EdgeGuard-FL: A Privacy-Preserving Federated Learning Framework for Intrusion Detection in Smart Campus IOT Networks

Prof.S. AbikayilAarthi
Assistant Professor,
Department of Computer Science and Engineering,
Kings College of Engineering,
Punalkulam, TN, India,
aarthi.cse@kingsengg.edu.in

Adhiga M
Department of Computer Science and Engineering,
Kings College of Engineering,
Punalkulam, TN, India,
aadhigamathiyalagan@gmail.com

Harini E
Department of Computer Science and Engineering,
Kings College of Engineering,
Punalkulam, TN, India,
harinielavarasan2@gmail.com

Hamsavarthini M
Department of Computer Science and Engineering,
Kings College of Engineering,
Punalkulam, TN, India,
hamsam7452@gmail.com

Monika P
Department of Computer Science and Engineering,
Kings College of Engineering,
Punalkulam, TN, India,
monikapremanand123@gmail.com

Jeevajothi P
Department of Computer Science and Engineering,
Kings College of Engineering,
Punalkulam, TN, India,
jeevapathma2212@gmail.com

Kavitha S
Department of Computer Science and Engineering,
Kings College of Engineering,
Punalkulam, TN, India,
kavidhamani2006@gmail.com

ABSTRACT—Smart campuses connect classroom projectors, attendance terminals, surveillance cameras, laboratory sensors, and hostel Wi-Fi infrastructure to a shared network. Detecting intrusions in such environments normally requires pooling traffic records from every department in one place, which raises privacy, bandwidth, and governance concerns. This paper proposes EdgeGuard-FL, a federated learning framework in which each campus edge node trains a lightweight intrusion-detection model on its own traffic and shares only protected model updates with a coordinating server. The design combines flow-level feature extraction, local anomaly and classification models, clipped and noised updates, trust-weighted aggregation to resist poisoned clients, and a drift monitor that triggers re-training when traffic behaviour changes. A graded response policy maps detector output to log, alert, rate-limit, or isolate actions. The paper defines the threat model, architecture, dataset and partitioning plan, candidate detection models, evaluation metrics, and deployment safeguards. As no implementation experiment is claimed, the results section specifies a reproducible evaluation plan rather than reporting invented measurements. EdgeGuard-FL aims to provide a practical foundation for collaborative network defence that balances detection quality, privacy, communication cost, and operational simplicity.

Keywords—Federated Learning, Intrusion Detection, Internet of Things, Differential Privacy, Anomaly Detection, Edge Computing, Network Security.

I. INTRODUCTION

A. Background

Educational institutions increasingly operate dense networks of connected devices. Lecture halls use networked displays and smart boards, libraries use RFID gates, laboratories use sensor boards and shared servers, and students connect personal laptops and phones through campus Wi-Fi. This diversity expands the attack surface: a compromised camera, a misconfigured lab server, or an infected student device can be used for scanning, credential guessing, or denial-of-service activity against other systems [3].

Machine-learning-based intrusion detection systems learn the difference between normal and malicious traffic from examples. The usual approach is centralised: every department forwards its traffic records to one server, where a model is trained. Centralisation, however, concentrates sensitive information such as device identifiers, browsing patterns, and internal addresses in one location, and it requires the transfer of large volumes of data over the campus backbone.

B. Problem Statement

Federated learning (FL) [1] allows several participants to train a shared model while keeping raw data local. Applying it to intrusion detection is not straightforward. Each department sees a different traffic mixture, so local data are non-identically distributed. Some participants may be compromised and submit poisoned updates. Shared gradients can leak information about the underlying traffic [12]. Edge nodes such as single-board computers have limited memory and processing power. The challenge is therefore both a learning problem and a system-design problem: obtaining a detector that generalises across departments without exposing local data or trusting every participant blindly.

C. Motivation

Attack patterns that appear first in one department, such as a port scan against the laboratory subnet, are valuable knowledge for the others. A federated detector can share this knowledge without sharing the traffic that produced it. At the same time, privacy regulations and institutional policies increasingly discourage uncontrolled copying of network logs. A framework that couples collaborative learning with explicit privacy and robustness controls is therefore attractive for institutions that lack a large dedicated security team.

D. Objectives and Contributions

This paper has six objectives: define a threat model for federated intrusion detection on a campus network; design a lightweight edge detector; specify privacy-preserving update

sharing; propose trust-weighted aggregation that tolerates a minority of faulty or malicious clients; define a dataset partitioning and evaluation plan; and describe deployment controls that remain effective if the learned model fails. The main contribution is the modular EdgeGuard-FL architecture with four response outcomes: log, alert, rate-limit, and isolate. The paper also proposes a reproducible comparison of centralised, local-only, and federated training. All proposed metrics and expected trade-offs are clearly distinguished from experimental results.

II. LITERATURE REVIEW

A. Federated Learning

Federated averaging [1] trains a global model by repeatedly sending it to clients, performing several local gradient steps, and averaging the returned weights on a server. Subsequent work addressed statistical heterogeneity, for example by adding a proximal term that limits local drift [2], and a broad survey [3] catalogues open problems in communication efficiency, personalisation, privacy, and robustness. Cross-silo settings, where a small number of organisational units participate reliably, are closer to the campus scenario than cross-device settings with millions of phones.

B. Learning-Based Intrusion Detection

Intrusion detection approaches are commonly divided into signature-based and anomaly-based methods. Signature methods are precise for known attacks but miss novel ones. Anomaly methods model normal behaviour and flag deviations; an ensemble of autoencoders for online detection on lightweight hardware was demonstrated in Kitsune [7]. Public benchmarks such as CIC-IDS2017 [8] and UNSW-NB15 [11] provide labelled flows for supervised evaluation, although their traffic does not fully represent any specific institution.

C. Privacy and Robustness Techniques

Differential privacy [4] bounds how much any single record can influence a released result, and its application to deep learning through gradient clipping and noise addition was introduced in [6]. Secure aggregation [5] lets a server compute the sum of client updates without seeing individual ones. On the robustness side, Byzantine-tolerant aggregation rules [9] resist arbitrary faulty clients, while model-replacement backdoors [10] show that a single malicious participant can implant hidden behaviour if aggregation is naive. Concept drift [13] describes the change of data distributions over time and is especially relevant to evolving network traffic.

D. Research Gap

A practical campus deployment needs a framework that combines these ingredients while remaining light enough for modest edge hardware. It must explain its decisions to administrators, tolerate unreliable or hostile participants, and not rely on a single privacy mechanism. A binary "attack or normal" output may also be too coarse, because a response such as isolating a device has real cost. EdgeGuard-FL addresses this gap by combining local detection, protected updates, trust-weighted aggregation, drift monitoring, and a graded response policy.

III. THREAT MODEL AND REQUIREMENTS

A. Assets to Protect

The assets include raw traffic records, device inventories, the global detection model, model-update channels, administrator credentials, and the availability of campus services. Integrity matters as much as confidentiality: a poisoned global model may stop detecting a specific attack, or may raise alarms against legitimate services and disrupt teaching. The system should detect malicious activity while preventing the learning process itself from leaking or corrupting information.

B. Adversary Model

The adversary may be an external attacker scanning or flooding the campus network, or an insider who controls a subset of edge nodes and submits crafted model updates. The adversary may observe protected updates and know the general architecture, but is assumed not to control the aggregation server's private keys, the secure-aggregation setup, or a majority of participants. The threat model does not assume that every client is honest; it assumes that client trust is uncertain and must be evaluated continuously.

C. Security and Functional Requirements

EdgeGuard-FL should keep raw traffic on the node where it was captured, limit information leakage from shared updates, tolerate a bounded fraction of poisoned clients, run within the memory and latency limits of edge hardware, and produce explainable alerts. The detector must never be the only control: network isolation actions require independent authorisation, and a defined fallback applies when the model or the server is unavailable.

D. Attack Categories

The taxonomy covers reconnaissance and port scanning, brute-force authentication, volumetric denial of service, botnet command-and-control beaconing, data exfiltration, model poisoning, backdoor insertion, gradient-inversion attempts, and free-riding clients that contribute no useful learning. Each example should be labelled by category, target asset, severity, and expected response. Benign but unusual events, such as a bulk software update or an examination-day traffic surge, should be labelled separately so that the detector learns that unusual does not always mean malicious.

IV. PROPOSED EDGEGUARD-FL ARCHITECTURE

A. Architecture Overview

EdgeGuard-FL has three tiers. Edge nodes sit at departmental switches or gateways, capture flow records, extract features, run the current detector, and train locally. A coordination server schedules training rounds, verifies client identity, aggregates protected updates, and publishes new global models. An administrator console displays alerts, reason codes, model versions, and client-trust indicators. Raw packets and flow records never leave the edge node; only clipped, noised model updates and aggregate statistics are transmitted.

B. Training Round Pipeline

Each round has seven stages: the server selects participating clients; clients download the signed global model; each client trains on recent local flows; the update is clipped to a bounded norm; calibrated noise is added; the update is masked for secure aggregation and uploaded; and the server validates, weights, and aggregates the updates. Models are evaluated on a held-out server-side validation set before deployment, and a model that degrades a regression suite is rejected and the previous version is retained.

C. Trust-Weighted Aggregation

The server maintains a trust score for each client based on update norm, cosine similarity to the median update, historical

consistency, and performance on a small clean validation set. The global update is computed as

$$w(t+1) = w(t) + \Sigma k \, [\, (nk \cdot \tau k) \, / \, \Sigma j \, (nj \cdot \tau j) \,] \cdot \Delta k \qquad (1)$$

where n_k is the number of local samples, $\tau_k \in [0,1]$ is the trust score, and Δ_k is the clipped update of client k. Clients whose similarity to the median falls below a threshold are excluded for the round. Trust parameters and thresholds must be selected using validation data; they are not claimed as experimentally optimised in this paper. Trust scoring supports consistent decisions but cannot prove that a client is honest.

D. Privacy Layer

Before upload, each update is clipped to norm C and perturbed with Gaussian noise:

$$\Delta k' = \Delta k \, / \, \max(1, \, \|\Delta k\|_2 \, / \, C) + N(0, \, \sigma^2 C^2 I) \qquad (2)$$

where sigma is the noise multiplier. A larger sigma gives stronger privacy but may reduce detection quality, so the privacy budget is treated as an explicit tuning parameter rather than a fixed constant. Secure aggregation [5] is proposed as a complementary layer so that the server sees only the sum of masked updates. Differential privacy and secure aggregation protect against different observers and are not interchangeable.

E. Drift Handling

Network behaviour changes with the academic calendar, new devices, and software updates. A drift monitor compares recent feature distributions and anomaly-score distributions against a reference window, using a statistical distance measure, and signals when drift exceeds a configurable bound [13]. Drift can trigger additional training rounds, but it can also indicate an attack, so a drift signal is reported to administrators rather than acted upon silently.

F. Graded Response Policy

Log: the flow is normal or the score is below the alert band. Alert: a suspicious flow is recorded with reason code and confidence band. Rate-limit: a source shows high-confidence volumetric behaviour and its traffic is throttled for a limited time. Isolate: a device is quarantined from the network only for critical patterns and with administrator confirmation. Each outcome records a reason code and model version. User-facing messages should be concise and should not reveal exact thresholds that could help an attacker tune future traffic.

V. DATASET DESIGN AND PREPROCESSING

A. Dataset Sources

The dataset should combine public intrusion benchmarks such as CIC-IDS2017 [8] and UNSW-NB15 [11], synthetic attack traffic generated in an isolated laboratory, and benign traffic collected with permission from the institution. Synthetic data improves coverage but should not dominate the test set because generated attacks may be easier to recognise than real ones. Data collection must respect licensing, privacy, and institutional requirements, and no personal payload content should be retained.

B. Label Schema

Each flow receives labels for benign or malicious status, attack category, severity, and recommended response. A separate label marks benign-but-unusual events. Annotators must distinguish scanning from legitimate network discovery and brute-force attempts from forgotten passwords. An annotation guide should include ordinary bulk transfers, backup jobs, and software updates so that the detector does not equate volume with maliciousness.

C. Feature Extraction

Flows are summarised by duration, packet and byte counts per direction, inter-arrival statistics, protocol and port indicators, TCP flag counts, and short-window rate features. Identifiers such as addresses are hashed or generalised before training, and the feature extractor must be deterministic so that every client computes identical features. Features are normalised using statistics shared in aggregate form rather than raw values.

D. Non-IID Partitioning and Splits

To emulate departments, the dataset is partitioned across simulated clients with deliberately different class mixtures and traffic volumes, for example using a Dirichlet distribution over class proportions. Random flow-level splitting can leak near-duplicate records across subsets and inflate performance, so splits should be made by capture session, attack tool, and collection period. Training data fit parameters, validation data select thresholds, and the test set is reserved for final evaluation. A separate out-of-domain set should contain attack families not seen in training.

E. Class Imbalance and Quality

Benign flows usually outnumber malicious ones by a large margin, so accuracy alone can look high even when attacks are missed. Training may use class weights or resampling, but evaluation should reflect realistic prevalence or clearly state how prevalence affects precision. Annotation agreement, duplicate rate, domain coverage, synthetic-data proportion, and missing fields should be reported. Unclear labels can make a good detector look weak or a weak detector look strong.

VI. DETECTION MODELS

A. Statistical and Rule Baseline

The simplest baseline uses thresholds on connection rates, failed-login counts, and port-fan-out to flag scanning, brute-force, and flooding behaviour. Rules are fast and explainable, and are useful as an early screening stage and as a comparison reference. They require manual tuning, miss novel behaviour, and may raise false alarms on legitimate bulk activity.

B. Classical Machine Learning

Logistic regression, random forests, and gradient-boosted trees can classify flow features and are inexpensive to train. Linear models are naturally compatible with federated averaging, while tree ensembles require different federation strategies and are therefore treated as centralised or local-only baselines. Their limitation is the lack of temporal context across flows.

C. Autoencoder Anomaly Detector

A small autoencoder trained on benign flows reconstructs normal traffic well and reconstructs unfamiliar traffic poorly, so the reconstruction error acts as an anomaly score [7]. This approach does not require labelled attacks and suits federated training because only benign data are needed locally. Its weakness is sensitivity to benign drift, which is why the drift monitor and administrator feedback are part of the design.

D. Lightweight Neural Classifier

A compact multilayer perceptron or one-dimensional convolutional network can learn supervised decision boundaries from labelled flows. Model size must remain small enough for edge devices, and confidence calibration is necessary because networks can be overconfident on

unfamiliar traffic. Evaluation should measure both attack detection and false alarms on benign unusual events.

E. Hybrid Cascade

The proposed cascade runs cheap rule checks first, then the autoencoder score, and finally the supervised classifier for borderline cases. Early exits keep average latency low on benign traffic. Mandatory rules handle critical patterns, and every model update is evaluated against a fixed regression suite with rollback available. The cascade should be benchmarked against each component individually to verify that added complexity produces measurable benefit.

VII. IMPLEMENTATION PLAN

A. Prototype Components

A practical prototype can be built in Python. A flow-capture module reads packet capture files or replays benchmark flows; a feature extractor produces normalised vectors; a local trainer runs on each simulated client; and an aggregation server combines updates. The response policy is a separate module with unit tests for each action. The initial system operates on replayed datasets before any live network or enforcement hardware is connected.

B. Suggested Technology Stack

Python supports data preparation, and scikit-learn can implement classical baselines. A deep-learning framework supports the autoencoder and classifier, while a federated-learning library or a custom simulation loop handles rounds and aggregation. A lightweight API framework can expose the server, and SQLite or structured JSON can store non-sensitive audit metadata. Package versions, random seeds, and model versions should be recorded. Enforcement actions must be disabled during detector evaluation to avoid unintended side effects.

C. Pseudocode

Input: global model w and clients K. For each round, select a subset of clients; send w; each client trains locally on recent flows, clips the update to norm C, adds noise, and uploads the masked result. The server removes outliers using median similarity, computes trust scores, aggregates with trust-weighted averaging, evaluates the candidate model on the validation set and regression suite, deploys it only if checks pass, and stores round metadata without sensitive content.

D. Logging and Monitoring

Operational logs should capture model version, round number, participating clients, risk band, action, latency, and error status. They should avoid storing raw flows or addresses unless a documented requirement and authorisation permit it. Monitoring should track false-positive reports, administrator overrides, client-trust trends, score drift, and communication failures. A sudden decline in detected attacks may indicate a new attack style or a broken capture pipeline rather than an improvement.

E. Failure Handling

If the server is unreachable, edge nodes continue to use the last verified global model and a local rule baseline. If a model fails integrity verification, it is discarded. A sensitive segment should fail safe by preferring alerting over automated isolation when the detector is unavailable. Fallback must never bypass administrator authorisation. Timeouts, malformed flows, and oversized updates should be treated as operational events and tested deliberately.

VIII. EXPERIMENTAL SETUP

A. Research Questions

The evaluation asks: RQ1, does federated training approach the detection quality of centralised training while keeping raw data local? RQ2, how much does differential-privacy noise reduce detection quality? RQ3, how well does trust-weighted aggregation tolerate poisoned clients? RQ4, what are the communication and latency costs on edge hardware? RQ5, how well does the detector generalise to unseen attack families and drifting traffic? These questions connect model quality with system usefulness rather than measuring accuracy alone.

B. Experimental Conditions

All methods should be evaluated on the same source-separated test set. Thresholds are selected on validation data, not on the final test set. Conditions include IID and non-IID partitions, varying numbers of clients, varying fractions of malicious clients, and varying noise multipliers. An ablation removes trust weighting, clipping, noise, secure aggregation, and drift monitoring one at a time. Repeated runs with fixed seeds help identify variation caused by training randomness.

C. Baselines and Reproducibility

The comparison should include rule-only detection, local-only training, centralised training, plain federated averaging, and the full EdgeGuard-FL configuration. Training parameters, model versions, split identifiers, thresholds, and preprocessing steps must be recorded. Confidence intervals should be reported where appropriate, and the limitations of the test corpus must be described clearly.

D. Evaluation Metrics

Attack precision is the fraction of flagged flows that are truly malicious; attack recall is the fraction of malicious flows detected; F1 balances the two. False-positive rate measures benign flows incorrectly flagged. Per-category recall exposes weaknesses hidden by overall scores. Communication cost is measured as bytes per round, and computation as training time and peak memory per client. End-to-end detection latency includes feature extraction, inference, and policy evaluation. Robustness is measured as the change in recall under poisoning, and privacy as the configured privacy budget.

E. Utility-Aware Evaluation

A detector that flags every flow achieves perfect recall but makes the network unusable. Evaluation must therefore measure the trade-off between security and availability, for example through the fraction of legitimate sessions disrupted by rate-limiting or isolation. The operating threshold should be justified by the risk of the protected segment, not solely by the highest F1 score.

IX. RESULTS AND ANALYSIS PLAN

A. Expected Analysis

No dataset was collected and no model was trained for this paper; therefore, this section does not claim measured accuracy. The first analysis will compare local-only, centralised, and federated training on recall and false-positive rate. The second will measure the effect of privacy noise on detection quality. The third will test robustness under poisoned clients. The fourth will evaluate unseen attack families and drift, where generalisation may be most difficult.

B. Error Analysis

False negatives should be grouped by attack category, low-and-slow behaviour, encrypted payloads, and drift period. False positives should be examined for backups, software updates, examination-day surges, and measurement tools. Each error should be assigned a root cause such as missing features, label ambiguity, client heterogeneity, threshold selection, or model limitation. This supports targeted improvement rather than indiscriminate increases in model size.

C. Ablation and Robustness

Ablations remove trust weighting, norm clipping, noise, and drift monitoring in turn. Robustness tests include label-flipping clients, scaled-update clients, backdoor-style clients [10], free-riders, dropped clients, and delayed updates. Tests should also check whether benign traffic from one department is flagged only because it differs from the global average.

D. Results Table Policy

The final report should contain real computed values for precision, recall, false-positive rate, communication cost, and latency. Until experiments are executed, table cells should remain marked as unavailable rather than filled with invented values. Results should include per-category performance and uncertainty estimates. A small benchmark can reveal implementation defects but cannot establish production safety.

X. SECURITY, PRIVACY, AND ETHICS

A. Defence in Depth

EdgeGuard-FL is a detection layer, not a complete security boundary. Networks still require segmentation, patching, strong authentication, and least-privilege access. The learned model must not be able to change firewall policy by itself; enforcement actions pass through an authorisation workflow. Model files are signed so that edge nodes accept only verified models.

B. Privacy-Preserving Telemetry

By default, logs should record decision codes, aggregate counts, hashes, and latency rather than raw flows or user identities. Access to diagnostic samples should be role-restricted, time-limited, and governed by retention policies. Federated learning reduces raw-data movement but does not by itself guarantee privacy, because updates can leak information [12]; the clipping, noise, and secure-aggregation layers are therefore required parts of the design.

C. Fairness and Coverage

A detector trained mainly on traffic from large departments may perform poorly on small ones, and may over-flag devices or users whose behaviour is atypical, such as students running legitimate security exercises. Evaluation should report performance by department and device type, and administrators should have a clear route to contest and correct false alarms.

D. Responsible Testing

A research prototype should use isolated laboratory traffic and replayed datasets. It should not scan or attack third-party systems or live campus services without authorisation, and it should not publish operational details that expose real credentials or infrastructure. The focus of the work is defensive detection.

XI. DISCUSSION AND APPLICATIONS

A. Design Trade-Offs

Stronger privacy noise protects participants but reduces model accuracy. Aggressive outlier rejection protects against poisoning but may exclude honest clients with unusual traffic. Frequent rounds improve adaptation but increase bandwidth and energy use. The staged design makes these trade-offs explicit so that each department can select a profile suited to its risk: a laboratory with research servers may accept more noise reduction in exchange for stricter monitoring, while a library kiosk network may need minimal overhead.

B. Why Federation Helps

A model trained only on one department sees a narrow slice of attacker behaviour. Federation lets the nodes learn from one another's experience without exchanging traffic. A scan seen first in the hostel network can improve detection in the laboratory before the same tool reaches it. The benefit depends on partition heterogeneity and must be measured rather than assumed.

C. Real-World Applications

Potential applications include campus-wide intrusion monitoring, hospital networks that cannot share patient-related traffic, small-office branch networks of a company, smart-factory sensor networks, and managed-service providers who protect several customers without pooling their data. In each setting, the model remains an advisory component and response actions are authorised independently.

D. Limitations

The framework is conceptual and has not been validated on a collected benchmark. Privacy and trust parameters require calibration. Encrypted traffic limits visible features, and novel multi-stage attacks may evade flow-level detection. Differential privacy offers a statistical guarantee that depends on the chosen budget and on correct implementation. Simulated departments are only an approximation of real institutional diversity.

XII. DETAILED USE-CASE ANALYSIS

A. Smart Classroom Network

Smart boards, projectors, and lecture-capture devices have predictable traffic and are rarely patched. An anomaly detector trained on their normal behaviour can flag unexpected outbound connections or scanning activity. Because classroom devices are limited in function, strict allow-lists complement the learned model, and an alert is preferred over automatic isolation during teaching hours.

B. Hostel and Student Wi-Fi

Student networks combine diverse devices, peer-to-peer applications, and legitimate bulk downloads. False positives are costly here because they disrupt service for many users, so rate-limiting is preferred over isolation and a clear appeal process is required. Privacy expectations are high, which supports the use of local training and minimal logging.

C. Laboratory and Research Servers

Laboratories host servers with high-value data and often run unusual software and experiments. They are good candidates for stronger monitoring but also produce unusual traffic that can appear anomalous. The framework allows laboratory-specific calibration and a feedback channel through which administrators mark expected experiments as benign.

D. Surveillance and Access-Control Devices

Cameras and access terminals are common targets for botnet recruitment. Beaconing and unexpected destinations are the main indicators. Because these devices cannot host a detector, monitoring occurs at the gateway, and isolation of a suspected device is a high-impact action requiring confirmation from security staff.

XIII. TESTING STRATEGY AND QUALITY ASSURANCE

A. Unit Tests

Unit tests should verify feature extraction, clipping, noise addition, aggregation arithmetic, trust-score calculation, and every response outcome. Tests should include empty flows, malformed records, extreme values, and missing fields. Unit tests do not establish security, but they prevent basic implementation defects from invalidating the evaluation.

B. Integration Tests

Integration tests should trace a round from client training through aggregation to model deployment. A malformed or oversized update must not crash the server or bypass validation. A model that fails the regression suite must not be deployed. Isolation requests must be rejected when authorisation fails, even if the detector score is high.

C. Regression and Acceptance Criteria

A fixed regression suite should contain confirmed false negatives, false positives, and previously observed evasion variants. Acceptance criteria such as target recall, maximum false-positive rate, latency budget, and communication budget should be set before final testing and must not be changed after seeing the results. If a criterion is not met, the report should show the actual value and explain the cause.

XIV. FINAL PROJECT IMPLEMENTATION ROADMAP

A. Phase One: Requirements and Threat Modelling

Identify the target network, assets, device classes, and actions that must be protected. Draw a data-flow diagram showing which data stay at the edge and which cross the trust boundary. Write the threat taxonomy and response outcomes before data collection.

B. Phase Two: Baseline and Dataset

Build a replay-based pipeline using a public benchmark, create the non-IID client partitions, and implement the rule baseline and a centralised classifier. Freeze source-separated splits before comparing more advanced methods.

C. Phase Three: Federated Training and Defences

Implement federated averaging, then add clipping, noise, and trust-weighted aggregation. Evaluate under injected poisoned clients. Add the autoencoder and drift monitor if resources allow.

D. Phase Four: Evaluation and Demonstration

Report precision, recall, false-positive rate, communication cost, latency, and resource use with error analysis. A demonstration dashboard can show client participation, trust indicators, alerts, reason codes, and model versions. The final report should separate measured outcomes from proposed features and avoid unsupported claims about complete security.

XV. SUMMARY OF DESIGN DECISIONS

A. Design Summary

Table I summarises the core design choices proposed for EdgeGuard-FL. These choices should be verified during implementation and may be adjusted based on measured results.

TABLE I. SUMMARY OF PROPOSED DESIGN DECISIONS

Component	Design choice	Reason
Data locality	Raw flows stay on the edge node	Privacy and lower bandwidth
Local models	Rules, autoencoder, light classifier	Fits edge hardware
Update privacy	Clipping, Gaussian noise, secure aggregation	Limits update leakage
Aggregation	Trust-weighted, median-filtered	Tolerates poisoned clients
Drift	Distribution monitor with alerts	Adapts to changing traffic
Response	Log, alert, rate-limit, isolate	Action matches risk
Enforcement	Independent authorisation	Detector cannot grant access

XVI. FUTURE WORK

A. Benchmark Development

The first priority is a benchmark with department-style partitions, consistent labels, and realistic benign-but-unusual traffic. A public benchmark should document licensing, privacy review, synthetic-data proportion, and known limitations.

B. Personalisation and Adaptive Privacy

Future versions may fine-tune a personalised local model per department and adapt noise levels according to the sensitivity of the segment. Adaptation must be bounded and tested so that attackers cannot manipulate the policy.

C. Hardware-Aware Compression

Quantisation, pruning, and distillation may allow stronger models on single-board computers. The accuracy loss from compression must be measured and reported.

D. Explainability and Continuous Evaluation

The detector should provide concise reason codes such as scan pattern, authentication burst, or beaconing, supporting debugging without exposing thresholds that enable evasion. Model updates should run in shadow mode before gradual rollout with rollback available.

XVII. CONCLUSION

EdgeGuard-FL proposes a privacy-conscious approach to collaborative intrusion detection in smart campus networks. Its central idea is to let edge nodes learn from one another's experience without sharing raw traffic. Local detection, clipped and noised updates, trust-weighted aggregation, drift monitoring, and a graded response policy work together to reduce privacy exposure and the effect of faulty or malicious participants. The framework also requires independent authorisation for enforcement actions, careful logging, source-separated evaluation, and explicit measurement of communication and latency cost. The approach is modular and suitable for a final-year CSE prototype that begins with

replayed benchmark data and gradually adds federated defences.

Because the framework has not yet been experimentally validated, its effectiveness remains a research question rather than a proven result. A completed implementation should use a well-labelled benchmark, report actual metrics, analyse errors, and state its limitations clearly. Secure collaborative detection requires layered controls and continuous evaluation; a learned detector is valuable, but it cannot replace network segmentation, patching, and access control.

ACKNOWLEDGMENT

The authors acknowledge the guidance and support of the faculty guide and the Department of Computer Science and Engineering, Kings College of Engineering, Punalkulam, Tamil Nadu, India. The academic environment and available learning resources supported the preparation of this proposed research framework.

REFERENCES

[1] H. B. McMahan et al., "Communication-Efficient Learning of Deep Networks from Decentralized Data," in Proc. AISTATS, PMLR vol. 54, pp. 1273–1282, 2017.

[2] T. Li et al., "Federated Optimization in Heterogeneous Networks," in Proc. Machine Learning and Systems (MLSys), 2020.

[3] P. Kairouz et al., "Advances and Open Problems in Federated Learning," Foundations and Trends in Machine Learning, vol. 14, no. 1–2, pp. 1–210, 2021.

[4] C. Dwork and A. Roth, "The Algorithmic Foundations of Differential Privacy," Foundations and Trends in Theoretical Computer Science, vol. 9, no. 3–4, pp. 211–407, 2014.

[5] K. Bonawitz et al., "Practical Secure Aggregation for Privacy-Preserving Machine Learning," in Proc. ACM CCS, pp. 1175–1191, 2017.

[6] M. Abadi et al., "Deep Learning with Differential Privacy," in Proc. ACM CCS, pp. 308–318, 2016.

[7] Y. Mirsky et al., "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," in Proc. NDSS, 2018.

[8] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in Proc. ICISSP, 2018.

[9] P. Blanchard et al., "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent," in Advances in Neural Information Processing Systems, vol. 30, 2017.

[10] E. Bagdasaryan et al., "How To Backdoor Federated Learning," in Proc. AISTATS, PMLR vol. 108, 2020.

[11] N. Moustafa and J. Slay, "UNSW-NB15: A Comprehensive Data Set for Network Intrusion Detection Systems," in Proc. Military Communications and Information Systems Conference (MilCIS), 2015.

[12] L. Zhu, Z. Liu, and S. Han, "Deep Leakage from Gradients," in Advances in Neural Information Processing Systems, vol. 32, 2019.

[13] J. Gama et al., "A Survey on Concept Drift Adaptation," ACM Computing Surveys, vol. 46, no. 4, 2014.

[14] National Institute of Standards and Technology, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, 2023.

[15] M. Mitchell et al., "Model Cards for Model Reporting," in Proc. Conference on Fairness, Accountability, and Transparency, pp. 220–229, 2019.

[16] T. Gebru et al., "Datasheets for Datasets," Communications of the ACM, vol. 64, no. 12, pp. 86–92, 2021.