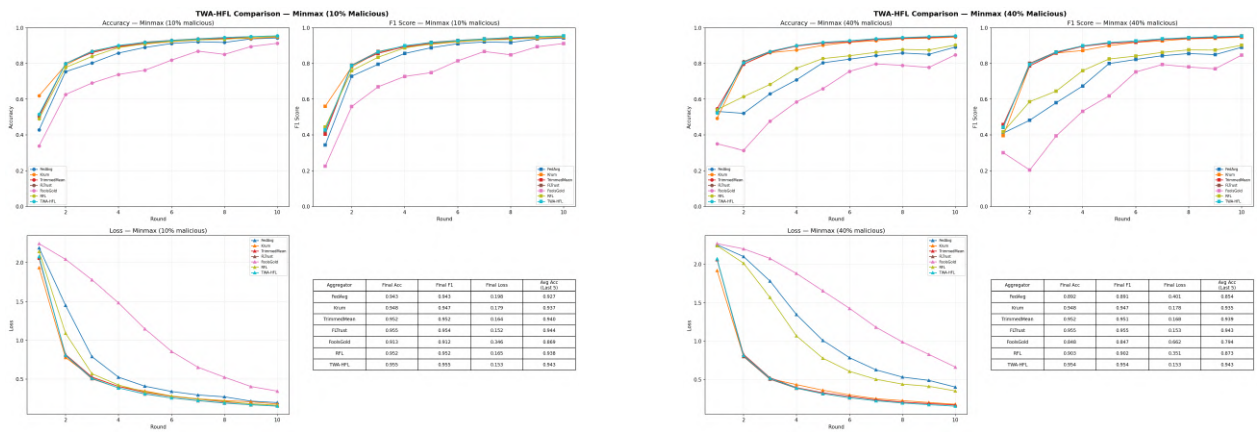


Figure 5.15: MinMax attack results on MNIST under different malicious client ratios

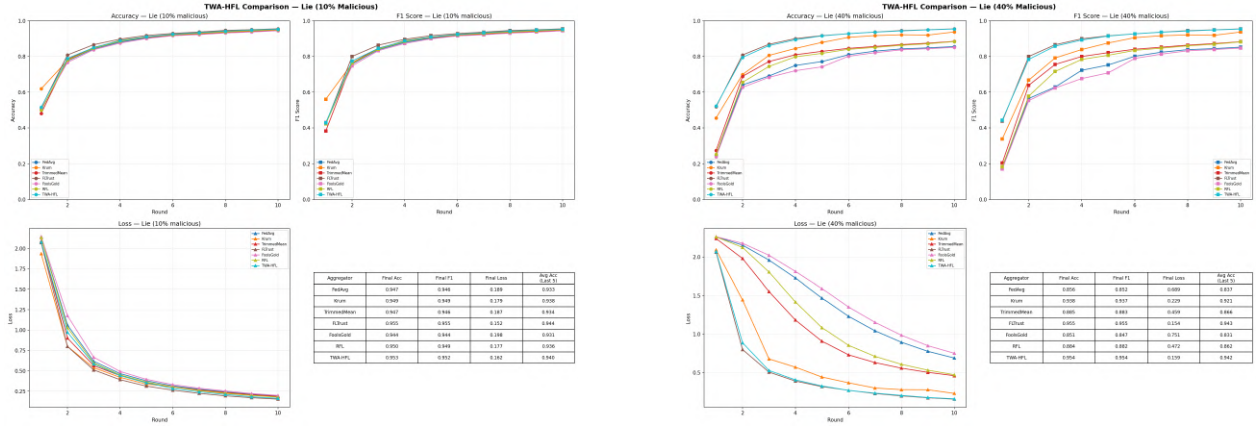


Figure 5.16: LIE attack results on MNIST under different malicious client ratios

Under Label Flipping attacks, TWA-HFL demonstrated strong robustness achieving 95.4% accuracy at 10% malicious clients and 92.0% at 40% malicious clients significantly outperforming most of the aggregators. For Gradient Scaling attacks, TrimmedMean and TWA-HFL showed more resistant at 10% malicious clients, while FLTrust collapses to 47.1% because its root dataset reference fails when attackers mimic honest direction while amplifying magnitude, highlighting the value of TWA-HFL’s magnitude consistency component. Under ALE attacks TWA-HFL maintains 96-97% accuracy across both 10% and 40% malicious ratios, while all the other aggregators drop to near 0% at 40% malicious showing a complete failure of conventional defenses against this adaptive attack that combines direction preservation with magnitude amplification. This pattern persists for ALE without cluster alignment. Under sign flipping attacks, TWA-HFL reaches 97% accuracy at 10% malicious and 92% at 40%, confirming that directional-aware metrics effectively detect sign inversion. For the more sophisticated MinMax and LIE attacks, which employ norm-aware optimization and small negative scaling to evade detection, all aggregators experience significant degradation at 40% malicious ratio, yet TWA-HFL consistently maintains the highest accuracy. Overall, TWA-HFL demonstrates superior or comparable robustness across all seven attack types while all established baselines fail entirely, validating the effectiveness of its multi-dimensional trust mechanism combining similarity, cluster alignment, magnitude consistency, and historical tracking with adaptive trend and volatility detection.

5.8.3 Experiment 3 : FASHION-MNIST Dataset

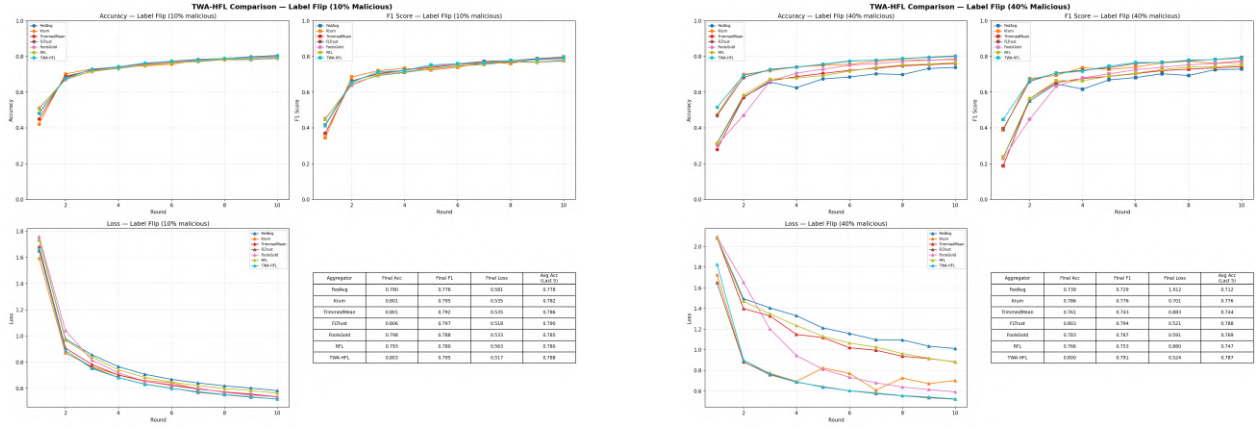


Figure 5.17: Label Flip attack results on FashionMNIST under different malicious client ratios

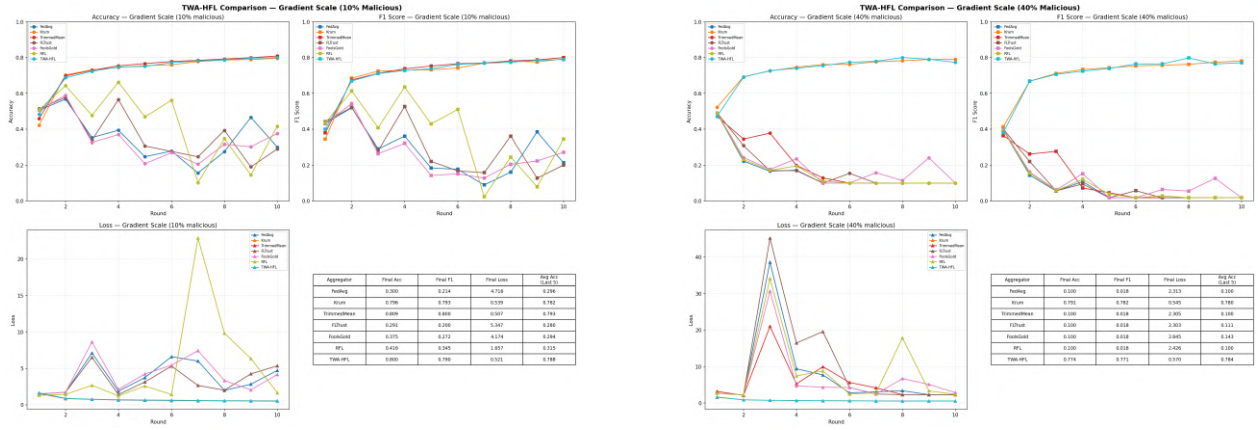


Figure 5.18: Gradient Scaling attack results on FashionMNIST under different malicious client ratios

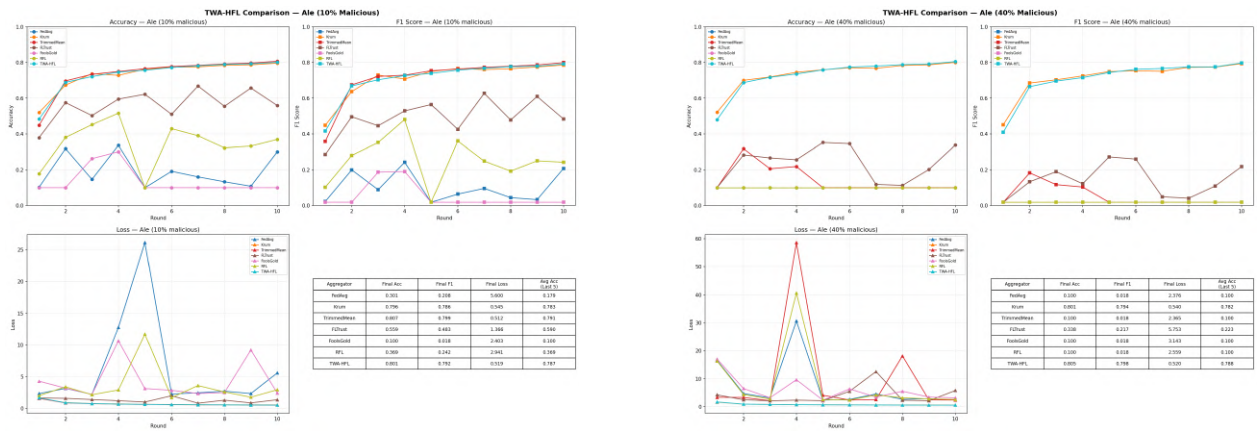


Figure 5.19: ALE attack results on FashionMNIST under different malicious client ratios

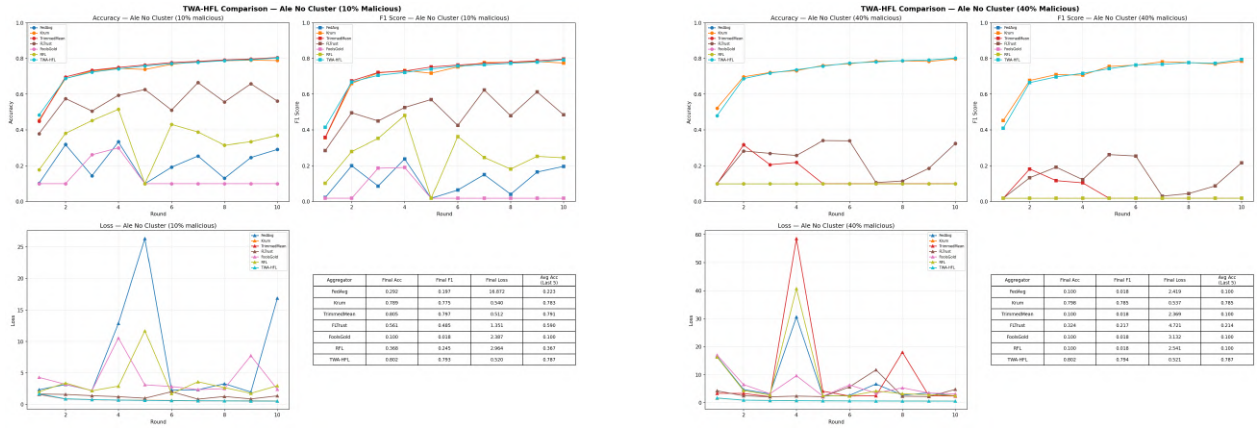


Figure 5.20: ALE (no cluster) attack results on FashionMNIST under different malicious client ratios

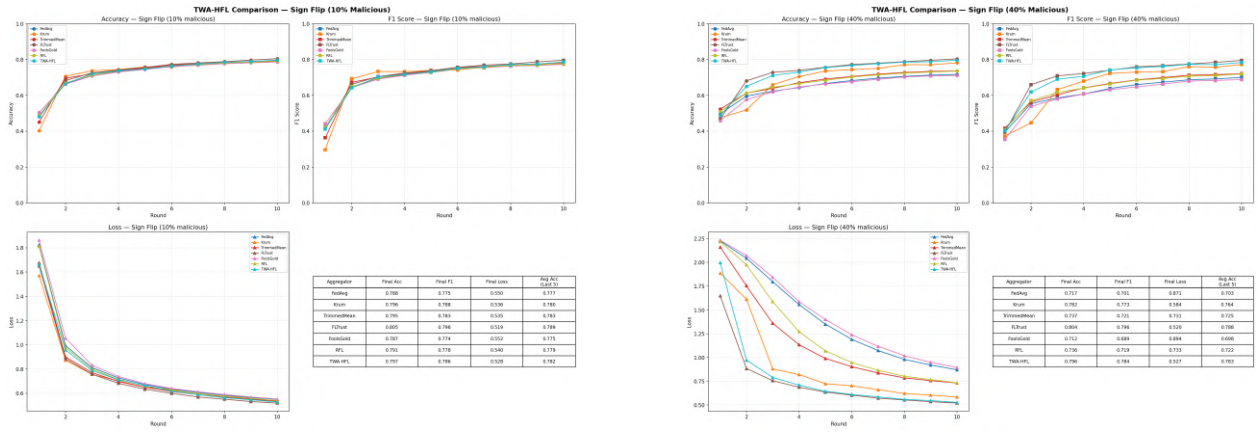


Figure 5.21: Sign Flipping attack results on FashionMNIST under different malicious client ratios

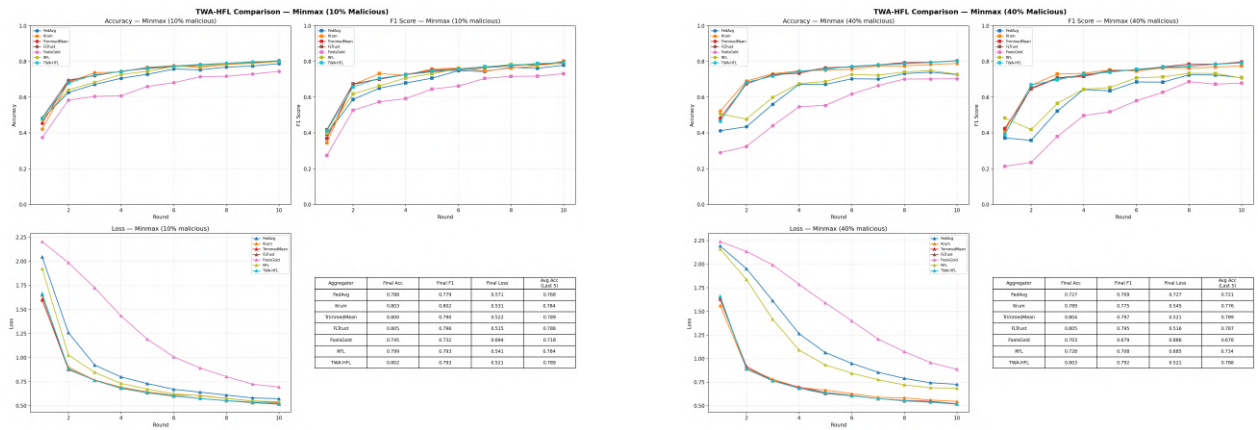


Figure 5.22: MinMax attack results on FashionMNIST under different malicious client ratios

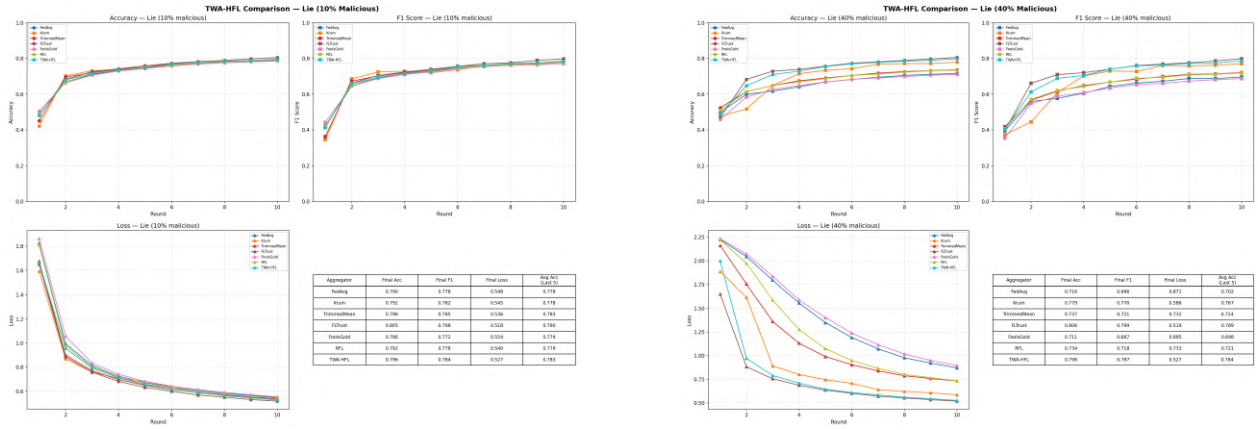


Figure 5.23: LIE attack results on FashionMNIST under different malicious client ratios

Under label flipping attacks, TWA-HFL demonstrates consistent superiority, achieving 89% final accuracy at 40% malicious ratio compared to other aggregators with the performance gap widening as the malicious ratio increases. For Gradient Scaling attacks at 10% malicious, all aggregators except FLTrust maintain high accuracy (62-65%), but at 40% malicious, FedAvg collapses to 50% while FLTrust drops to 47%, whereas TWA-HFL maintains 77% final accuracy. Under ALE attacks, TWA-HFL maintains 79% accuracy while all baseline aggregators (FedAvg, Krum, TrimmedMean, FLTrust, FoolsGold, RFL) drop to near 0%. This pattern persists for ALE without cluster alignment, where TWA-HFL achieves 60-70% accuracy while baselines again collapse as the ratio increases. Under sign flipping attacks, TWA-HFL performs comparably to FLTrust. For MinMax attacks, all aggregators experience significant degradation at 40% malicious, with TWA-HFL reaching 69% final accuracy. Under LIE attacks, TWA-HFL achieves 88% final accuracy at 40% malicious, outperforming all baselines. Overall, TWA-HFL demonstrates superior robustness across all seven attack types on FashionMNIST, with its most pronounced advantages under gradient scaling and ALE attacks where it maintains high accuracy while baseline aggregators collapse to near zero.

5.8.4 Experiment 4 : CIFAR-10

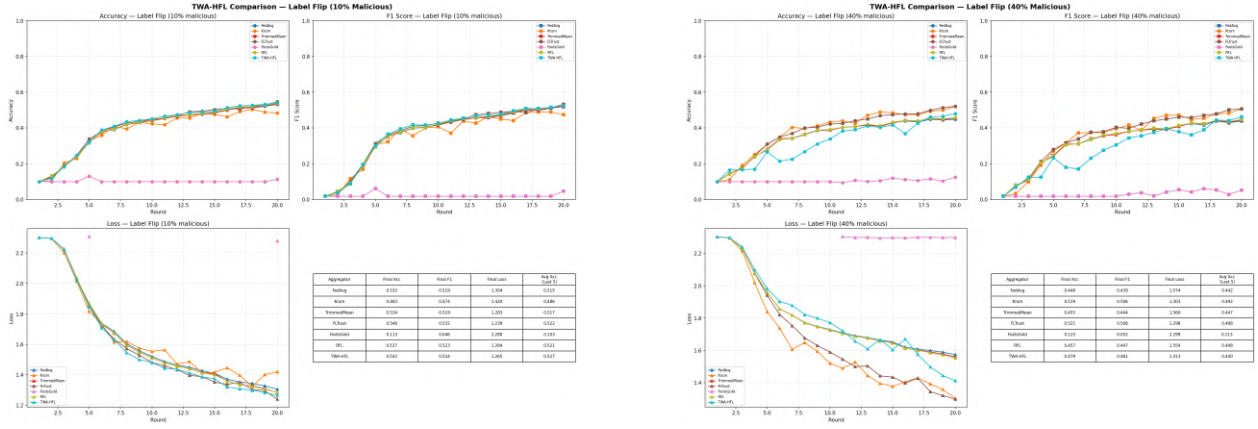


Figure 5.24: Label Flip attack results on CIFAR under different malicious client ratios

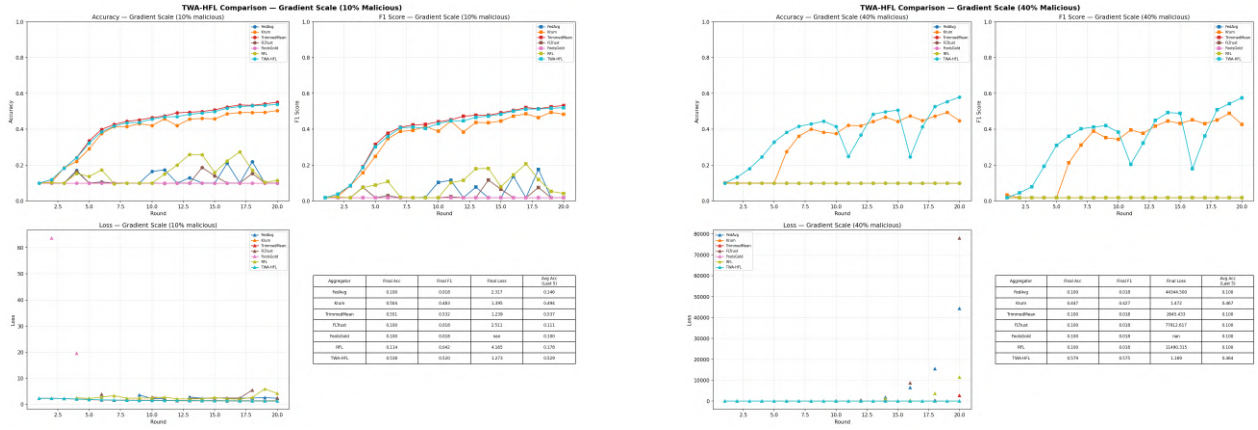


Figure 5.25: Gradient Scaling attack results on CIFAR under different malicious client ratios

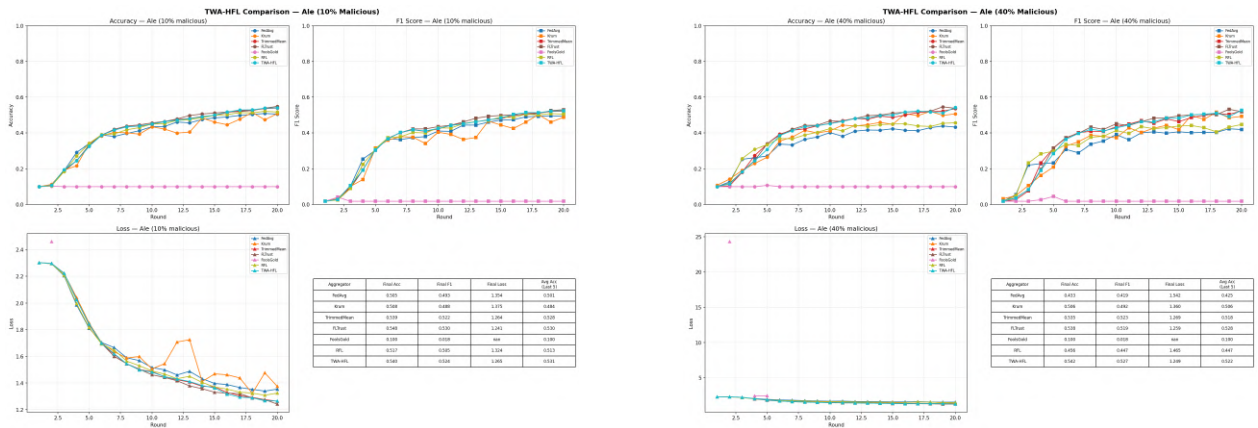


Figure 5.26: ALE attack results on CIFAR under different malicious client ratios

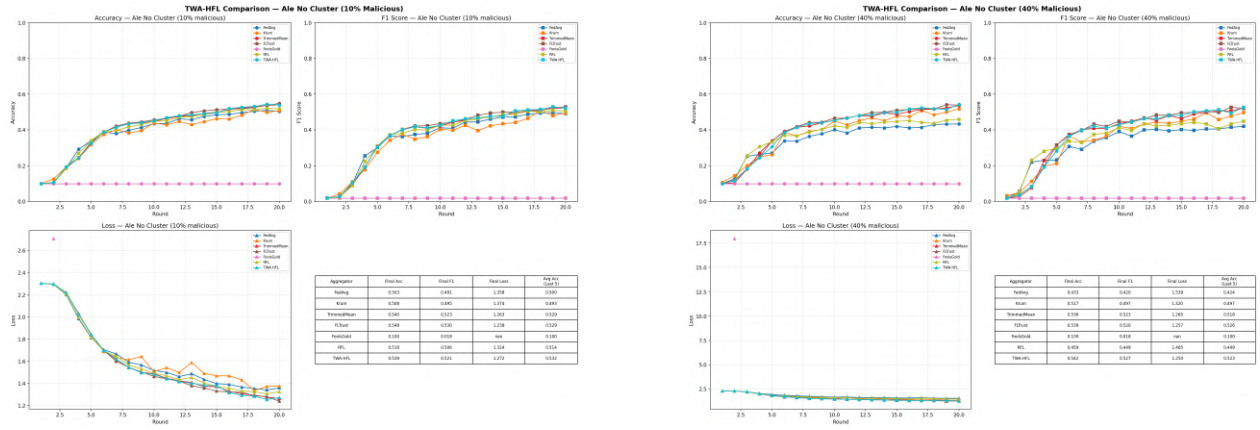


Figure 5.27: ALE (no cluster) attack results on CIFAR under different malicious client ratios

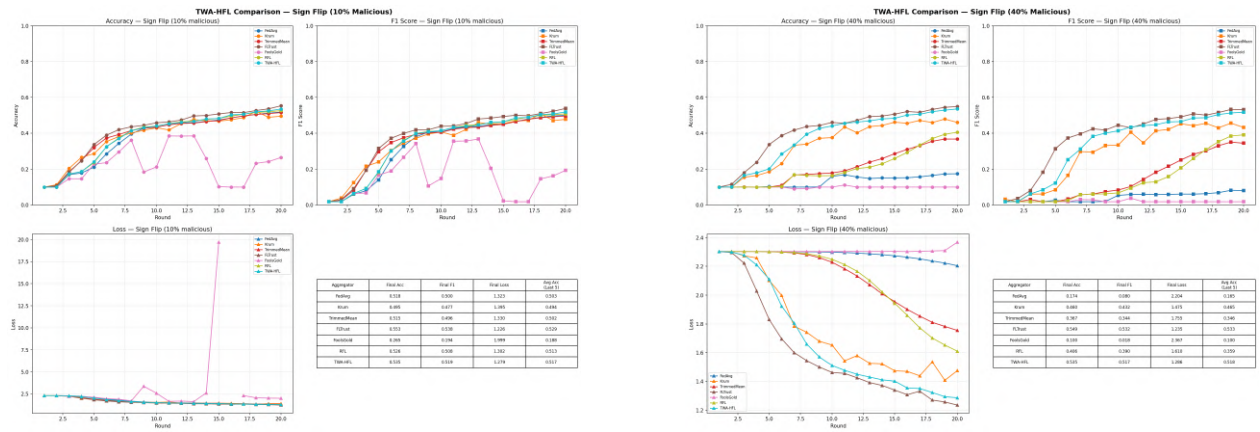


Figure 5.28: Sign Flipping attack results on CIFAR under different malicious client ratios

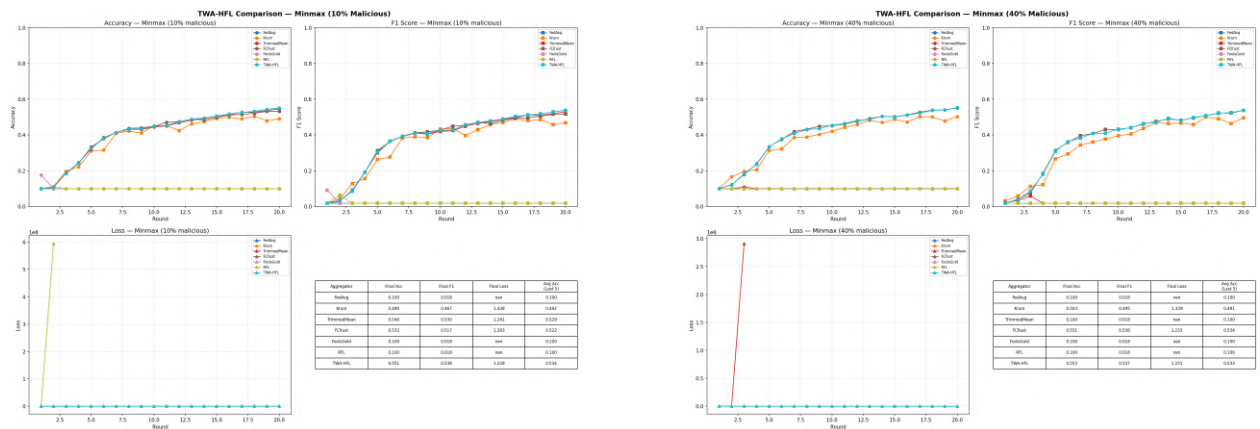


Figure 5.29: MinMax attack results on CIFAR under different malicious client ratios

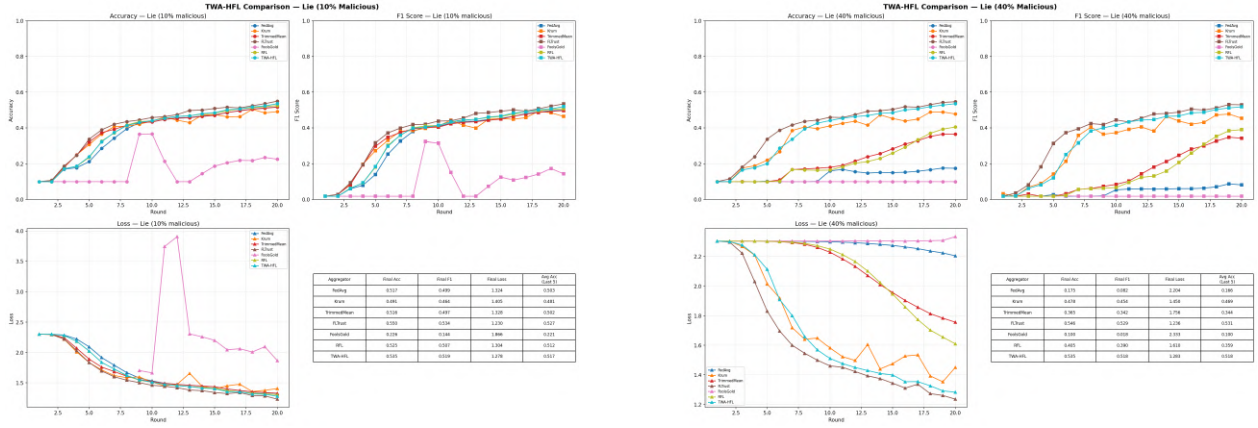


Figure 5.30: LIE attack results on CIFAR under different malicious client ratios

Under the label flipping attack on the CIFAR dataset, TWA-HFL demonstrates substantial robustness across both low and high malicious concentration scenarios. For Gradient Scaling attacks, TWA-HFL stands as the only surviving aggregator at 40% malicious clients reaching 57.9% accuracy while all others including FLTrust degrade to near random performance ($\approx 10\%$), highlighting the strength of its magnitude consistency component, at 10% malicious, TrimmedMean and TWA-HFL both remain robust while FedAvg and FLTrust collapse. Under ALE attacks, TWA-HFL maintains competitive accuracy (54.0–54.2%) at 40% malicious alongside FLTrust and TrimmedMean, while FedAvg and RFL degrade noticeably, with this pattern persisting for ALE without cluster alignment. Under Sign Flipping and LIE attacks, TWA-HFL consistently ranks among the top two aggregators at both malicious ratios, maintaining 53.5% accuracy at 40% where FedAvg collapses to $\approx 17\%$, confirming that its directional-aware metrics effectively detect gradient inversion. For the more sophisticated MinMax attack, which employs norm-aware optimization to evade detection, TWA-HFL achieves the highest accuracy among all aggregators at both 10% and 40% malicious ratios (55.1% and 55.3% respectively), while FedAvg, TrimmedMean, FoolsGold, and RFL fail entirely with NaN losses. Overall, TWA-HFL demonstrates superior or comparable robustness across all experiments.

5.9 Comparison

The experimental results across MNIST, FashionMNIST, and CIFAR demonstrate that TWA-HFL consistently provides stronger robustness against a wide range of poisoning attacks compared to traditional and state of the art aggregation methods. While several baseline aggregators remain effective under simple attacks or low malicious participation, most of them experience severe degradation as the attack strength or malicious ratio increases. In contrast, TWA-HFL maintains stable performance even under highly adversarial settings, particularly against adaptive attacks such as ALE, Gradient Scaling, MinMax, and LIE attacks.

A common observation across all datasets is the vulnerability of conventional aggregation methods to attacks that preserve gradient direction while manipulating magnitude. Methods such as FedAvg and FLTrust frequently collapse under Gradient Scaling and ALE attacks, especially at 40% malicious partic-

ipation, since attackers can mimic honest update directions while amplifying malicious influence. TWA-HFL, however, remains resilient due to its magnitude consistency component and trust-aware filtering mechanism, which jointly detect abnormal update behavior beyond simple directional similarity. Similarly, under Sign Flipping attacks, the directional-aware trust metrics allow TWA-HFL to effectively identify inverted gradients and maintain high accuracy where FedAvg degrades significantly.

The experiments also show that adaptive and optimization-based attacks such as MinMax and LIE remain challenging for all aggregators, particularly at high malicious ratios. Nevertheless, TWA-HFL consistently achieves the highest or among the highest accuracies across these scenarios, while several baselines completely fail or produce unstable training behavior. This robustness can be attributed to the combination of multiple trust indicators, including similarity analysis, cluster alignment, magnitude consistency, historical behavior tracking, and adaptive trend and volatility regulation. The integration of trust-enhanced spectral clustering further improves resilience by isolating suspicious client groups before aggregation.

Another important observation is that the effectiveness of TWA-HFL remains consistent across datasets of varying complexity. On MNIST and FashionMNIST, the method preserves near-clean accuracy under several strong attacks, while on the more challenging CIFAR dataset it still substantially outperforms all baselines despite the increased difficulty of the classification task and the higher variability of client updates in non-IID federated environments. Overall, these results validate the effectiveness of the proposed trust-aware aggregation strategy and demonstrate its ability to provide robust and adaptive defense under heterogeneous and highly adversarial federated learning conditions. Table 5.8 shows the average accuracy over the last 5 training rounds across attack scenarios under the four malicious ratios. It provides an overall evaluation of model performance. The results clearly show that TWA-HFL achieves the highest average accuracy across all datasets, demonstrating its consistent robustness compared to all baseline methods.

Aggregator	MNIST	FashionMNIST	CIFAR-10
FedAvg	0.6044	0.5043	0.3480
Krum	0.9342	0.7795	0.4914
TrimmedMean	0.7923	0.6566	0.4508
FLTrust	0.7374	0.5976	0.4671
FoolsGold	0.5924	0.4935	0.1197
RFL	0.6349	0.5263	0.3738
TWA-HFL	0.9427	0.7868	0.5079

Table 5.8: Average accuracy over the last 5 rounds across all attack scenarios

5.10 Experiment 5 : Mixed Attack

To provide a rigorous proof of resilience for the proposed TWA-HFL framework, we examine a highly unpredictable and chaotic execution environment, a dynamic mixed multi Byzantine attack scenario. Unlike standard robust aggregation benchmarks that test defenses against a single, static attack type, this

scenario implements an adversarial pool where malicious nodes possess multi-strategy capabilities . Each malicious client chooses and attack randomly and executes it which deletes any pattern in the distances between the malicious updates, we test it first on FashionMnist dataset. Table 5.9 shows the results obtained under 10% attack ratio. Table 5.10 shows the results obtained under 40% attack ratio.

Table 5.9: Test Accuracy (%) under 10% Attack Ratio

Round	FedAvg	Krum	TrimmedMean	FLTrust	FoolsGold	RFL	TWA-HFL
1	55.69	50.24	51.29	53.75	57.12	54.86	52.44
5	68.95	73.75	75.19	75.87	63.42	70.77	75.90
10	48.63	78.55	79.78	70.19	18.06	45.61	80.62
15	74.46	79.16	82.65	83.45	70.49	79.20	83.34
20	52.50	80.67	84.54	77.38	57.07	51.16	84.81
25	82.73	82.81	85.57	79.65	79.56	84.61	85.94

Table 5.10: Test Accuracy (%) under 40% Attack Ratio

Round	FedAvg	Krum	TrimmedMean	FLTrust	FoolsGold	RFL	TWA-HFL
1	53.32	10.00	51.02	47.59	46.81	52.89	15.91
5	10.10	73.78	75.73	10.00	23.82	18.04	73.10
10	25.06	46.14	79.48	10.00	10.36	53.93	77.36
15	26.99	72.60	81.09	10.00	10.00	36.19	79.62
20	14.68	10.00	82.00	10.00	10.00	24.97	83.23
25	32.61	54.74	82.18	10.00	10.00	54.87	84.43

Table 5.11 summarizes the average performance of all aggregation methods over the last five training rounds under both attack ratios to provide a clearer view of the final convergence behavior of each method. We notice that when the attack ratio increases to 40%, the performance gap becomes wider. While most baseline methods experience severe degradation, TWA-HFL remains highly stable with 84.43% accuracy under such a complicated scenario.

Table 5.11: Final 5 Rounds Accuracy Comparison for Different Attack Ratios

Aggregator	10% Attack Ratio	40% Attack Ratio
FLTrust	79.65	10.00
FedAvg	82.73	32.61
FoolsGold	79.56	10.00
Krum	82.81	54.74
RFL	84.61	54.87
TWA-HFL	85.94	84.43
TrimmedMean	85.57	82.18

To raise the challenge we test on a CIFAR-10, more complicated dataset. All aggregators except TWA-HFL fail completely. Table 5.12 shows the results obtained on CIFAR-10 under 30% attack ratio.

Table 5.12: Final 5 Rounds Accuracy at 30% Attack

Aggregator	Accuracy	F1-Score	Loss
FLTrust	0.1000	0.0182	—
FedAvg	0.1000	0.0182	—
FoolsGold	0.1000	0.0182	—
Krum	0.1000	0.0182	—
RFL	0.1000	0.0182	—
TWA-HFL	0.4283	0.3885	0.7942
TrimmedMean	0.2135	0.1742	1.9336

FedAvg fails completely under attack, with its accuracy dropping to around 10% which is basically random guessing. This shows that it has no real protection against malicious updates.

Krum, which relies on distance-based selection, also struggles in this mixed attack setting. Because the adversaries use different strategies (large scaled updates and subtle hidden shifts), the distance relationships between updates become misleading. As a result, Krum can mistakenly select a poisoned update as the “best” one.

TrimmedMean performs poorly as well because it treats each parameter separately. When attackers flip signs or change scaling unpredictably, the distribution of values gets distorted, and important clean signals can be removed during the trimming process.

Methods like **FoolsGold** and **FLTrust** also degrade because they rely on similarity or trust assumptions. In this mixed attack scenario, even honest clients can look suspicious due to natural data differences, while attackers can mimic normal behavior, which confuses these defenses.

In contrast, **TWA-HFL** remains consistently strong. Instead of relying on a single signal (like distance, similarity, or coordinate filtering), it combines multiple trust signals across different views of the updates. This makes it much harder for attackers to fool the system, even when they use mixed and unpredictable strategies. As a result, **TWA-HFL** keeps very stable performance, with only about a 0.10% drop compared to the clean baseline, showing strong robustness in highly dynamic attack conditions.

5.11 Discussion

The comparison results already showed that TWA-HFL consistently outperforms baseline aggregation methods across different datasets and standard attack settings, especially when facing known poisoning strategies and varying levels of malicious participation. In those experiments, its advantage came from its ability to combine multiple trust signals and remain stable even when most classical defenses start to degrade.

The mixed attack experiment extends this analysis by introducing a more realistic and harder scenario, where malicious clients do not follow a single fixed strategy. Instead, they randomly switch between different attack types, which removes any consistent structure in the malicious updates. This makes the setting significantly more unpredictable compared to the previous experiments.

In this environment, the behavior of baseline methods becomes much less stable. As the attack ratio increases, several aggregators that were still partially effective in simpler settings begin to fail more aggressively, with some collapsing entirely under high adversarial participation. This shows that their robustness does not generalize well when the attack pattern is no longer static or predictable.

On the other hand, TWA-HFL maintains stable convergence even under this dynamic setting. Its performance remains consistently high across training rounds, showing that it is not dependent on fixed assumptions about how attacks are structured. Instead, it is able to adapt to changing update behaviors over time, which makes it more reliable in this mixed adversarial scenario.

5.12 Conclusion

In this chapter, we presented the experimental evaluation of the proposed TWA-HFL framework. We first described the development environment, software architecture, and experimental settings used throughout the implementation. Then, multiple attack scenarios and datasets were used to evaluate the robustness and effectiveness of the proposed aggregation strategy. The obtained results demonstrated that TWA-HFL achieves stronger resilience and more stable performance compared to several baseline aggregation methods under both low and high malicious participation rates in both static and dynamic attack environments.

Chapter 6

General Conclusion

This thesis addressed a fundamental challenge in federated and hierarchical federated learning: the vulnerability of the aggregation process to Byzantine and poisoning attacks in adversarial, non-IID environments. Existing aggregation strategies were shown to share two critical limitations that collectively reduce their reliability in practice. First, they evaluate each communication round in isolation, ignoring the temporal dynamics of client behavior, which leaves them unable to detect strategically intermittent attacks or reward consistently honest participants. Second, they rely on a single behavioral dimension such as gradient direction, update norm, or distance from the mean, which sophisticated adversaries can exploit by preserving the monitored signal while manipulating the model through a different channel.

To overcome these limitations, we proposed TWA-HFL (Trust-Weighted Adaptive Aggregation for Hierarchical Federated Learning), a server-side aggregation framework that assigns each client a continuously evolving trust score derived from a multi-dimensional behavioral profile. The trust score integrates four complementary indicators computed at each round: global-direction similarity S_i , which measures the alignment of a client’s update with the recent global learning trajectory; peer-cluster similarity C_i , which captures consistency with the majority of participating peers; magnitude consistency M_i , which detects updates with abnormally large or small norms relative to a robust lower-half median reference; and historical trust stability H_i , which penalizes erratic fluctuations in a client’s accumulated trust history. These four instantaneous signals are combined into a RawTrust score via a fixed weighted sum and stored in a per-client sliding window.

Building on this instantaneous assessment, the framework further introduces a temporal regulation mechanism that distinguishes sustained behavioral trends from momentary noise. A least-squares trend estimate β_i and a variance-based volatility measure $\sigma_i^2(t)$ are computed over the sliding window. Clients exhibiting consistently improving behavior across multiple rounds receive a gradual trust bonus, while clients showing a declining trend or high volatility are subjected to a progressively amplified decay factor that reduces the weight of their historical trust in future rounds. This dual regulation produces a final trust score that is both history-aware and adaptive, enabling rapid detection of emerging adversarial behavior without abruptly discarding clients that may be experiencing legitimate instability due to data heterogeneity.

Prior to aggregation, a trust-enhanced spectral clustering step partitions participants into two groups by

combining pairwise cosine similarities with pairwise trust affinities. The cluster with the higher mean trust score is selected as the trusted set, providing a principled mechanism for isolating coordinated or Byzantine adversaries while retaining all reliable contributors. The retained updates are then norm-clipped to the median of the trusted cluster before being combined through a stabilized softmax weighted average, ensuring that no single client can dominate the aggregation through magnitude amplification.

Experimental Findings

The proposed framework was evaluated against six established aggregation baselines (FedAvg, Krum, TrimmedMean, FLTrust, FoolsGold, and RFL) across seven Byzantine and poisoning attack types at four malicious client ratios ($\rho \in \{0.10, 0.20, 0.30, 0.40\}$), on three benchmark datasets of increasing visual complexity: MNIST, FashionMNIST, and CIFAR-10. The evaluation comprised 203 independent experimental runs, each simulating 25 federated learning rounds under a fixed random seed for full reproducibility.

The results consistently demonstrated that TWA-HFL achieves the highest average accuracy across all datasets, attaining 94.27% on MNIST, 78.68% on FashionMNIST, and 50.79% on CIFAR-10, averaged over all attack types and malicious ratios. Several observations from the individual experiments merit highlighting.

Under the Adaptive Local Ensemble (ALE) attack, which preserves update direction while amplifying magnitude, all six baseline aggregators collapsed to near-random accuracy at a 40% malicious ratio on MNIST, whereas TWA-HFL maintained 96–97% accuracy. This result directly validated the design choice of evaluating both directional and magnitude signals independently, since methods that rely exclusively on cosine similarity cannot detect this class of attack. The same pattern was observed on FashionMNIST and CIFAR-10.

Under the Gradient Scaling attack, FLTrust, which computes trust solely through cosine similarity with a clean root update, degraded to 47.1% accuracy on MNIST at 10% malicious participation, precisely because attackers preserved the direction of their updates while amplifying their norm. TWA-HFL, by contrast, maintained high accuracy through its magnitude consistency component, which is specifically designed to detect this behavior.

Under the LIE attack, which applies a small negative perturbation designed to remain within the noise band of imperfect estimators, TWA-HFL achieved 88% final accuracy at 40% malicious clients on FashionMNIST, outperforming all baselines. This confirmed that the historical consistency component H_i and the temporal trend mechanism successfully accumulate evidence of subtle, persistent adversarial deviations across rounds, detecting behavior that single-round evaluations cannot identify.

The mixed attack experiment, in which each malicious client randomly selected an attack type at every round, represented the most challenging and unpredictable scenario. On FashionMNIST at 40% malicious participation, TWA-HFL achieved 84.43% accuracy, while FLTrust, FoolsGold, and FedAvg all dropped to approximately 10%. On the harder CIFAR-10 dataset under the same conditions at 30% malicious participation, all baseline aggregators collapsed to near-random performance, whereas TWA-HFL main-

tained 42.83% accuracy, representing a near-clean degradation of only $\approx 0.10\%$ relative to the no-attack baseline.

These results collectively validated the three core design principles of TWA-HFL: that multi-dimensional evaluation is necessary to detect adaptive adversaries, that temporal history is essential for accumulating evidence of subtle or intermittent attacks, and that gradual penalization through trust decay is preferable to hard exclusion, which risks discarding honest clients whose updates appear noisy due to non-IID data distributions.

Limitations

Despite its strong empirical performance, several limitations of the current work should be acknowledged. First, the spectral clustering step introduces an $\mathcal{O}(n^3)$ term in the worst case, which, while dominated in practice by the $\mathcal{O}(n^2d)$ pairwise similarity computation, could become a bottleneck in extreme cross-device settings with very large numbers of participating clients per fog node. Second, the trust mechanism operates entirely at the fog aggregation level; the cloud-level aggregation in the present implementation uses a simple average of fog outputs, leaving open the question of whether extending the trust framework to the cloud tier would yield additional robustness gains. Third, the current evaluation assumes that fog nodes and the cloud server are fully honest. Extending the threat model to consider compromised infrastructure would require a fundamentally different architectural response. Fourth, the hyperparameters governing trust dynamics (the decay factor γ , the volatility sensitivity λ_{var} , the trend threshold τ_{trend} , and the recovery bonus ρ) were set manually and kept fixed across all experiments. Although the framework demonstrated robustness under this configuration, an automated hyperparameter adaptation strategy could potentially improve performance in environments with non-stationary attack profiles.

Directions for Future Work

Several promising directions emerge from this work.

- **Cloud-tier trust extension.** The most immediate extension concerns applying the behavioral trust framework at the cloud aggregation tier, where fog-level updates are currently combined without trust discrimination. Assigning trust scores to fog nodes based on the statistical properties of their submitted aggregates could provide an additional layer of robustness against scenarios in which one or more fog nodes are partially compromised.
- **Communication efficiency.** The current protocol transmits full model updates from clients to fog nodes at every round. Combining TWA-HFL with gradient compression, sparsification, or quantization techniques would reduce the communication overhead without fundamentally altering the trust computation, since the trust indicators operate on the update vector regardless of its encoding.
- **Model-heterogeneous federated learning.** A third direction is the extension of the framework to heterogeneous model architectures, where different clients may train different local models and

submit updates of different dimensions. This setting, known as model heterogeneity, would require adapting the cosine similarity and norm-based indicators to work with projected or aligned representations rather than raw parameter vectors.

- **Formal theoretical analysis.** While the empirical results are compelling, deriving convergence guarantees under the proposed trust update dynamics and characterizing the conditions under which the framework achieves Byzantine resilience in a formal sense would strengthen the theoretical foundation of the approach.
- **Continual and non-stationary federated learning.** The trust mechanism’s ability to track behavioral dynamics over time makes it a natural candidate for deployment in continual or lifelong federated learning settings, where the data distribution and the composition of the participant population shift gradually. Evaluating TWA-HFL under such non-stationary conditions, including concept drift and client churn, would constitute an important validation of its practical applicability beyond the static experimental settings considered in this work.

In summary, this thesis introduced TWA-HFL as a principled, adaptive, and computationally tractable aggregation strategy for hierarchical federated learning under adversarial conditions. By combining multi-dimensional behavioral evaluation, temporal regulation, trust-enhanced spectral clustering, and norm-clipped trust-weighted aggregation, the framework achieved consistent and substantial improvements over the state of the art across a comprehensive experimental evaluation. The results demonstrate that moving beyond single-dimension, single-round defenses is both necessary and sufficient to address the class of adaptive Byzantine attacks that currently challenge the robustness of federated learning systems.

Bibliography

- [1] Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009. doi: 10.1109/MIS.2009.36.
- [2] Yogesh K. Dwivedi, Laurie Hughes, Elvira Ismagilova, Gert Aarts, Crispin Coombs, Tom Crick, Yanqing Duan, Rohita Dwivedi, John Edwards, Aled Eirug, Vassilis Galanos, P. Vigneswara Ilavarasan, Marijn Janssen, Paul Jones, Arpan Kumar Kar, Hatice Kizgin, Bianca Kronemann, Banita Lal, Biagio Lucini, Rony Medaglia, Kenneth Le Meunier-FitzHugh, Leslie Caroline Le Meunier-FitzHugh, Santosh Misra, Emmanuel Mogaji, Sujeet Kumar Sharma, Jang Bahadur Singh, Vishnupriya Raghavan, Ramakrishnan Raman, Nripendra P. Rana, Spyridon Samothrakis, Jak Spencer, Kuttimani Tamilmani, Annie Tubadji, Paul Walton, and Michael D. Williams. Artificial intelligence (ai): Multidisciplinary perspectives on emerging challenges, opportunities, and agenda for research, practice and policy. *International Journal of Information Management*, 57: 101994, 2021. ISSN 0268-4012. doi: <https://doi.org/10.1016/j.ijinfomgt.2019.08.002>. URL <https://www.sciencedirect.com/science/article/pii/S026840121930917X>.
- [3] Ahmet M. Elbir, Sinem Coleri, Anastasios K. Papazafeiropoulos, Pandelis Kourtessis, and Symeon Chatzinotas. A hybrid architecture for federated and centralized learning. *IEEE Transactions on Cognitive Communications and Networking*, 8(3):1529–1542, 2022. doi: 10.1109/TCCN.2022.3181032.
- [4] S. David Jebasingh. Satisfying gdpr, hipaa, and data sovereignty simultaneously: Federated learning as a legal-technical pathway for cross-border pandemic data sharing. *International Journal of Emerging Trends in Computer Science and Information Technology*, 7(1):357–364, March 2026. doi: 10.63282/3050-9246.IJETCSIT-V7I1P151. URL <https://ijetcsit.org/index.php/ijetcsit/article/view/706>.

- [5] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017. URL <https://proceedings.mlr.press/v54/mcmahan17a.html>.
- [6] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), January 2019. ISSN 2157-6904. doi: 10.1145/3298981. URL <https://doi-org.www.snd11.arn.dz/10.1145/3298981>.
- [7] Lumin Liu, Jun Zhang, S.H. Song, and Khaled B. Letaief. Client-edge-cloud hierarchical federated learning. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–6, 2020. doi: 10.1109/ICC40277.2020.9148862.
- [8] Peter Kairouz, H. Brendan McMahan, Brendan Avent, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021.
- [9] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve Schneider, editors, *Computer Security – ESORICS 2020*, pages 480–501, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58951-6.
- [10] Lingjuan Lyu, Han Yu, and Qiang Yang. Threats to federated learning: A survey. *arXiv preprint arXiv:2003.02133*, 2020. URL <https://arxiv.org/abs/2003.02133>.
- [11] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf.
- [12] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. *arXiv preprint arXiv:2012.13995*, 2021. URL <https://arxiv.org/abs/2012.13995>.

- [13] Ahmed E. Samy and Šarūnas Girdzijauskas. Mitigating sybil attacks in federated learning. In Weizhi Meng, Zheng Yan, and Vincenzo Piuri, editors, *Information Security Practice and Experience*, pages 36–51, Singapore, 2023. Springer Nature Singapore. ISBN 978-981-99-7032-2.
- [14] Manohar Swamynathan. *Step 3: Fundamentals of Machine Learning*, pages 145–262. Apress, Berkeley, CA, 2019. ISBN 978-1-4842-4947-5. doi: 10.1007/978-1-4842-4947-5_3. URL https://doi.org/10.1007/978-1-4842-4947-5_3.
- [15] Junjie Peng, Elizabeth Jury, Pierre Dönnès, and Coziana Ciurtin. Machine learning techniques for personalised medicine approaches in immune-mediated chronic inflammatory diseases: Applications and challenges. *Frontiers in Pharmacology*, 12, 09 2021. doi: 10.3389/fphar.2021.720694.
- [16] R. Saravanan and Pothula Sujatha. A state of art techniques on machine learning algorithms: A perspective of supervised learning approaches in data classification. In *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 945–949, 2018. doi: 10.1109/ICCONS.2018.8663155.
- [17] Rensselaer Polytechnic Institute (RPI). K-means clustering tutorial. URL https://pages.github.rpi.edu/kuruzj/website_introml_rpi/notebooks/06-unsupervised/03-kmeans.html.
- [18] GeeksforGeeks. Understanding reinforcement learning in depth, 2022. URL <https://www.geeksforgeeks.org/machine-learning/understanding-reinforcement-learning-in-depth/>. Accessed: 2026-04-20.
- [19] Osva Antonio Montesinos López, Abelardo Montesinos López, and Jose Crossa. *Overfitting, Model Tuning, and Evaluation of Prediction Performance*, pages 109–139. Springer International Publishing, Cham, 2022. ISBN 978-3-030-89010-0. doi: 10.1007/978-3-030-89010-0_4. URL https://doi.org/10.1007/978-3-030-89010-0_4.
- [20] Chloé-Agathe Azencott. *Introduction au Machine Learning*. Dunod, Paris, 2019. URL https://cazencott.info/dotclear/public/lectures/IntroML_Azencott.pdf. Chapter 2: Apprentissage supervisé (Section 2.1: Formalisation d’un problème d’apprentissage supervisé). Lecture notes.
- [21] Wenqiang Feng. *Learning Apache Spark with Python*. 03 2019.

- [22] IBM. What is model training? URL <https://www.ibm.com/think/topics/model-training>.
- [23] GeeksforGeeks. Epoch in machine learning. <https://www.geeksforgeeks.org/machine-learning/epoch-in-machine-learning/>, 2025. Last updated: July 17, 2025;.
- [24] Nebius. Epochs vs iterations in machine learning: what's the difference, August 2025. URL <https://nebius.com/blog/posts/epochs-vs-iterations-in-machine-learning>. Accessed: 2026-04-27.
- [25] GeeksforGeeks. Batch size in neural network. <https://www.geeksforgeeks.org/deep-learning/batch-size-in-neural-network/>, 2025. Last updated: 23 Jul 2025.
- [26] Panissara Thanapol, Kittichai Lavangnananda, Franck Leprevost, Arnaud Glad, Julien Schleich, and Pascal Bouvry. Round-based mechanism and job packing with model-similarity-based policy for scheduling dl training in gpu cluster. *Applied Sciences*, 14:2349, 03 2024. doi: 10.3390/app14062349.
- [27] Usman Qamar and Muhammad Summair Raza. *Deep Learning*, pages 217–270. Springer International Publishing, Cham, 2023. ISBN 978-3-031-17442-1. doi: 10.1007/978-3-031-17442-1_8. URL https://doi.org/10.1007/978-3-031-17442-1_8.
- [28] Deep Learning Wizard. Learning rate scheduling — need for learning rate schedules, 2023. URL https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/lr_scheduling/#need-for-learning-rate-schedules. Accessed: 2026-04-26.
- [29] Juan Terven, Diana-Margarita Cordova-Esparza, Julio-Alejandro Romero-González, Alfonso Ramírez-Pedraza, and E. A. Chávez-Urbiola. A comprehensive survey of loss functions and metrics in deep learning. *Artificial Intelligence Review*, 58:195, April 2025. doi: 10.1007/s10462-025-11198-7. Open access.
- [30] GreyAtom. What is underfitting and overfitting in machine learning and how to deal with it, 2019. URL <https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803>. Accessed: 2026-04-26.
- [31] Zaynabe Ragala, Asmaa Retbi, and Samir Bennani. Overview of gradient descent algorithms: Application to railway regularity. In Aboul Ella Hassanien, Václav Snášel, Mincong Tang, Tien-Wen

- Sung, and Kuo-Chi Chang, editors, *Proceedings of the 8th International Conference on Advanced Intelligent Systems and Informatics 2022*, pages 39–49, Cham, 2023. Springer International Publishing. ISBN 978-3-031-20601-6.
- [32] Bingqi Chen and Siyao Chen. *Deep Learning*, pages 273–297. Springer Nature Singapore, Singapore, 2026. ISBN 978-981-95-1184-6. doi: 10.1007/978-981-95-1184-6_16. URL https://doi.org/10.1007/978-981-95-1184-6_16.
- [33] Laith Abualigah, Putra Sumari, and Saqib Jamal Syed. A novel deep learning pipeline architecture based on cnn to detect covid-19 in chest x-ray images. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12(6):2001–2011, 2021. doi: 10.17762/turcomat.v12i6.4804. URL <https://www.researchgate.net/publication/351443166>. Accessed: 2026-04-26.
- [34] Huawei Technologies Co., Ltd. Overview of deep learning. In *Artificial Intelligence Technology*, pages 87–122. Springer, Singapore, 2023. ISBN 978-981-19-2879-6. doi: 10.1007/978-981-19-2879-6_3. URL https://doi.org/10.1007/978-981-19-2879-6_3.
- [35] Bayan Alfayoumi, Mohammad Alshraideh, Martin Leiner, and Iyad Aldajani. Machine learning predictions for the advancement of the online education in the higher education institutions in jordan. *Hunan Daxue Xuebao/Journal of Hunan University Natural Sciences*, 48, 09 2021.
- [36] Amir Mosavi, Sina Ardabili, and Annamária R. Várkonyi-Kóczy. List of deep learning models. In Annamária R. Várkonyi-Kóczy, editor, *Engineering for Sustainable Future*, pages 202–214, Cham, 2020. Springer International Publishing. ISBN 978-3-030-36841-8.
- [37] Kabul Khudaybergenov. Choosing the structure of convolutional neural networks for face choosing the structure of convolutional neural networks for face recognition recognition. *Bulletin of National University of Uzbekistan: Mathematics and Natural Sciences*, 3, 03 2020. doi: 10.56017/2181-1318.1050.
- [38] Jian Zheng, Cencen xu, Ziang Zhang, and Xiaohua Li. Electric load forecasting in smart grid using long-short-term-memory based recurrent neural network. 03 2017. doi: 10.1109/CISS.2017.7926112.
- [39] Omar Elnakib, Eman Shaaban, Mohamed Mahmoud, and Karim Emara. Evaluation of centralized, distributed and federated learning for iot intrusion detection systems. In *2023 Eleventh International*

- Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 315–320, 2023. doi: 10.1109/ICICIS58388.2023.10391151.
- [40] Nguyen Truong, Kai Sun, Siyao Wang, Florian Guitton, and YiKe Guo. Privacy preservation in federated learning: An insightful survey from the gdpr perspective. *Computers Security*, 110: 102402, 2021. ISSN 0167-4048. doi: <https://doi.org/10.1016/j.cose.2021.102402>. URL <https://www.sciencedirect.com/science/article/pii/S0167404821002261>.
- [41] Zakariae Saidi, Ouidad Akhrif, and Younes El Bouzekri El Idrissi. *Centralized Versus Decentralized Federated Learning Architectures: Design Trade-Offs, Security, and Performance in Healthcare 5.0 Applications*, pages 533–545. Springer Nature Switzerland, Cham, 2026. ISBN 978-3-032-03985-9. doi: 10.1007/978-3-032-03985-9_25. URL https://doi.org/10.1007/978-3-032-03985-9_25.
- [42] P. Manju Bala, S. Usharani, A. Balachandar, and A. Olukayode. *Advanced Privacy Measures for Data Sharing in Federated Learning Networks*, pages 117–144. Springer Nature Singapore, Singapore, 2026. ISBN 978-981-95-1394-9. doi: 10.1007/978-981-95-1394-9_6. URL https://doi.org/10.1007/978-981-95-1394-9_6.
- [43] Rasoul Jafari Gohari, Laya Aliahmadipour, and Ezat Valipour. Fedbrain-distill: Communication-efficient federated brain tumor classification using ensemble knowledge distillation on non-iid data, 09 2024.
- [44] Faisal Mahmud, Fahim Mahmud, and Rashedur M. Rahman. A survey of federated learning: Advances in architecture, synchronization, and security threats. *Computers, Materials and Continua*, 86(3), 2026. ISSN 1546-2218. doi: <https://doi.org/10.32604/cmc.2025.073519>. URL <https://www.sciencedirect.com/science/article/pii/S1546221826000524>.
- [45] Wenzhi Cheng, Wei Ou, Xiangdong Yin, Wanqin Yan, Dingwan Liu, and Chunyan Liu. A privacy-protection model for patients. *Security and Communication Networks*, 2020:1–12, 12 2020. doi: 10.1155/2020/6647562.
- [46] Hongyi Zhang, Jan Bosch, and Helena Holmström Olsson. Federated learning systems: Architecture alternatives. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pages 385–394, 2020. doi: 10.1109/APSEC51365.2020.00047.

- [47] Mohammed H. Alsharif, Raju Kannadasan, Wei Wei, Kottakkaran Sooppy Nisar, and Abdel-Haleem Abdel-Aty. A contemporary survey of recent advances in federated learning: Taxonomies, applications, and challenges. *Internet of Things*, 27:101251, 2024. ISSN 2542-6605. doi: <https://doi.org/10.1016/j.iot.2024.101251>. URL <https://www.sciencedirect.com/science/article/pii/S2542660524001926>.
- [48] Chao Huang, Ming Tang, Qian Ma, Jianwei Huang, and Xin Liu. Promoting collaboration in cross-silo federated learning: Challenges and opportunities. *IEEE Communications Magazine*, 62(4): 82–88, 2024. doi: 10.1109/MCOM.005.2300467.
- [49] Syeda Shahid, Tara Salman, Mohamed Baza, and Marcio Teixeira. Hierarchical federated learning: Approaches, applications and open challenges. *IEEE Access*, PP:1–1, 01 2025. doi: 10.1109/ACCESS.2025.3628455.
- [50] Amani Chachoua, Abdelhamid Malki, and Samir Ouchani. A review on hierarchical federated learning: Architectures and learning algorithms. In *2025 International Conference on Intelligent Computer Systems, Data Science and Applications (IC2SDA)*, pages 1–8, 2025. doi: 10.1109/IC2SDA68097.2025.11331572.
- [51] Chunlu Chen, Ji Liu, Haowen Tan, Xingjian Li, Kevin I-Kai Wang, Peng Li, Kouichi Sakurai, and Dejing Dou. Trustworthy federated learning: privacy, security, and beyond. *Knowledge and Information Systems*, 67(3):2321–2356, 2025. ISSN 0219-3116. doi: 10.1007/s10115-024-02285-2. URL <https://doi.org/10.1007/s10115-024-02285-2>.
- [52] Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. A survey on federated learning: challenges and applications. *International Journal of Machine Learning and Cybernetics*, 14(2):513–535, February 2023. ISSN 1868-808X. doi: 10.1007/s13042-022-01647-y. URL <https://doi.org/10.1007/s13042-022-01647-y>.
- [53] Azim Akhtarshenas, Mohammad Ali Vahedifar, Navid Ayoobi, Behrouz Maham, Tohid Alizadeh, Sina Ebrahimi, and David López-Pérez. Federated learning: A cutting-edge survey of the latest advancements and applications. *Computer Communications*, 228:107964, 2024. ISSN 0140-3664. doi: <https://doi.org/10.1016/j.comcom.2024.107964>. URL <https://www.sciencedirect.com/science/article/pii/S0140366424003116>.

- [54] Wei Guo, Fuzhen Zhuang, Xiao Zhang, Yiqi Tong, and Jin Dong. A comprehensive survey of federated transfer learning: challenges, methods and applications. *Frontiers of Computer Science*, 18(6):186356, July 2024. ISSN 2095-2236. doi: 10.1007/s11704-024-40065-x. URL <https://doi.org/10.1007/s11704-024-40065-x>.
- [55] Liu Yang, Ben Tan, Vincent W. Zheng, Kai Chen, and Qiang Yang. *Federated Recommendation Systems*, pages 225–239. Springer International Publishing, Cham, 2020. ISBN 978-3-030-63076-8. doi: 10.1007/978-3-030-63076-8_16. URL https://doi.org/10.1007/978-3-030-63076-8_16.
- [56] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020. doi: 10.1109/MSP.2020.2975749.
- [57] Hangyu Zhu, Jinjin Xu, Shiqing Liu, and Yaochu Jin. Federated learning on non-iid data: A survey. *Neurocomputing*, 465:371–390, 2021. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2021.07.098>. URL <https://www.sciencedirect.com/science/article/pii/S0925231221013254>.
- [58] Daniel M. Jimenez G., David Solans, Mikko Heikkilä, Andrea Vitaletti, Nicolas Kourtellis, Aris Anagnostopoulos, and Ioannis Chatzigiannakis. Non-iid data in federated learning: A survey with taxonomy, metrics, methods, frameworks and future directions, 2024. URL <https://arxiv.org/abs/2411.12377>.
- [59] Boyu Fan, Siyang Jiang, Xiang Su, Sasu Tarkoma, and Pan Hui. A survey on model-heterogeneous federated learning: Problems, methods, and prospects. In *2024 IEEE International Conference on Big Data (BigData)*, pages 7725–7734, 2024. doi: 10.1109/BigData62323.2024.10825769.
- [60] Aneel Sharma and Ningrinla Marchang. A review on client-server attacks and defenses in federated learning. *Computers Security*, 140:103801, 2024. ISSN 0167-4048. doi: <https://doi.org/10.1016/j.cose.2024.103801>. URL <https://www.sciencedirect.com/science/article/pii/S0167404824001020>.
- [61] Giorgio Severi, Matthew Jagielski, Gökberk Yar, Yuxuan Wang, Alina Oprea, and Cristina Nita-Rotaru. Network-level adversaries in federated learning. In *2022 IEEE Conference on Communications and Network Security (CNS)*, pages 19–27, 2022. doi: 10.1109/CNS56114.2022.9947237.

- [62] Attia Qammar, Jianguo Ding, and Huansheng Ning. Federated learning attack surface: taxonomy, cyber defences, challenges, and future directions. *Artificial Intelligence Review*, 55, 06 2022. doi: 10.1007/s10462-021-10098-w.
- [63] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursay, and Ling Liu. Data poisoning attacks against federated learning systems, 2020. URL <https://arxiv.org/abs/2007.08432>.
- [64] Najeeb Moharram Jebreel, Josep Domingo-Ferrer, David Sánchez, and Alberto Blanco-Justicia. Lfighter: Defending against the label-flipping attack in federated learning. *Neural Networks*, 170:111–126, 2024. ISSN 0893-6080. doi: <https://doi.org/10.1016/j.neunet.2023.11.019>. URL <https://www.sciencedirect.com/science/article/pii/S0893608023006421>.
- [65] Léo Lavaur, Yann Busnel, and Fabien Autrel. Investigating the impact of label-flipping attacks against federated learning for collaborative intrusion detection. *Computers & Security*, 156:104462, 2025.
- [66] Fethi Imad Benaribi, Nabil Sahli, Abdelaziz Amara Korba, and Samir Ouchani. A resilient genetic-centric aggregation for hierarchical federated learning against scaling attacks. *Procedia Computer Science*, 280:454–461, 2026. ISSN 1877-0509. doi: <https://doi.org/10.1016/j.procs.2026.04.058>. URL <https://www.sciencedirect.com/science/article/pii/S1877050926010719>. The 17th International Conference on Ambient Systems, Networks and Technologies Networks (ANT)/the 9th International Conference on Emerging Data and Industry 4.0 (EDI40).
- [67] Moran Baruch, Gilad Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 8632–8642. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/ec1c59141046cd1866bbbcdfb6ae31d4-Paper.pdf>.
- [68] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 1467–1474, 2012. URL <https://arxiv.org/abs/1206.6389>. Available at <https://arxiv.org/abs/1206.6389>.
- [69] Liuwan Zhu Wantong Li. Randeye: On-sensor stochastic image transformation for backdoor-resistant edge inference. In *Proceedings of the ACM Conference*. Association for Computing Ma-

- chinery, 2025. doi: 10.1145/3716368.3735245. URL <https://doi.org/10.1145/3716368.3735245>.
- [70] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning, 2019. URL <https://arxiv.org/abs/1807.00459>.
- [71] Jieyi Deng, Congduan Li, Nanfeng Zhang, Jingfeng Yang, and Jun Gao. Udfed: A universal defense scheme for various poisoning attacks on federated learning. *IEEE Transactions on Information Forensics and Security*, 20:10480–10494, 2025. doi: 10.1109/TIFS.2025.3611126.
- [72] Hanxi Guo, Hao Wang, Tao Song, Yang Hua, Zhangcheng Lv, Xiulang Jin, Zhengui Xue, Ruhui Ma, and Haibing Guan. Siren: Byzantine-robust federated learning via proactive alarming. In *Proceedings of the ACM symposium on cloud computing*, pages 47–60, 2021.
- [73] Shenghui Li, Edith C.-H. Ngai, and Thiemo Voigt. An experimental study of byzantine-robust aggregation schemes in federated learning. *IEEE Transactions on Big Data*, 10(6):975–988, 2024. doi: 10.1109/TBDDATA.2023.3237397.
- [74] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to Byzantine-Robust federated learning. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1605–1622. USENIX Association, August 2020. ISBN 978-1-939133-17-5. URL <https://www.usenix.org/conference/usenixsecurity20/presentation/fang>.
- [75] Evaluating the performance of distributed ledger - blockchain technologies in iot environment. Master’s thesis, Information and Communication Systems Engineering, October 2018. Advisor: Georgios Kambourakis.
- [76] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the works of leslie lamport*, pages 203–226. 2019.
- [77] Junyu Shi, Wei Wan, Shengshan Hu, Jianrong Lu, and Leo Yu Zhang. Challenges and approaches for mitigating byzantine attacks in federated learning. In *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 139–146, 2022. doi: 10.1109/TrustCom56396.2022.00030.
- [78] Silas Pohl. Sydag-fl: a sybil-resistant federated learning approach based on directed acyclic graphs, 2025.

- [79] Moran Baruch, Gilad Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning, 2019. URL <https://arxiv.org/abs/1902.06156>.
- [80] Attia Qammar, Jianguo Ding, and Huansheng Ning. Federated learning attack surface: taxonomy, cyber defences, challenges, and future directions. *Artificial Intelligence Review*, 55(5):3569–3606, 2022. ISSN 1573-7462. doi: 10.1007/s10462-021-10098-w.
- [81] Hao Fang, Yixiang Qiu, Hongyao Yu, Wenbo Yu, Jiawei Kong, Baoli Chong, Bin Chen, Xuan Wang, Shu-Tao Xia, and Ke Xu. Privacy leakage on dnns: A survey of model inversion attacks and defenses. *arXiv preprint arXiv:2402.04013*, 2024.
- [82] Zhenqian Zhu, Jiangang Shu, Xing Zou, and Xiaohua Jia. Advanced free-rider attacks in federated learning. In *The 1st neurIPS workshop on new frontiers in federated learning privacy, fairness, robustness, personalization and data ownership*, page 10, 2021.
- [83] Yupeng Jiang, Yong Li, Yipeng Zhou, and Xi Zheng. Mitigating sybil attacks on differential privacy based federated learning, 2020. URL <https://arxiv.org/abs/2010.10572>.
- [84] Kaixi Nan Chunhua Su Zhuotao Lian, Chen Zhang. Spoil: Sybil-based untargeted data poisoning attacks in federated learning. In *Network and System Security*. Springer, 2023. doi: 10.1007/978-3-031-39828-5_13. URL https://doi.org/10.1007/978-3-031-39828-5_13.
- [85] Hibatallah Kabbaj, Rachid El-Azouzi, and Abdellatif Kobbane. Robust federated learning via weighted median aggregation*. In *2024 2nd International Conference on Federated Learning Technologies and Applications (FLTA)*, pages 298–303, 2024. doi: 10.1109/FLTA63145.2024.10839757.
- [86] Tianxiang Wang, Zhonglong Zheng, and Feilong Lin. Federated learning framework based on trimmed mean aggregation rules. *Expert Systems with Applications*, 270:126354, 2025.
- [87] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Zeno: Byzantine-suspicious stochastic gradient descent. *arXiv preprint arXiv:1805.10032*, 24, 2018.
- [88] Martina Savoia, Ciro Della Bruna, Ciro D’Addio, and Francesco Piccialli. Sage: Sustainable and green energy-aware client selection for federated learning in iot. *Applied Energy*, 410:127568, 2026. ISSN 0306-2619. doi: <https://doi.org/10.1016/j.apenergy.2026.127568>. URL <https://www.sciencedirect.com/science/article/pii/S0306261926002205>.

- [89] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. *arXiv preprint arXiv:2012.13995*, 2020.
- [90] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*, 2018.
- [91] Zhendong Song, Hongguang Sun, Howard H Yang, Xijun Wang, Yan Zhang, and Tony QS Quek. Reputation-based federated learning for secure wireless networks. *IEEE Internet of Things Journal*, 9(2):1212–1226, 2021.