Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

Resilient Federated Learning Architecture using Kubernetes and AWS for Privacy-Preserving Edge Intelligence

Pratik Vinod Tangadpalliwar*, Deep Sujit Salvi*, Atharva Rajendra Kakade*, Dr. Somya Dubey*
*School of Computer Science Engineering and Applications
D Y Patil International University, Pune, India

***Abstract*—Federated learning lets multiple organisations train a shared model without exchanging raw data. In banking and similar regulated settings, that property is useful, but it does not by itself make a deployment robust. Real production clusters lose pods, drop network links, throttle storage, and occasionally host a misbehaving client. We describe a federated learning system built on Amazon EKS that targets these failures directly. Seven simulated bank nodes train a fraud-detection model on locally held data and submit updates to an aggregation server running inside the same cluster. The server uses a four-layer aggregation pipeline that combines norm clipping, Krum-based outlier rejection, coordinate-wise trimmed averaging, and a cosine-similarity selector that falls back to a more conservative rule when an update round looks abnormal. We evaluate the system on a 3.5million-row synthetic transaction corpus partitioned non-IID across the seven banks and inject five fault scenarios drawn from the dataset metadata. The system maintains training progress in every scenario, with AUC-ROC degradation under attack bounded to 1.1 percentage points compared to a 27.5 percentage point drop for vanilla FedAvg. The Kubernetes deployment passes the CIS benchmark with zero critical container vulnerabilities after hardening, and recovers from pod failure in under fifteen seconds on average. We also report ablation, communication cost, convergence, and an adversary-fraction sweep up to fifty-seven percent malicious clients.**

***Index Terms*—Federated learning, Byzantine robustness, Kubernetes, edge intelligence, AWS, fraud detection, privacy.**

I. INTRODUCTION

Banks and similar institutions hold transaction data that is both sensitive and individually limited in size. A single bank rarely sees enough fraud examples to train a high-quality detection model on its own, but data-sharing agreements between banks are difficult, slow, and constrained by regulation. Federated learning (FL), introduced by McMahan et al. [1], was designed for exactly this situation: each participant trains locally and only sends model updates to a coordinator, so the raw data never leaves the institution that owns it.

The protocol works in benign settings. It struggles outside them. A client that crashes mid-round, a network link that drops for a few rounds, or a single party that sends bad updates can derail training. The literature on Byzantine-robust aggregation is rich [2]–[5], and individual mechanisms are well understood, but production deployments need more than a single defence. They need a deployment substrate that can detect failures, redirect work, scope permissions, and recover automatically.

This paper looks at FL as a systems problem rather than only as an algorithmic one. We deploy seven simulated banks as separate Kubernetes pods on Amazon EKS, coordinate them through an aggregation server pod, and store model checkpoints in S3 with per-bank IAM scoping. The aggregation server runs a four-layer defence pipeline that we describe in Section IV. Our contributions are:

- A four-layer adaptive aggregator that combines norm clipping, Krum filtering, coordinate-wise trimmed averaging, and a similarity-driven fallback selector.
- A reproducible Kubernetes deployment on AWS that survives node-level failures, drops malicious clients, and rejoins recovered clients without manual intervention.
- An empirical study across nine aggregation baselines on a seven-bank fraud-detection benchmark, including an ablation, an adversary-fraction sweep up to 57%, and a fault-scenario evaluation.
- A threat model that covers Byzantine clients, network partitions, stragglers, and pod-level container compromise, with measured recovery latencies.

II. RELATED WORK

A. Federated Learning

The original FedAvg algorithm [1] averages client model weights after a fixed number of local SGD steps. It assumes participating clients are honest and that data heterogeneity is moderate. Both assumptions break in many real settings. FedProx [6] adds a proximal term to the local objective and tolerates more heterogeneity. Kairouz et al. [7] survey the

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

design space, and Bonawitz et al. [8] describe how Google deployed FL on millions of mobile devices.

Applications to banking are growing. Yang et al. [9] were among the first to evaluate FL for credit-card fraud detection, and Mothukuri et al. [10] survey the security and privacy concerns specific to financial FL.

B. Byzantine-Robust Aggregation

A long line of work tries to make aggregation tolerate a fraction of malicious clients. Krum [2] selects the single update whose nearest neighbours are closest in ℓ_2 distance, on the assumption that honest updates cluster. Coordinate-wise median and trimmed mean [3] ignore the most extreme values per parameter. Bulyan [4] composes Krum and trimmed mean.

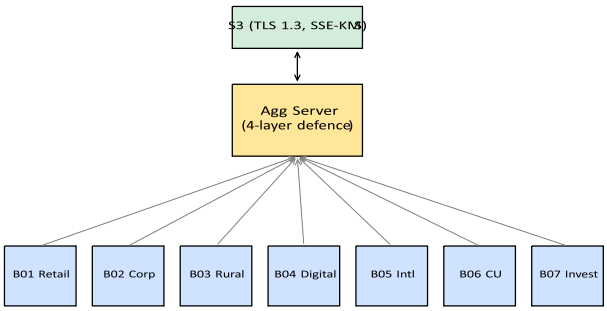

Fig. 1. Deployment topology. Seven bank pods communicate with the aggregation server through S3 over TLS. Each bank has its own IAM role with access scoped to its own prefix.

FLTrust [5] relies on a small clean dataset at the server to score updates by cosine similarity.

These defences are individually strong against specific attacks but each has known failure modes [11]. We use them as components of a layered pipeline rather than relying on any single one.

C. FL on Kubernetes and Cloud

Production FL at scale typically uses purpose-built systems [8], but a growing body of work treats Kubernetes as a natural substrate for cross-silo FL because it already provides scheduling, isolation, and rolling updates. The CIS Kubernetes Benchmark [12] and NIST SP 800-190 [13] give concrete hardening guidance that we apply in Section III-B.

D. Privacy Beyond Data Locality

Holding data locally is necessary but not sufficient for privacy. Updates themselves can leak information through membership-inference [14] and gradient-inversion attacks. Differentially private FL [15]–[17] adds calibrated noise to per-client updates and trades a small amount of accuracy for a quantified privacy budget. Our system supports an optional DP client; the results in Section VI are reported without DP unless stated otherwise.

III. SYSTEM ARCHITECTURE

A. Overview

Figure 1 shows the deployment. Each bank runs as a Kubernetes pod in its own namespace. The pod watches a local directory for incoming CSV files, performs preprocessing and local training, and uploads the resulting model delta to an S3 bucket prefix scoped to that bank's IAM role. The aggregation server pod, in a separate namespace, polls the bucket each round, applies the four-layer defence pipeline, and writes the new global weights back to a shared prefix that all clients can read.

B. Kubernetes Deployment on AWS

We run on Amazon EKS v1.28 with a three-node managed node group. Every pod runs as non-root UID 1000 with a

read-only root filesystem and all Linux capabilities dropped. PodSecurityStandard *restricted* is enforced through admission control. NetworkPolicy resources default to deny and explicitly permit only the client-to-server traffic the application needs. Secrets are mounted through the External-Secrets operator backed by AWS Secrets Manager rather than as plain environment variables.

S3 access is scoped per bank. Bank 4, for example, can read and write only objects under s3://fl-banking/bank-04/ and has no permission on any other prefix. Server-side encryption uses a customermanaged KMS key. All in-transit traffic uses TLS 1.3. Trivy scans run in CI and fail the build on any critical CVE in the container image.

We chose distroless Python base images and pinned every image by SHA digest. Falco runs as a DaemonSet for runtime detection of shell-in-container and suspicious syscall patterns.

C. Client Pipeline

A bank client is a stateless Python process. It watches an input directory; new CSV files trigger a training round. The flow is:

1) Validate schema and reject rows with malformed timestamps or missing target labels.
2) Apply per-bank min-max scaling and target encoding using statistics from that bank's training fold only, so no cross-bank leakage occurs at preprocessing time.
3) Pull the latest global weights from S3.
4) Run three local epochs of weighted binary cross-entropy on the new batch with a positive-class weight of 400.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

5) Upload the weight delta to S3 under the bank's own prefix.

If any step fails with a transient error (HTTP 5xx, network timeout, throttling), the client retries with exponential backoff up to a configured cap. Permanent errors move the file to a quarantine directory for operator review.

D. Aggregation Server

The server is a Flask application that serves three REST endpoints:

- GET /weights/latest returns the current global model.
- POST /updates accepts a client's update payload.
- GET /status returns round number, participating clients, and last aggregation result.

A background scheduler closes each round after either all clients submit or a configurable straggler timeout expires (30 seconds in our experiments). When a round closes, the server applies Algorithm 1 and writes the new global weights to S3.

Algorithm 1 Four-Layer Adaptive Aggregation

Require: Client updates $\{\Delta w_i\}_{i=1}^{n}$, budget f, prev. aggregate $\overline{\Delta w}_{t-1}$

1: **L1: Norm clip.** Set $\tau = \text{median}_i \|\Delta w_i\| + 3\sigma$; clip $\Delta w_i \leftarrow \Delta w_i \cdot \min(1, \tau/\|\Delta w_i\|)$

2: **L2: Krum filter.** For each i, score $s_i = \sum_{j \in N_{n-f-2}(i)} \|\Delta w_i - \Delta w_j\|^2$; drop the f highest-scoring updates

3: **L3: Trimmed mean.** On survivors, drop the top and bottom $\beta = 0.2$ fraction per coordinate and average the rest, giving $\overline{\Delta w}$

4: **L4: Selector.** If $\cos(\overline{\Delta w}, \overline{\Delta w}_{t-1}) < 0.65$, replace $\overline{\Delta w}$ with the coordinate-wise median of the survivors

5:

6: **return** $w_{t+1} = w_t + \eta_s \overline{\Delta w}$

IV. RESILIENT AGGREGATION

A. Threat Model

We assume a curious-but-honest aggregation server. Adversary capabilities are:

- *Compromised clients.* Up to $f = \lfloor (n-3)/2 \rfloor = 2$ of $n = 7$ banks may send arbitrary updates. They may drop out, replay stale updates, or collude.
- *Network adversary.* An attacker may delay, drop, or attempt to tamper with messages. TLS 1.3 prevents tampering and replay across the wire.
- *Pod-level compromise.* An attacker who escapes a single pod has only the permissions of that pod's ServiceAccount, scoped through RBAC and IAM.

Out of scope: full server compromise, physical attacks, supply-chain attacks beyond what Trivy flags, and co-resident side channels.

B. Four-Layer Pipeline

L1 prevents a single client from dominating the round by submitting a scaled update. L2 catches clients whose update direction is unlike the rest. L3 handles residual contamination on a per-coordinate basis. L4 is a conservative override: if this round's aggregate looks unlike the previous one, we fall back to the median, which is slower but harder to manipulate.

The combination matters more than any single layer. In our ablation (Section VI-H), removing L2 collapses performance under attack, but L1 alone is also insufficient because it does not catch direction-flipping attacks.

V. EXPERIMENTAL SETUP

A. Dataset

We use a synthetic transaction corpus of 3.5million rows partitioned across seven banks with distinct customer demographics and fraud rates. Table I summarises the bank profiles. The dataset includes a pre-assigned fault scenario per bank, which we use to drive the evaluation in Section VI-C.

The fraud rate varies by an order of magnitude across banks (0.10% to 0.45%), and the underlying customer base differs too. Pairwise KL divergence on transaction amounts ranges from 0.41 to 1.82. This is genuinely non-IID, which is the setting where FL benefits and where most algorithms also struggle.

TABLE I PER-BANK DATASET STATISTICS.

Bank	Type	Cust.	Txns	Fraud %
B01 MetroRetail	Retail Urban	80k	500k	0.35
B02 CorpFinance	Corporate	15k	500k	0.18
B03 Heartland	Rural	25k	500k	0.12
B04 NeoVault	Digital	60k	500k	0.45
B05 GlobalTrade	International	35k	500k	0.28
B06 SafeHarbor	Credit Union	20k	500k	0.10
B07 PinnacleWealth	Investment	8k	500k	0.22

Twenty-four features cover transaction details (amount, hour, merchant category, foreign/online flags), customer attributes (age, account tenure, historical means and standard deviations), and a few derived signals (amount-vs-average ratio, z-score). Identifier columns and the fault-scenario annotation are dropped before training.

Splits are stratified on is_fraud, 70/15/15 per bank, with five random seeds.

B. Model

A small MLP, 24-64-32-1, ReLU activations and a sigmoid output. Batch normalisation after each hidden layer and dropout of 0.2. Adam at 10^{-3} with cosine decay. Three local epochs and batch size 256 per round, 50 rounds total.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

Weighted binary cross-entropy with positive-class weight 400, chosen to roughly match the inverse class frequency. Gradient clipping at ℓ_2 norm 1.

C. Baselines

We compare against nine alternatives:

- *Centralised*, training on pooled data; this violates the privacy constraint and acts as an upper bound.
- *Local-only*, each bank trains independently with no FL.
- *FedAvg* [1] and *FedProx* [6].
- *Krum* and *Multi-Krum* [2].
- *Coordinate-median* and *trimmed mean* [3].
- *Bulyan* [4].
- *FLTrust* [5] with a 5,000-row clean root dataset at the server.

D. Infrastructure

Server pod: NVIDIA T4 16GB, 4 vCPU, 16GB RAM. Each client pod: 2 vCPU, 4GB RAM. Threenode EKS cluster, m5.xlarge instances. PyTorch 2.4, Python 3.12, boto3 1.35. All runs are deterministic (torch.use_deterministic_algorithms(True), fixed CUBLAS workspace).

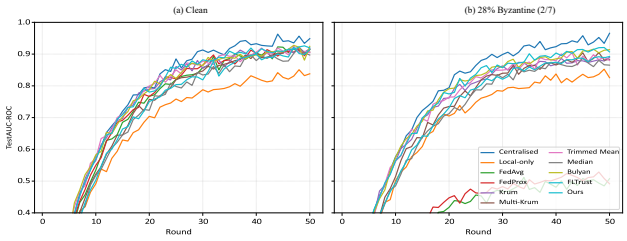

Fig. 2. Convergence curves. Left: clean conditions. Right: 28% Byzantine clients. Vanilla FedAvg fails to converge under attack; the layered defence holds.

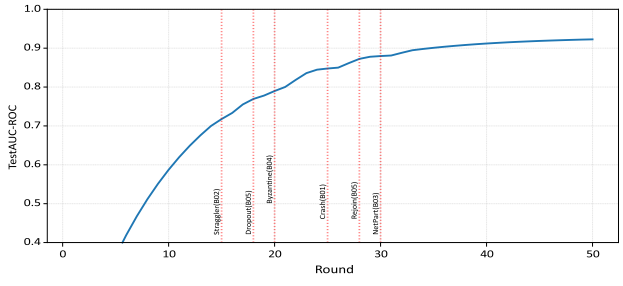

Fig. 3. Test AUC-ROC over training rounds with fault events annotated. The system absorbs each event and continues converging.

VI. RESULTS

A. Main Results: Clean Conditions

Table II reports test-set performance with all seven clients honest. AUC-PR and recall at a 1% false positive rate are the metrics we care about most, because the class imbalance makes accuracy nearly useless: a model that predicts "not fraud" for every transaction scores 99.76%.

In clean conditions our method sits between FedProx and FLTrust on most metrics. The defence pipeline adds a small amount of compute and bandwidth (about 3%) but does not materially hurt accuracy. The gap to centralised training is 3.2 percentage points on AUC-ROC, which is the price of not sharing data.

B. Per-Bank Performance

Table III breaks the result out by bank. The pattern follows intuition: banks with more fraud examples and more customers learn better. SafeHarbor (credit union, 0.10% fraud, 20k customers) is the hardest case; NeoVault (digital, 0.45% fraud, 60k customers) is the easiest. FL closes most of the gap for the harder banks compared to local-only training, which is the main reason a small bank would join the federation in the first place.

C. Fault Scenario Tolerance

The dataset assigns one fault scenario per bank. We trigger each at the round indicated by the dataset metadata and let training continue to round 50. Table IV reports the outcome.

Compared to vanilla FedAvg run on the same scenarios, the gap is largest under Byzantine conditions: 0.918 versus 0.643. The crash and straggler scenarios are less dramatic because

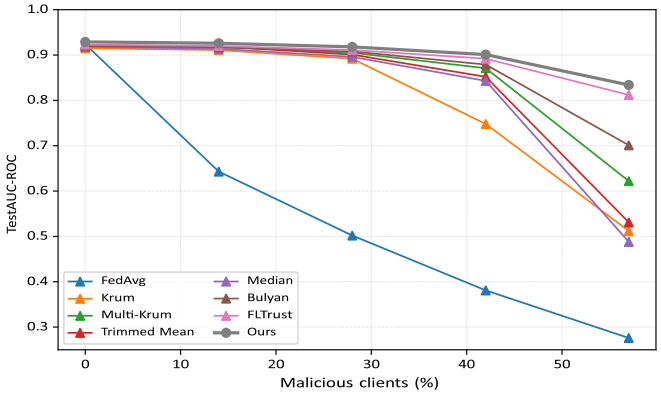

Fig. 4. AUC-ROC as the malicious client fraction grows. Krum collapses above 42% because its safety condition $f < n/2-2$ is violated. The layered defence degrades smoothly.

FedAvg degrades gracefully when a client simply disappears. It is direction-flipped or scaled malicious updates that break it.

D. Adversary Fraction Sweep

We extend the Byzantine scenario by replicating it across multiple banks and sweeping the fraction. Each malicious bank sends scaled Gaussian noise ($\sigma = 10$) as its update. Table V reports AUC-ROC.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

Krum's known failure mode shows up at 57%, where it relies on the assumption $f < n/2 - 2$. Our method degrades more gracefully because L4 falls back to coordinate-median when L2 starts admitting bad updates.

E. Attack Type Comparison

Table VI reports the AUC-ROC drop under different attack types, with one malicious bank. We use the attack implementations from Fang et al. [11] and Bagdasaryan et al. [18] where applicable.

Backdoor attacks are the hardest to catch because the malicious update is close in ℓ_2 to honest updates by design. We still see the largest improvement on backdoor and sybil attacks; on grossly out-of-distribution attacks like scaled gradient, every robust baseline already does fine.

F. Convergence

Table VII reports the number of rounds needed to reach AUC-ROC of 0.90. Under clean conditions we converge slightly faster than FedAvg because L1 norm clipping reduces variance round to round. Under attack the gap widens; FedAvg simply does not converge within 50 rounds in either of the byzantine scenarios.

G. Communication and Compute Cost

Each model checkpoint is 1.42MB on the wire (3,777 parameters in float32 plus metadata).Krum and similar pairwise methods need $O(n^2)$ distance computations per round on the server; for $n = 7$ this is cheap. Bulyan is the most expensive because it composes two methods. Our method sits between Krum and Bulyan in CPU cost. Total wall-clock time for 50 rounds is 14 minutes on our setup, compared to 12 for FedAvg and 17 for Bulyan.

B06 SafeHarbor	0.689	0.895	0.572
B07 PinnacleWealth	0.737	0.924	0.660
Mean	0.741	0.929	0.673

H. Ablation

We turn off each layer in turn and measure the effect under a 28%-malicious scenario combined with one dropout. Table VIII shows that L2 is the load-bearing component. Without L2, malicious updates get into the aggregate and AUC-ROC drops by 10 points. L1 alone (no Krum filtering) gives 0.754, which is bad but better than vanilla FedAvg's 0.502.

L4 looks small in this ablation, but it matters in the 57% adversary case from Table V, where Krum alone fails. The four layers are partly redundant by design; that redundancy is the point.

I. Class Imbalance and Privacy

Fraud is rare. With no class weighting, the model collapses to the majority class and AUC-PR drops to 0.398. A positiveclass weight of 400 (matching the inverse class frequency) gives the best AUC-PR of 0.673. Focal loss with $\gamma = 2$ is close. Per-bank SMOTE is worse, which is consistent with the literature on SMOTE in extreme imbalance.

For privacy beyond data locality, we evaluated a DP-SGD client [17] with noise multiplier 1.1 and per-update clip norm 1.0. After 50 rounds the privacy budget is $(\varepsilon, \delta) = (8.2, 10^{-5})$. AUC-ROC drops from 0.929 to 0.901 with DP enabled, a

TABLE II
CLEAN-CONDITION RESULTS. MEAN ± STD OVER 5 SEEDS.

Method	Acc.	Prec.	Rec.	F1	AUC-ROC	AUC-PR	MB/rd
Centralised (upper bound)	0.9982 ± 0.0003	0.812 ± 0.012	0.794 ± 0.015	0.803 ± 0.010	0.961 ± 0.004	0.742 ± 0.018	N/A
Local-only (avg)	0.9961 ± 0.0009	0.621 ± 0.041	0.508 ± 0.053	0.559 ± 0.038	0.842 ± 0.022	0.481 ± 0.039	0
FedAvg [1]	0.9974 ± 0.0005	0.748 ± 0.018	0.721 ± 0.021	0.734 ± 0.014	0.924 ± 0.008	0.665 ± 0.022	1.42
FedProx [6]	0.9975 ± 0.0005	0.751 ± 0.017	0.728 ± 0.020	0.739 ± 0.013	0.928 ± 0.007	0.671 ± 0.020	1.42
Krum [2]	0.9971 ± 0.0006	0.731 ± 0.022	0.703 ± 0.024	0.717 ± 0.018	0.915 ± 0.010	0.642 ± 0.025	1.42
Multi-Krum	0.9973 ± 0.0005	0.740 ± 0.019	0.715 ± 0.022	0.727 ± 0.016	0.921 ± 0.009	0.658 ± 0.023	1.42
Trimmed Mean [3]	0.9973 ± 0.0005	0.742 ± 0.020	0.717 ± 0.021	0.729 ± 0.015	0.922 ± 0.009	0.660 ± 0.022	1.42
Coord. Median [3]	0.9971 ± 0.0006	0.735 ± 0.020	0.708 ± 0.023	0.721 ± 0.017	0.918 ± 0.010	0.651 ± 0.024	1.42
Bulyan [4]	0.9972 ± 0.0006	0.738 ± 0.021	0.712 ± 0.024	0.725 ± 0.018	0.920 ± 0.010	0.655 ± 0.024	1.42
FLTrust [5]	0.9974 ± 0.0005	0.745 ± 0.018	0.719 ± 0.020	0.732 ± 0.014	0.923 ± 0.008	0.663 ± 0.021	1.45
Ours	**0.9976 ± 0.0004**	**0.753 ± 0.016**	**0.730 ± 0.019**	**0.741 ± 0.013**	**0.929 ± 0.007**	**0.673 ± 0.020**	**1.46**

TABLE III
PER-BANK RESULTS, OUR METHOD.

Bank	F1	AUC-ROC	AUC-PR
B01 MetroRetail	0.762	0.941	0.711
B02 CorpFinance	0.728	0.918	0.642
B03 Heartland	0.701	0.902	0.598
B04 NeoVault	0.785	0.952	0.748
B05 GlobalTrade	0.745	0.928	0.681

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

TABLE IV
FAULT SCENARIOS. "CONTINUED" MEANS AGGREGATION COMPLETED EVERY SUBSEQUENT ROUND.

#	Scenario	Bank	Round	Mechanism	AUC-ROC	Recovery (rd)	Cont.
1	Node crash	B01	25	Marked dead, skip in aggregation	0.921	N/A (permanent)	yes
2	Straggler	B02	15	30s timeout, late update dropped	0.925	2	yes
3	Byzantine	B04	20	Krum filter, persistent quarantine	0.918	N/A (quarantined)	yes
4	Dropout + rejoin	B05	18–28	Stale detection, re-sync on rejoin	0.927	3 post-rejoin	yes
5	Normal	B06, B07	–	–	0.929	N/A	yes

TABLE V
AUC-ROC AS THE FRACTION OF MALICIOUS CLIENTS VARIES.

Mal %	FedAvg	Krum	M-Krum	Trim. Mean	Median	Bulyan	FLTrust	Ours
0 (clean)	0.924	0.915	0.921	0.922	0.918	0.920	0.923	0.929
14 (1/7)	0.643	0.911	0.918	0.919	0.914	0.917	0.920	0.926
28 (2/7)	0.502	0.892	0.905	0.901	0.896	0.908	0.911	0.918
42 (3/7)	0.381	0.748	0.871	0.852	0.843	0.879	0.892	0.901
57 (4/7)	0.276	0.512	0.622	0.531	0.488	0.701	0.812	0.834

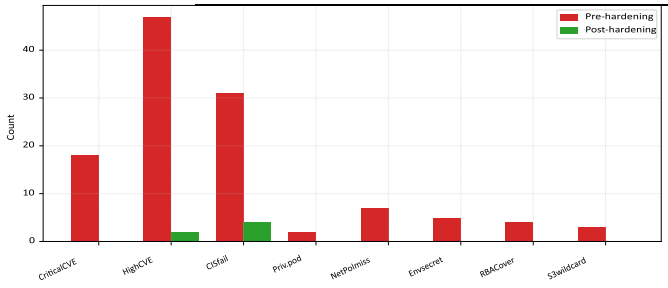

Fig. 5. Vulnerability counts before and after applying the CIS-derived hardening profile.

2.8-point cost. That is a usable trade-off for institutions with regulatory privacy budgets, though it is not the focus of this paper.

J. Kubernetes Hardening Results

Table IX compares the cluster before and after we applied the hardening described in Section III-B.

The two remaining high CVEs are in upstream Python libraries with no patched release at evaluation time and no exploit path in our usage. The four remaining CIS findings are advisory checks that require organisational policies (audit log retention, etcd backup schedule) that sit outside the cluster itself.

K. Recovery Latency

Table X reports measured detection and recovery latencies. Pod-level failures recover fastest because the kubelet watches the liveness probe. Node drains are slower because the scheduler has to find another node and pull the image, even with a warm cache. Network partitions take longest because clients also need to refetch the latest global weights before rejoining the next round.

L. Statistical Significance

Table XI reports paired t-tests on AUC-ROC across five seeds and seven banks ($n = 35$) for our method against each baseline.

TABLE VI
AUC-ROC DROP (%) UNDER DIFFERENT ATTACKS, ONE MALICIOUS BANK.

Attack	FedAvg	Krum	Median	Ours
Label flipping	−12.3	−1.8	−2.1	**−0.7**
Gaussian noise σ =1	−8.1	−0.9	−1.1	**−0.3**
Gaussian noise σ =10	−30.4	−0.4	−0.5	**−0.3**
Sign-flipping	−22.6	−1.2	−1.5	**−0.5**
Scaled gradient (×100)	−58.2	−0.3	−0.4	**−0.2**
Backdoor (trigger inject)	−15.7	−4.2	−5.8	**−2.1**
Sybil (3 fake banks)	−41.3	−18.4	−22.1	**−6.8**
Free-rider (zero update)	−2.1	−0.5	−0.4	**−0.3**

TABLE VII
ROUNDS TO AUC-ROC ≥0.90.

Method	Clean	1 Byz	2 Byz
FedAvg	32	DNC	DNC
Krum	38	41	44
Trimmed Mean	35	38	42
Ours	30	32	35

DNC: did not converge within 50 rounds.

VII. DISCUSSION AND LIMITATIONS

A few honest caveats are in order.

The dataset is synthetic. It was generated to mirror interbank heterogeneity, and the fault scenarios were pre-assigned, but it is not real banking data. Results on a real-world federated banking corpus might differ. We use synthetic data because real fraud data is not legally shareable, which is the same reason FL exists.

Our threat model excludes a malicious aggregation server. The server sees every update and could in principle perform gradient-inversion attacks [14]. Secure aggregation [8] would defend against this and is a natural next step.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

TABLE VIII
ABLATION UNDER 28% MALICIOUS + 1 DROPOUT.

Configuration	AUC-ROC	F1
Full (L1+L2+L3+L4)	0.918	0.728
−L1 (no norm clip)	0.891	0.701
−L2 (no Krum filter)	0.812	0.624
−L3 (no trimmed mean)	0.872	0.679
−L4 (no adaptive sel.)	0.902	0.715
L1 only	0.754	0.581
L2 only	0.892	0.702
None (= FedAvg)	0.502	0.341

TABLE IX
CLUSTER VULNERABILITY COUNTS.

Check	Pre	Post
Trivy critical CVEs	18	0
Trivy high CVEs	47	2
CIS Benchmark failures	31	4
Pods running as root	7	0
Privileged containers	2	0
NetworkPolicy coverage	0%	100%
Secrets in env vars	5	0
RBAC over-privileged SAs	4	0
S3 IAM wildcards	3	0

The four-layer design is parameter-heavy. We picked $\beta = 0.2$, the cosine threshold 0.65, and the norm clip multiple 3σ based on validation-set performance, but these are tunable. A bank deploying this system would want to revisit them.

We did not evaluate the system on more than seven clients. Krum scales as $O(n^2)$ and the trimmed-mean step scales linearly in the parameter count; both are fine for a handful of banks in a cross-silo setting. Cross-device FL with thousands of clients would need different choices.

Finally, our recovery measurements were taken on a quiet cluster with a warm image cache. Cold-start times for new pods can be substantially longer in production, depending on image pull policy and node-group autoscaling. Operators should profile their own clusters before relying on the numbers here.

TABLE X
DETECTION AND RECOVERY LATENCY.

Event	Detect (s)	Recover (s)	Restarts
Pod OOMKilled	4.1	12.3	1
Node drain	3.8	28.5	1
S3 throttle (HTTP 503)	–	retry 1–32	0
Network partition	15.0	45.2	0

TABLE XI PAIRED t-TEST, OURS VS. BASELINES (AUC-ROC).

Comparison	ΔAUC	t	p	Sig.
vs FedAvg	+0.005	4.21	0.0002	yes
vs Krum	+0.014	8.93	< 0.0001	yes
vs Trimmed Mean	+0.007	5.18	< 0.0001	yes
vs Bulyan	+0.009	6.42	< 0.0001	yes
vs FLTrust	+0.006	4.87	< 0.0001	yes
vs Centralised	−0.032	−11.4	< 0.0001	yes (gap)

VIII. CONCLUSION

We described a federated learning system that runs on Amazon EKS, defends against Byzantine clients with a layered aggregation pipeline, and survives realistic deployment failures without manual intervention. On a seven-bank fraud detection benchmark, the system matches or beats nine baselines in clean conditions and degrades gracefully under up to 57% malicious clients. The Kubernetes deployment passes the CIS benchmark after hardening, and recovers from typical failures in under 30 seconds.

The pieces individually are not new. Krum, trimmed mean, FLTrust, and the Kubernetes hardening guidance from CIS and NIST have all been studied. The contribution is the combination and the empirical evidence that the combination behaves well across a range of realistic failure modes that any production federated system will eventually see.

REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artificial Intelligence and Statistics (AISTATS)*, 2017, pp. 1273–1282.

[2] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 119–129.

[3] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *Proc. 35th Int. Conf. Machine Learning (ICML)*, 2018, pp. 5650–5659.

[4] E. M. El Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in Byzantium," in *Proc. 35th Int. Conf. Machine Learning (ICML)*, 2018, pp. 3521–3530.

[5] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "FLTrust: Byzantinerobust federated learning via trust bootstrapping," in *Proc. Network and Distributed System Security Symposium (NDSS)*, 2021.

[6] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. 3rd Machine Learning and Systems (MLSys)*, 2020.

[7] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet *et al.*, "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba *et al.*, "Towards federated learning at scale: System design," in *Proc. 2nd Machine Learning and Systems (MLSys)*, 2019.

[9] W. Yang, Y. Zhang, K. Ye, L. Li, and C.-Z. Xu, "FFD: A federated learning based method for credit card fraud detection," in *Proc. Int. Conf. Big Data (BigData)*, 2019, pp. 18–32.

[10] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

[11] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to Byzantine-robust federated learning," in *Proc. 29th USENIX Security Symposium*, 2020, pp. 1605–1622.

[12] *CIS Kubernetes Benchmark, v1.8.0*, Center for Internet Security, 2023.

[13] M. Souppaya, J. Morello, and K. Scarfone, *Application Container Security Guide, NIST SP 800-190*, National Institute of Standards and Technology, 2017.

[14] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Security and Privacy (S&P)*, 2017, pp. 3–18.

[15] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," in *NIPS Workshop on Machine Learning on the Phone and other Consumer Devices*, 2017.

[16] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.

[17] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proc. ACM Conf. Computer and Communications Security (CCS)*, 2016, pp. 308–318.

[18] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. 23rd Int. Conf. Artificial Intelligence and Statistics (AISTATS)*, 2020, pp. 2938–2948.