

**International Journal of Information and Computer Security**

ISSN online: 1744-1773 - ISSN print: 1744-1765

<https://www.inderscience.com/ijics>

---

**Enhancing network security using FLAT for classifying intrusion attacks**

Priyanka Hanumanthappa, Ananya Menon, Manjula Gururaj Rao

**DOI:** [10.1504/IJICS.2025.10071904](https://doi.org/10.1504/IJICS.2025.10071904)

**Article History:**

|                   |                   |
|-------------------|-------------------|
| Received:         | 07 May 2024       |
| Last revised:     | 21 October 2024   |
| Accepted:         | 05 November 2024  |
| Published online: | 05 September 2025 |

---

## Enhancing network security using FLAT for classifying intrusion attacks

---

Priyanka Hanumanthappa\* and  
Ananya Menon

Department of CSE,  
PES University,  
Bengaluru, India  
Email: drpriyankahsachin@gmail.com  
Email: ananya4am@gmail.com  
\*Corresponding author

Manjula Gururaj Rao

Department of ISE,  
PES University,  
Bengaluru, India  
and  
NMAMIT,  
Kalya, India  
and  
NITTE,  
Karkala, India  
Email: dr.manjulagururajh@gmail.com

**Abstract:** Concerns over the security of sensitive data handled by internet of things (IoT) devices are being raised by their rising presence in many facets of our everyday life. Network security becomes critical when smart homes and industrial settings are used. This research presents a novel method for intrusion detection in network communication that combines federated learning approaches with conventional machine learning and deep learning algorithms. Federated learning with adversarial training (FLAT), the suggested approach, mixes decentralised (post-FLAT) and centralised (pre-FLAT) methods in a novel way to improve security measures. Adversarial training is incorporated into the model to provide an extra line of defence against possible attacks. With roughly 92% accuracy, precision, recall, and F1-score for the man-in-the-middle attack dataset, the FLAT method performs competitively. Additionally, for the active wiretap attack dataset, FLAT exhibits a commendable accuracy (93%), precision (95%), recall (87%), and F1-score (91%). This research contributes to advancing security measures in IoT environments by introducing FLAT as a powerful tool for intrusion detection.

**Keywords:** FLAT; intrusion attacks; network security.

**Reference** to this paper should be made as follows: Hanumanthappa, P., Menon, A. and Rao, M.G. (2025) 'Enhancing network security using FLAT for classifying intrusion attacks', *Int. J. Information and Computer Security*, Vol. 28, No. 1, pp.103–143.

**Biographical notes:** Priyanka Hanumanthappa has 8+ years of experience in teaching Computer Science and Engineering. She completed her PhD in Computer Science and Engineering at VTU, Belgaum. She is currently working as an Associate Professor at PES University in Bangalore, Karnataka, India. Her areas of interest include AI, ML, DL, image processing, cloud computing, smart city, health care sectors, energy consumption, IoT, and data analytics. She has several publications in national and international journals and conferences.

Ananya Menon is a final year student of Computer Science and Engineering at PES University, Bangalore, she currently pursuing her Bachelor of Technology. Her passion lies at the intersection of machine learning, artificial intelligence, cloud, and data analytics. With a focus on cutting edge projects, she has actively contributed to various research domains, such as meta learning, OpenCV and image processing, natural language processing, and graph data analysis.

Manjula Gururaj Rao has 20+ years of experience in teaching Computer Science and Engineering. She completed her PhD in Computer Science and Engineering at Jain University, Bangalore. She is currently working as an Associate Professor at NMAMIT, NITTE in Udupi, Karnataka, India. Her areas of interest include AI, ML, DL, image processing, cloud computing, smart city, health care sectors, energy consumption, IoT, and data analytics. She has several publications in national and international journals.

## 1 Introduction

In IoT networks, network attacks take advantage of weaknesses caused by resource constraints on devices, unreliable connectivity, insufficient security protocols, and the extensive use of IoT devices (Trajanovski and Zhang, 2021; Li et al., 2018). IoT networks are vulnerable to assaults due to a variety of factors, including inadequate security mechanisms, insecure communication, and scarce resources (Jiyal et al., 2022; Muhammad et al., 2023; Yin et al., 2022). Man-in-the-middle, device spoofing, and denial-of-service attacks are examples of common attacks. Network attacks can take the form of communication interceptions, denial of service attacks (overloading a network with excessive resources), or network sniffing – the act of intercepting and examining network traffic to obtain private data. For early detection, response planning, risk assessment, and ongoing IoT security enhancement, the categorisation of network assaults is essential (Xu, 2020; Hussain et al., 2021). The paper proposes a comprehensive framework for network attack classification, emphasising the application of FLAT in IoT environments. The study compares machine learning algorithms and FLAT methodology, evaluating client performance during attack detection in both centralised and decentralised setups. Machine learning and deep learning algorithms are employed for attack and benign location detection, with FLAT enhancing privacy by training models on decentralised, resource-constrained clients without sharing raw data. The integration of adversarial training further bolsters the model's resilience against potential attacks (Pontara da Costa, 2022).

IoT devices are susceptible to weak security measures, weak authentication, and authorisation are often resource-constrained, and have limited processing power and

memory (Muhammad et al., 2023). The research provides a thorough analysis of device and client performance pre- and post-FLAT integration, contributing to the development of a robust and privacy-aware network attack detection system. The paper aims to detect and resolve network attacks using a combination of the following technologies:

- 1 Predefined data collection which describes network attacks taking place in an IoT environment.
- 2 Benign and attack label detection using a collection of machine and deep learning algorithms.
- 3 Federated learning over ten rounds with 100 clients employed to study IoT network attack detection in a decentralised fashion and enhance traditional algorithms' privacy and security.
- 4 To test the effect of adding a federated method to sort attack vectors.
- 5 FLAT, with adversarial samples generated using the fast gradient sign method (FGSM) in the federated learning process, aimed at improving the robustness of the model against adversarial attacks. A comparative analysis of evaluation metrics and individual device and client performance of centralised and decentralised/federated techniques, pre- and post-FLAT integration.
- 6 Visualisation of the results using inbuilt libraries such as Matplotlib, Seaborn, etc.

The proposed technologies are to be used to make a comparative analysis of existing technologies and base a fresh strategy based on the previously employed ideas that solve the previously mentioned challenges. Our aim is to design a novel method for intrusion detection in network communication using federated learning with adversarial training and deep learning techniques.

## 2 Related work

In Mothukuri et al. (2022), the study makes use of a dataset from a Modbus-based network and employs a decentralised federated learning strategy using ensemble learning to find anomalies. The approach uses long-short-term memory (LSTM), and gated recurrent unit (GRU) which are improved versions of recurrent neural network (RNN) on the Modbus dataset. Each client node engages in federated training, which is then sent to the global server for aggregation for a predetermined number of rounds at predetermined intervals. Each device receives a copy of the global model. Ensembler combines the output of several global models in different rounds to detect which attack is based on the highest probability. The strategy employed has the benefits of employing the LSTM and GRU models and implementing federated learning. However, deploying this methodology loses the merits of using Adversarial Training to increase the hardness of attacks occurring under high noise (Xu, 2020).

In Chauhan and Atulkar (2021), one popular public dataset with well-known BotNet attacks is the BOT-IoT dataset. The dataset is reduced, and then the tree-based ensemble classifier is trained and tested. Finally, performance metrics are calculated. Feature extraction has been done to extract the ten best features and normalisation of the extracted features using min-max scaling. Five distinct models are used: extra trees (ET)

model, gradient boosting, extreme gradient boosting machine (XGB), random forest, and lightweight gradient boosting machine. The strategy employed here forms a strong foundation for the implementation of classifiers on an IoT dataset. However, the lack of employment of a Federated methodology drops the accuracy and reliability of the overall methodology significantly.

In Singh et al. (2021), UNSW-NB15 has been generated in a synthetic environment which is a combination of modern attacks. It has been discovered that data sent to the cloud undergoes pre-processing and computation, which delays the detection of the attack (Zou et al., 2022). The fog layer is introduced that helps save computational time and reduces execution time thus helping in faster identification of an attack to the IoT network (Sun et al., 2022). Data is transferred to the fog layer instead of the cloud layer to reduce execution time. An alert message is sent to the administrator while the IoT network remains unaffected. The convolutional neural network (CNN) algorithm provides higher accuracy with low positive and negative rates (Singh et al., 2021). This strategy was selected for its application of distribution to reduce latency and improve computation times. The choice of using CNN algorithms was also used here which showed greater promise than the artificial neural network (ANN) or RNN used previously. The solution could have used enhancements on the computation layer to improve accuracies, with, for example, adversarial training.

In Neto et al. (2022), this experiment made use of the CICDDOS2019 dataset, which contains data that has been subjected to six common distributed denial-of-service (DDOS) attacks. Portmap, network basic input/output system (NetBIOS), lightweight directory access protocol (LDAP), user datagram protocol (UDP), and UDP lag. The cloud requests 'n' tenants. To simplify the learning process, tenants scale data locally, and scaling models are not exchanged. Tenants update local versions of the DDOS detection model based on cloud inputs and calculate new parameters using their local data (Kousar et al., 2021; Ling et al., 2022). Trained parameters are extracted and shared in every round where 'n' tenants provide the cloud with a new version of the model parameters. A centralised aggregation procedure is used. The second part is based on calculating new parameters for DDOS detection based on parameters provided by tenants. After the training phase is complete, tenants can access aggregate model parameters in the cloud. Current tenants can update their models at the end of every training round, or if their model version is lost during the process. The concept implemented in this project is invested in distribution. By distributing the computation of the tenants and the centralised model, the computation can be split and nodes can act as redundant nodes giving this a significant role in redundancy management and latency limitation (Sahingoz et al., 2021). The proposed system identifies the architectural flaws that resulted in the Bruteforce attack and determines whether any open ports could be used to exploit the system. It also suggests developing a vulnerability scanner tool for IoT devices that can be run in the system to detect flaws. DOS and DDOS attacks are carried out to determine whether the system is vulnerable and whether the data stored in IoT devices is encrypted or not. The strategy implemented is to mitigate inherent architectural flaws and reduce the threat of attack vectors in the foundational hardware and software. This however does not build a solution for the existing threat vectors of the present.

### 3 Challenges

Some key considerations and challenges in ensuring network security in IoT devices are:

- 1 Limited resources: Since many IoT devices operate with limited computing resources, there are challenges in implementing robust security measures due to the need for lightweight protocols (You et al., 2022). This designated a need for distribution of computing and security. Thereby warranting a strategy to push security requirements away from devices, to the upper layers.
- 2 Heterogeneous and diverse ecosystem: The IoT device ecosystem comprises heterogeneous devices with different manufacturers and different device architectures. Interoperability and ensuring standardised security measures in this diverse landscape pose a major challenge (Shahid et al., 2021). Creation of separate upper layer protocols or nodes to handle diverse inputs will help solving this issue. Implementation of distributed computing frameworks for differing requirements and specifications helps cater to varying needs.
- 3 Challenges in maintaining data privacy: IoT devices often collect sensitive information and traditional centralised modes of attack detection promote a single point of attack. Moreover, if the central server is compromised, all the user data becomes vulnerable (Korkmaz et al., 2022). Creating a distributed framework using technology such as federated learning will help create redundant nodes preventing a full scale security failure.
- 4 IoT environments are dynamic, and the normal behaviour of devices can change over time. Traditional anomaly detection models may struggle to adapt to such changes (Song et al., 2018). The implementation of models based on learning will be able to adapt to changes in the environment or expected attack vectors over time (Niknami and Wu, 2022; Bebin Josey and Misbha, 2022).
- 5 Adversarial examples crafted for one IoT device may transfer to similar devices, leading to security risks across a network of IoT devices (Song et al., 2018). Creating slightly varied ensemble based nodes throughout the server will help reduce such weaknesses in the detection algorithm.
- 6 New types of attacks and threats continually emerge, requiring detection models to adapt and learn from evolving patterns.

### 4 Datasets

The dataset under examination corresponds to the Kitsune network attack dataset (Beutel et al., 2020), which encompasses nine instances of network attacks observed within an IoT device network. These incidents were recorded by either an IP-based commercial surveillance system or an IoT-enabled network. The dataset includes information for each attack, comprising a dataset and a corresponding label vector, both stored in comma separated value (.csv) file format. Additionally, users have access to a packet capture (pcap) file containing the original network capture data, should they wish to undertake their feature engineering.

OS scan, fuzzing, video injection, address resolution protocol man-in-the-middle (ARP-MitM), active wiretap, simple service discovery protocol (SSDP) flood, SYN DOS, secure socket layer (SSL) renegotiation, and Mirai Botnet are among the attacks contained in the dataset. Each dataset follows a specific structure: clean network traffic was recorded for the initial one million packets, after which the designated attack was executed. Notably, each dataset is characterised by a unique number of rows, representing individual packets, and is organised into 115 columns, devoid of headers.

In the research paper titled ‘Kitsune: an ensemble of autoencoders for online network intrusion detection’, Beutel et al. (2020) propose an innovative framework that performs network intrusion detection through an ensemble of autoencoders. This ensemble is named ‘Kitsune’. The dataset used in Beutel et al. (2020) is generated through an AfterImage feature extractor, which is a powerful framework designed for observing millions of numerical data streams and is primarily used for extracting relevant features from network traffic data. It operates on numerical data streams, such as network packets, and computes feature vectors that capture essential information about the network behaviour at specific moments. The AfterImage feature extractor plays a crucial role in the Kitsune network attack dataset, which is used for network intrusion detection and is subsequently used as a dataset for the approach of classifying network attacks using FLAT.

By extracting features from network packets, it enables the detection of abnormal patterns associated with network attacks. The AfterImage feature extractor computes 115 features for each packet. These features capture various aspects of the network, including packet sizes, inter-arrival times, protocol distribution, host behaviour metrics, traffic patterns, flow characteristics, and indicators of anomalies. A detailed explanation of the features extracted from network packets using the AfterImage feature extractor in the Kitsune network attack dataset are:

- 1 *Context and purpose:* Each row in the CSV corresponds to a captured packet, arranged chronologically. The feature vector for each packet provides recent temporal statistics that describe the context of the packet’s channel and the communicating parties.
- 2 *Behavioural snapshot:* When a packet arrives, the AfterImage feature extractor captures a behavioural snapshot of the hosts and protocols involved in the communication. This snapshot consists of 115 traffic statistics, which offer a glimpse into the general behaviour of the packet’s sender and the traffic between the packet’s sender and receiver.
- 3 *Traffic statistics:* These summarise various aspects of the network traffic such as:
  - *Originating from source MAC and IP (SrcMACIP):* Captures behaviour related to the packet’s source.
  - *Originating from source IP (SrcIP):* Focuses on traffic from the packet’s source IP.
  - *Sent between source and destination IPs (channel):* Describes communication between specific IP addresses.
  - *Sent between source and destination transmission control protocol (TCP)/UDP socket (socket):* Details communication via specific sockets.

- 4 *Features per time window*: The feature extraction process considers a time window ( $\lambda$ ) for each packet. Five time-damped windows are used and within each window, features are extracted, providing a temporal context for the packet's behaviour.

The five time-damped windows are:

- approximately 100 ms
  - approximately 500 ms
  - approximately 1.5 seconds
  - approximately 10 seconds
  - approximately 1 minute.
- 5 *Zeroed features*: Not every packet applies to every channel type. For instance, if a packet does not contain a TCP or UDP datagram, certain features are zeroed.
  - 6 Pcap files are then converted to csv files for further use and analysis.

The 115 features extracted from network packets play a crucial role in enhancing network attack classification using FLAT by providing a rich representation of network behaviour, capturing various aspects such as packet sizes, inter-arrival times, protocol distribution, and more. This feature richness allows FLAT models to learn discriminative patterns associated with different types of network traffic. Furthermore, capturing traffic statistics from five-time windows of varying lengths allows the FLAT model to consider historical context and identify patterns in network behaviour over time. By analysing these traffic statistics and their temporal patterns, the FLAT algorithm can learn to distinguish between normal network behaviour and characteristics associated with different attack types.

In the context of MitM and active wiretapping attacks, here's how traffic statistics and features in the Kitsune network attack dataset can help distinguish between the two.

In a MitM attack, an attacker secretly relays communication between two parties, allowing them to eavesdrop and potentially modify the data. Therefore, there might be an increased traffic volume compared to normal communication between the two parties. This is because the attacker is essentially duplicating the traffic, sending it to both the intended recipient and themselves. Also, there is the situation of an attacker's IP address appearing as a source or destination in some communication, even though it was not originally involved. Unusual ports might be shown as being used for communication, especially if the attacker is trying to hijack established connections. When it comes to temporal characteristics, the attacker's activity might show specific time windows of increased traffic corresponding to when they are actively manipulating communication.

In an active wiretap attack, it might show similar overall traffic volume to normal communication, but with some anomalies in the direction. The attacker might be receiving copies of traffic not intended for them. This can be a key indicator for attack detection since the source and destination IP addresses and the port usage might remain consistent with normal communication, as the attacker is simply listening in. There might not be significant changes in temporal patterns compared to normal communication either.

#### 4.1 Kitsune network attacks – man-in-the-middle attack dataset

ARP-MitM involves the manipulation of a target device's address resolution protocol (ARP) cache within a local network. By tampering with the ARP cache, the attacker gains the ability to obstruct or intercept communication between two network entities, potentially resulting in unauthorised access or network interception. To provide context, ARP serves as a protocol in local area networks, utilised for mapping an IP address to the physical machine address of a system. The ARP cache specifically retains the mappings of IP addresses to MAC addresses. Being a stateless protocol is the main issue with ARP. This indicates that devices do not check the origin of ARP messages they get. By pretending to be the genuine device and sending phony ARP answers (spoofed MAC addresses), an attacker can take advantage of this. This is why it is challenging to harden ARP itself. Firstly, the ARP Protocol is simple. ARP's design prioritises ease of use and effectiveness. Complicated authentication methods would add to overhead and might cause network communication to lag. ARP messages are typically broadcast on the local network. It would be computationally and resource-intensive for network devices to implement source verification for each broadcast message. The dataset associated with this attack comprises approximately 2,504,266 rows and 115 columns. Simultaneously, the label vector includes 1,358,995 benign (0) points and 1,145,272 attacks (1) points. ARP-MitM attacks can be identified using the Kitsune network attack dataset's extensive feature set by looking for discrepancies between observed and predicted network behaviour.

Monitoring ARP traffic-features pertaining to ARP activity in the network are included in the dataset. The system can detect unusual patterns that could point to attempts of ARP poisoning by examining these features, such as:

- *High frequency of ARP requests:* ARP requests suddenly increasing, especially from unknown sources, may indicate spoofing attempts. This is due to the fact that attackers frequently submit erroneous ARP queries in an attempt to alter network ARP caches.
- *Conflicting MAC addresses:* Spoofing may be shown by discrepancies in the dataset if it records mappings between IP addresses and MAC addresses (as in a genuine ARP cache). An attacker may be sending ARP answers, or fake MAC addresses, pretending to be the genuine device, for instance, if the same IP address is linked to several distinct MAC addresses. This is the fundamental method of ARP poisoning.
- *Examining traffic trends:* Features capturing traffic patterns can also provide clues about potential ARP poisoning.
- *Unexpected communication:* ARP poisoning may be attempted by the attacker if a device begins to communicate with a new IP address that it has never connected with previously and there is unusual ARP activity. An attacker can fool devices into sending their communication to their machine rather than to the intended recipient by sending bogus ARP answers.
- *Communication interrupted:* When devices that have established communication channels suddenly encounter delays or disturbances, it may indicate that an attacker is using ARP poisoning to disrupt the network flow. By manipulating the ARP

cache, the attacker can disrupt the normal routing of packets, leading to communication issues.

#### 4.2 Kitsune network attacks – active wiretap dataset

The term ‘wiretapping’ denotes the illicit interception of data as it traverses through a network, conducted without the consent of the communicating devices. In the context of active wiretapping, an unauthorised entity engages in eavesdropping or intercepts communication between devices within a network. This unauthorised access is achieved through techniques such as packet sniffing, where the attacker seizes and analyses network packets to extract sensitive information. Traditional Wiretapping involves passively listening in on a communication channel by physically attaching to a network cable. It is essentially similar to eavesdropping on a conversation. Active wiretapping involves more active measures. Attackers can use denial-of-service attacks, introduce devices into the network connection, or even take advantage of network protocol flaws to obstruct normal traffic flow and subsequently harvest data that is exposed. The associated dataset encompasses approximately 2,278,688 rows, each representing a network packet, and is organised into 115 columns. Correspondingly, the label vector comprises 1,355,473 benign (0) points and 923,216 attacks (1) points. The Kitsune network attacks dataset does not target a specific concrete type of active wiretapping attack. It is designed for generic active wiretapping identification. It categorises network packets as benign or malicious, not pinpointing the specific technique used by the attacker. A generic active wiretap attack does not necessarily refer to a single specific technique, but rather a broad category of attacks that involve actively intercepting data transmissions on a network. It is an umbrella term encompassing various methods attackers can use to eavesdrop on communication channels.

The following information present in the dataset helps in the identification of generic active wiretapping attacks.

- *Packet size and volume:* Abnormal increases in the size of a packet or in overall traffic volume could be signs that an attacker is sniffing traffic on the same network segment and is collecting more information than usual.
- *Origin and destination of packets:* Unusual communication patterns may indicate the attempt by unauthorised equipment to intercept traffic. Excessive amounts of traffic coming from unidentified or suspicious IP addresses may be cause for concern.

Apart from the properties of individual packets, the dataset also includes statistical data regarding network traffic trends. This consists of:

- *Modifications to packet arrival rates:* Abrupt or severe departures from the typical baseline traffic rate may be a sign of manipulation or interference from an ongoing eavesdropping operation.
- *Packet loss rates:* An attacker may be performing a DoS attack to obstruct communication and may seize exposed data if there are unexplained spikes in packet loss.

## 5 Methodology

The methodology summarised in Algorithm 1, begins by subjecting the datasets to the following supervised machine learning and deep learning models: random forest, logistic regression (LR), extreme gradient boosting (XGBoost), light gradient boosting machine (LGBM), and the deep learning models: ANN, CNN, and the evaluation metrics: accuracy, precision, recall, and F1-score. Traditional machine learning models are further optimised by introducing the FLAT methodology, which not only provides a secure IoT attack detection framework but also involves sharing model updates among clients and the server rather than the raw data and ensures model training while maintaining data privacy and security; it is especially useful in scenarios where centralised training is not feasible (Korkmaz et al., 2022). The objective is to see if FLAT offers a competitive performance to a traditional centralised training scenario. Figure 1 depicts the high-level architecture diagram of the overall network attack classification scenario using centralised attack classification and the FLAT methodology. Figure 2 represents the logic behind dataset partitioning among clients during adversarial training and testing and Figure 3 represents the flow diagram of the FLAT methodology which includes federated learning along with adversarial training, with adversarial samples being generated using the FGSM. The dataset is partitioned among clients. The server only contains the dataset for testing. The server initially sends a copy of the global model to all clients, which contains the dataset for training. Table 1 gives details on the federated learning configuration during FLAT methodology.

### Algorithm 1 FLAT methodology

---

**Data:** Training dataset, testing dataset, number of clients ( $N$ ), and number of training rounds ( $R$ )

**Result:** Global model with improved attack classification

- 1    **for**  $r$  1 to  $R$  **do**
  - 2     Randomly select 10 clients for training from the total  $N$  clients (with no repetitions);
  - 3     Train each client locally on its train dataset for one epoch using normal and adversarial samples;
  - 4     Generate adversarial samples using the fast gradient sign method;
  - 5     Compute updates based on the differences between the server's global model parameters and the local model parameters of the selected clients;
  - 6     Send updates from all 10 clients to the server;
  - 7     Use the FedAvg strategy to aggregate updates on the server and update the global model;
  - 8     Randomly select 25 clients for validation from the total  $N$  clients;
  - 9     Validate the performance of the updated global model on the local validation datasets of 25 clients after every training round;
  - 10    Send the aggregated accuracy, precision, recall, and F1-score from all 25 validation clients to the server;
  - 11    Monitor the performance of the global model and improve it for the next training round based on the metrics received;
  - 12   **end**
  - 13   Test the final updated global model on the server's test dataset;
  - 14   Calculate the overall accuracy, precision, recall, and F1-score on the test dataset;
-

**Figure 1** High-level architecture diagram of the proposed pre- and post-FLAT network attack classification (see online version for colours)

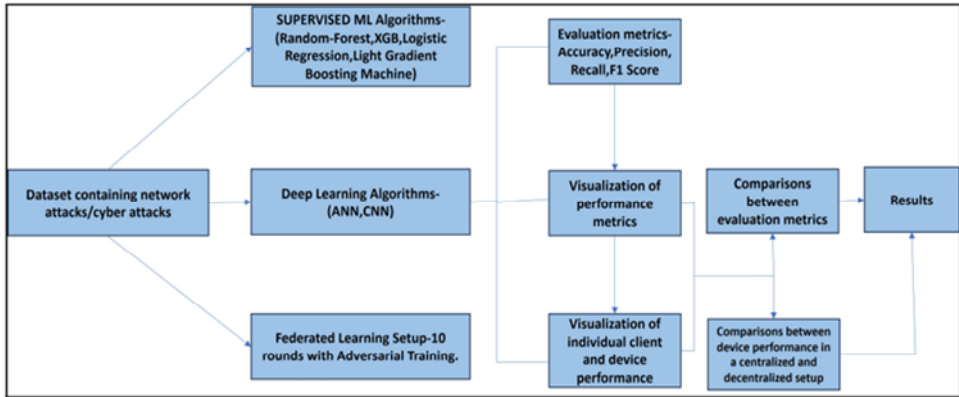


**Figure 2** Dataset partitioning logic during the FLAT process (see online version for colours)

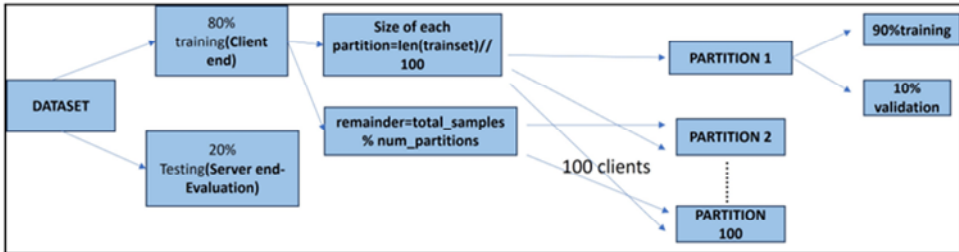
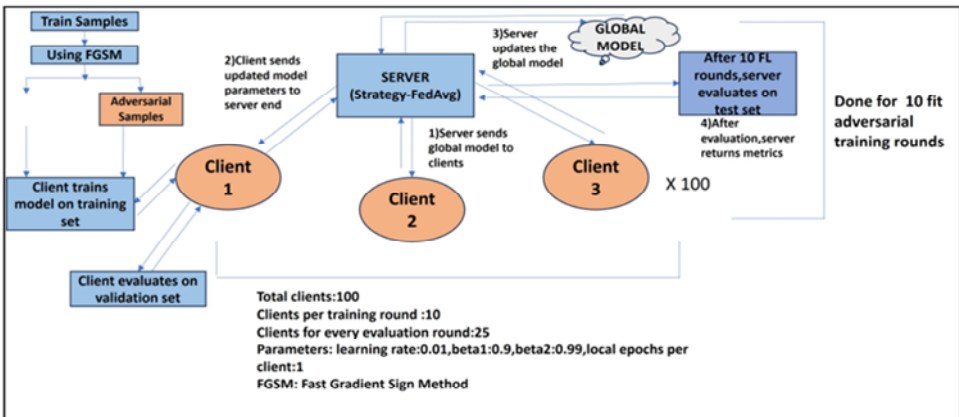


**Figure 3** Flowchart of proposed method (see online version for colours)



There are 100 clients in total. Federated learning undergoes a certain number of training rounds, in this case, there are ten adversarial training rounds. For every training round, ten clients are chosen at random for every round of training-however, it is ensured that no clients are repeated and all 100 clients are utilised. The random selection of ten clients out of a total of 100 for each adversarial training round adds an element of diversity and

randomness to the training process. This is to ensure that the global model is exposed to a variety of data distributions and characteristics from different clients. The clients train on their local train dataset for one epoch and send model updates based on the parameters given in Table 2.

**Table 1** Federated learning configuration

|                                                 |                        |
|-------------------------------------------------|------------------------|
| Total clients                                   | 100                    |
| Training rounds                                 | 10                     |
| Clients per fit training round                  | 10                     |
| Clients per validation round                    | 25                     |
| Client Selection                                | Random (no repetition) |
| Total epochs across all clients during training | 100                    |

**Table 2** Federated learning with adversarial training (FLAT) parameters

| <i>Parameter</i>        | <i>Value</i> |
|-------------------------|--------------|
| Learning rate           | 0.01         |
| Beta 1 (Adam optimiser) | 0.9          |
| Beta 2 (Adam optimiser) | 0.999        |
| Local client epoch      | 1            |
| Epsilon (perturbation)  | 0.01         |

Adversarial samples are generated using the FGSM which adds a small perturbation to the original train samples in the direction of the gradient of the loss concerning the input train samples (Chen and Zhang, 2020). The model is trained by the clients both on clean and adversarial samples, increasing its robustness to attacks and leading to an improvement in attack classification (Song et al., 2018). The training function is used by every client to locally train on one epoch. The loss function or criterion used is cross-entropy loss. The updates which are computed by taking a difference between the server's deployed global model parameters and the clients' local model parameters are sent to the server from all ten clients and using the FedAvg strategy, the server performs an aggregation on the client model parameters and updates the global model (Neto et al., 2022). After updating, 25 clients are chosen at random to test the updated global model on their local validation datasets. The aggregated accuracy, precision, recall, and F1-score from all 25 validation datasets is sent to the server. The server uses this information to monitor the performance of the global model and to see if the global model can be improved for the next training round. Finally, after all ten fit training rounds, the server uses the updated model to test or evaluate the testing set that is present at the server's end. The overall accuracy, precision, recall, and F1-score after evaluating the test set is returned.

## 6 Experimentation

### 6.1 Partitioning of the dataset

The dataset is initially partitioned into 80% for training and 20% for testing. The testing part of the dataset is retained on the server end for evaluation on the test set, while the training portion is distributed to the clients. To determine the number of partitions per client on the training set, where the total number of partitions is 100 (representing the total number of clients), this procedure is followed: First, the total number of samples in the training dataset is obtained. Next, the number of samples per partition or client is calculated by dividing the total number of samples by the total number of partitions or clients. This results in each client having an equal share of the dataset. If there is a remainder after the division, it is distributed among the first set of clients. For example, if the remainder is 12, these 12 samples are distributed, with one additional sample for each of the first 12 clients. As a result, the dataset is partitioned equally among 100 clients, with the first 12 clients having one extra sample for training. Once partitions are assigned to each client, 90% of the partitioned data is utilised by the client for training, while the remaining 10% is reserved as a validation dataset. The validation dataset is used to evaluate the updated global model after every fit Adversarial Training round. The evaluation results are then sent to the server, which utilises these metrics to monitor the global model's performance and refine it for the next training round.

### 6.2 Model structure and information about the parameters exchanged between clients and server

The FLAT global model is a simple ANN in the PyTorch deep learning framework for FLAT methodology. Since the sample dataset pertains to a binary classification task with num-samples set to 2, the input layer corresponds to the shape of the x-train and consists of 115 neurons. The model acts as the configuration for the updated global model and subsequently deploys local models among clients. The parameter 'num\_classes' corresponds to 2, as it is a binary classification problem, for the classification of benign (0) and attack (1) points. Table 3 describes the global model and subsequently, the updated local model configuration used in FLAT.

**Table 3** Simple feed forward neural network

| <i>Layer type</i>        | <i>Neurons</i>  | <i>Activation</i> | <i>Dropout rate</i> |
|--------------------------|-----------------|-------------------|---------------------|
| Input                    | 115             | -                 | -                   |
| Fully connected (hidden) | 32              | ReLU              | -                   |
| Dropout                  | -               | -                 | 0.5                 |
| Fully connected (hidden) | 16              | ReLU              | -                   |
| Dropout                  | -               | -                 | 0.5                 |
| Fully connected (output) | Num_classes = 2 | Sigmoid           | -                   |

Note: The model parameter sent from the clients to the server, initially set to 0.01.

### 6.3 Strategy

The strategy used by the server in the FLAT setup is the FedAvg strategy. It is commonly implemented in most federated learning frameworks and involves aggregating local model updates from multiple clients to create a global model. The client computes the differences between its locally trained model and the global model sent to it by the server after local training (Korkmaz et al., 2022).

The update formula for a client  $i$  during training is given by:

$$\theta(t+1) = \theta(t) - \eta \nabla F_i(\theta(t)) \quad (1)$$

where

- $\theta_i$  is the model parameters at client  $i$  after  $t$  training rounds
- $\eta$  is the learning rate
- $F_i(\theta_i)$  is the gradient of the local loss function
- $F_i$  with respect to the model parameters.

These changes/updates are sent to the server, which aggregates them to create a new global model. A common aggregation method used in the FedAvg strategy is simple averaging of model parameters (Lai and Q. Yan, 2022). The code sample sets up the configuration for the FedAvg strategy, for FLAT simulation defining how many clients to involve per training/fit rounds (in our case it is set to 10) and how many clients to involve for evaluation (in our case it is 25). The fraction fit is initially set to 0.0 and direct specification is later done by min-fit-clients and min-evaluate-clients.

The FedAvg strategy aggregates the model parameters from different clients. The aggregated model  $\bar{\theta}(t+1)$  at the server after  $t$  training rounds is calculated as the update formula for a client  $i$  during training is given by:

$$\bar{\theta}(t+1) = \theta(t) - \eta \nabla F_i(\theta(t)) \quad (2)$$

where

- $\theta_i$  is the model parameters at client  $i$  after  $t$  training rounds
- $\eta$  is the learning rate
- $F_i(\theta_i)$  is the gradient of the local loss function
- $F_i$  with respect to the model parameters.

The optimiser used is the Adam optimiser with hyperparameters beta-1 and beta-2, which are exponential decay rates for the first and second-moment estimates. The loss function employed is the cross-entropy loss function.

The FedAvg strategy in federated learning allows clients to share model updates with the server instead of raw data. This approach helps preserve data privacy. It is important to recognise though, that even these model upgrades carry some danger of information leaking. These updates may be used by malicious actors to deduce private information about the local training set. Incorporating strategies like safe aggregation protocols and differential privacy can help reduce these risks and improve security in subsequent implementations. Differential privacy methods makes it more difficult to piece together

sensitive information from individual client data by adding noise to model updates during aggregation. It is also crucial to note that FGSM generates adversarial samples by adding noise to training data, while differential privacy adds noise to model updates. Secure aggregation protocols use cryptographic techniques during the communication of model updates between clients and the server, to guarantee the integrity and secrecy of the data.

#### 6.4 Adversarial training with FGSM

Adversarial samples are created with the FGSM (Chen and Zhang, 2020). Typically, a normal deep-learning model uses back-propagated gradients to modify the weights in an attempt to reduce the loss. The attack uses the same back-propagated gradients to modify the input data to maximise the loss. The ‘fgsm\_attack’ function is used to implement the procedure. By adding a tiny perturbation to the direction of the input data, which are the input train samples, and in this manner, the function creates adversarial samples. Given an input sample  $x$ , the adversarial example  $x_{adv}$  is computed as:

$$X_{adv} = x + \epsilon \text{ sign}(\nabla J(x, y)) \quad (3)$$

where

- $x$  is the original input sample
- $X_{adv}$  is the generated adversarial example
- $\epsilon$  is a small constant representing the perturbation magnitude
- $J(x, y)$  is the gradient of the loss function  $J$  with respect to the input  $x$  for a given true label  $y$
- $\text{sign}(\cdot)$  denotes the sign function.

The magnitude of perturbation introduced to the input train samples, defined as ‘epsilon’, is a parameter taken by the function and set to 0.01. The function computes the output of the model on the input data, computes the gradients of the loss concerning the input data, computes the loss between the output and the true label, and adds a small perturbation to the input data in the direction of the gradients’ sign. The function then returns the modified data as its output. The ‘fgsm\_attack’ function generates both clean and adversarial samples, which are used by the train function to train the model. The clean samples are the initial input train samples, while the adversarial samples are produced by using the ‘fgsm\_attack’ function on the input train samples. To increase the model’s resilience against adversarial network attacks, it is trained on both clean and adversarial samples (Chen and Zhang, 2020). Adversaries are typically not included in the testing phase as the goal is to make the model more resistant to perturbations and develop more robustness against adversarial attacks (Pontara da Costa, 2022). During the evaluation/testing phase, the FLAT model’s performance is assessed on clean, real-world data to ensure its generalisation ability.

#### 6.5 The server

The server is responsible for orchestrating the entire FLAT process. It initialises the global model, which serves as a starting point for the federated learning process

(Korkmaz et al., 2022). It also selects several clients for the training and the evaluation phase, the selection process by which it selects these clients can be random, through weighted sampling and other methods. The server sends the current model parameters and configuration details to the clients. These configuration details can include hyperparameters, and learning rates among others. The server periodically requests evaluation results from a subset of clients to assess the performance of the global model on local datasets. After this, it distributes the updated FLAT global model to the selected clients for the next training round. The server monitors the convergence of the global model over multiple rounds too. It also facilitates communication and synchronisation amongst the clients, resulting in a smooth federated learning process (Neto et al., 2022).

The flower server is not a physical machine. It is more of a programmatic entity implemented within the flower simulation script. When running the simulation locally on the machine, the server code executes within the same Python script as the client processes. It does not have a dedicated central processing unit (CPU) allocation like the clients but shares processing power with the client processes based on the overall workload at any given time. The operating system manages this dynamic sharing based on factors like process priority and resource demands. Ray-a distributed computing library efficiently schedules the server code and client training processes across available CPU cores on the machine.

## 6.6 *The clients*

The clients have a local copy of the dataset which is split into training and validation sets. The training set is used to train the global model sent by the server and the validation set is used by the server to monitor the performance of the global model after one training round. Client selection can be random or can be done using weighted sampling. In our case, client selection is random but every client is utilised and no clients are repeated. In one fit training round, ten clients are selected labelled from numerical values 1 to 10 and in the next training round, a different set of clients are sampled labelled from 1 to 10. It is important to note that while the labelling of clients may be the same, the same client is not repeated for the next training round. The clients are trained on clean and adversarial samples locally for 1 epoch. The client computes updates/differences between the global model and its locally trained model. These updates are sent to the server and are aggregated on the server end using the FedAvg strategy (Neto et al., 2022). For each FLAT round, a copy of the global updated model is sent to all selected clients (Sharma et al., 2023).

In federated learning, training the model with ten participants every round can be slow and require a lot of communication. To address this, client combinations can be used. Here, a random subset of clients (like a study group) is chosen to participate in each training round. This improves efficiency by reducing data transfer and training time. Additionally, these random selections expose the model to diverse data, potentially leading to a better performing model. However, there's a trade-off. Randomness can introduce some instability in training, and each round might have limited data access. To mitigate this, one can increase training rounds or use techniques that consider data quality from different clients. Overall, client combinations offer a valuable balance between efficiency and reliability for large-scale federated learning.

Virtual clients are used in the flower simulation to represent physical devices or independent procedures. These simulated clients operate independently, handling their

own local model copy, training data partition, and communication with the main flower server. This allows for parallel local training on multiple clients simultaneously. Flower leverages Ray—a distributed computing library for task scheduling across available CPU cores on your machine, ensuring efficient resource utilisation. These virtual clients offer a safe environment for testing federated learning algorithms without requiring physical hardware, all while simulating real-world client behaviour when interacting with the central server. CPU allocation for Flower clients and the server is handled by the operating system during local execution. It is crucial to differentiate between scheduling and allocation. Ray focuses on scheduling tasks across available cores, but does not directly control how much CPU each process gets.

By retaining private information on client devices, FLAT re-imagines network intrusion detection. In order to extract features for attacks such as ARP-MitM, clients examine their local traffic. Direct sharing of these features is never done. Rather, local models are trained by clients, who then transmit anonymous updates to a central server. The server trains a strong model by aggregating information from all around the network, utilising this collective wisdom. This enables the model to identify more general attack patterns, such as anomalous surges in ARP queries or suspicious communication, even in cases when a particular attack was not noted on a single device. Additionally, FLAT is strengthened by the scattered nature of the data. Because every client sees a different portion of the network, the model is exposed to a greater variety of attack methods. Further improving detection can be achieved through adversarial training on the server with synthetic data that mimics ARP-MitM attacks. Combining these strategies, FLAT provides a strong defence system that protects privacy by using the network's collective knowledge to recognise and neutralise even the most complex threats.

When it comes to active wiretapping attack detection, FLAT leverages client-side data and collaboration to enable active wiretapping detection while maintaining privacy. In order to extract information such as packet size, origin, destination, and protocol, clients analyse the traffic on their local network. Eavesdropping can be detected by anomalies such as sudden increases in traffic volume, strange communication patterns, or large amounts of data transfer protocols coming from unidentified sources. The decentralised strategy is essential. An expanded and varied understanding of network behaviour is acquired by the server's global model through training on anonymised model updates from every client. This is important because different network segments may have distinct attack signatures, and attackers frequently use a variety of approaches. While one client may only be observing one particular strategy, aggregating updates from multiple clients exposes the model to a greater variety of possible assault patterns. As a result, even if these patterns have never been explicitly observed on a single client, it can now learn and recognise them. In addition, merging updates facilitates the model's acquisition of more broadly applicable characteristics suggestive of active wiretapping in various network scenarios. Adversarial training using noisy data to simulate attempts at active wiretapping have the potential to improve the server model's and the clients' models' capacity to identify and fend off real-world threats.

Figure 4 represents the logs after one fit adversarial training/FLAT round. It is observed that the training round number is 10, and ten clients are selected. The clients train on clean and adversarial samples and each client's adversarial training loss is computed. From the logs, it is observed that the adversarial training round has been executed successfully, with 0 failures. After the server selects 25 random clients for

evaluation of the global updated model on their validation sets, a good accuracy of about 92%, a good precision of about 91.8%, a good recall, and an F1-score of 91.6% and 91.7% respectively are found.

**Figure 4** Logs obtained after one adversarial training round (see online version for colours)

```
[2023-11-28 18:41:29,943][flwr][DEBUG] - fit_round 10: strategy sampled 10 clients (out of 100)
(DefaultActor pid=3870) Epoch 1, Loss: 0.8504871654008286
(DefaultActor pid=3870) Finished Training with Adversarial Examples
(DefaultActor pid=3870) Epoch 1, Loss: 0.8342465761627696
(DefaultActor pid=3870) Finished Training with Adversarial Examples
(DefaultActor pid=3870) Epoch 1, Loss: 0.8259615454731918
(DefaultActor pid=3870) Finished Training with Adversarial Examples
(DefaultActor pid=3870) Epoch 1, Loss: 0.8267539101931578
(DefaultActor pid=3870) Finished Training with Adversarial Examples
(DefaultActor pid=3870) Epoch 1, Loss: 0.8362437898330308
(DefaultActor pid=3870) Finished Training with Adversarial Examples
(DefaultActor pid=3870) Epoch 1, Loss: 0.8360681703640458
(DefaultActor pid=3870) Finished Training with Adversarial Examples
(DefaultActor pid=3870) Epoch 1, Loss: 0.8292020800504875
(DefaultActor pid=3870) Finished Training with Adversarial Examples
(DefaultActor pid=3870) Epoch 1, Loss: 0.8423147401101389
(DefaultActor pid=3870) Finished Training with Adversarial Examples
(DefaultActor pid=3870) Epoch 1, Loss: 0.8252309643376429
(DefaultActor pid=3870) Finished Training with Adversarial Examples
(DefaultActor pid=3870) Epoch 1, Loss: 0.8227335229152586
(DefaultActor pid=3870) Finished Training with Adversarial Examples
DEBUG flwr 2023-11-28 18:42:51,640 | server.py:236 | fit_round 10 received 10 results and 0 failures
[2023-11-28 18:42:51,640][flwr][DEBUG] - fit_round 10 received 10 results and 0 failures
INFO flwr 2023-11-28 18:45:18,449 | server.py:125 | fit progress: (10, 0.38687918078695065, {'accuracy': 0.924692624996506, 'precision': 0.918602922521
[2023-11-28 18:45:18,449][flwr][INFO] - fit progress: (10, 0.38687918078695065, {'accuracy': 0.924692624996506, 'precision': 0.9186029225293608, 'reca
```

## 6.7 Technologies and frameworks used

- **Flower:** An open source framework for building federated learning systems. It provides a set of tools for the development, deployment, and experimentation of federated learning applications. Developers can use their preferred deep learning libraries such as PyTorch/Tensorflow. Flower has specific support for PyTorch so that is implemented in the setup (Rahman et al., 2020).
- **Libraries:** Pytorch – a popular deep learning library, Hydra (1.3.2) – a configuration management library, that maintains consistency and reliability while handling large-scale systems. Ray (2.6.0) is a general-purpose distributed computing library. Seaborn – a data visualisation library, Matplotlib – a data visualisation library, Scikit-Learn machine learning library (Rahman et al., 2020).

Flower focuses on federated learning logic, while Ray handles communication infrastructure, distribution of tasks, and resource management (Rahman et al., 2020).

Ray performs the task of allocating work among the CPU cores on your computer that are available when running a flower simulation. These responsibilities involve running the server code in the flower script as well as the client training procedures. Ray guarantees that your machine's resources are used to their fullest potential by scheduling these jobs efficiently. Ray also makes parallelism and concurrency easier. This makes it possible for client training procedures to run in parallel, which is necessary for effective federated learning simulations. Ray makes training faster by efficiently allocating these tasks among cores. Ray can be setup to be aware of the total number of CPU cores on your computer, even though it does not directly control virtual instances or CPU allocation. Ray can avoid overtaxing the system and make well-informed scheduling decisions thanks to this awareness (Rahman et al., 2020).

## 6.8 The need for FLAT in network attack detection

The need for FLAT methodology arises in IoT scenarios, where centralised training may pose several challenges, with a few as follows:

- 1 Challenges in maintaining data privacy: IoT devices often collect sensitive information, and centralised training for attack detection involves pooling all data in one location, raising privacy concerns. In federated learning, model training is done on the client devices themselves. Moreover, the clients when communicating with the server send model updated parameters and not the actual raw data itself. This way user data is not compromised. Therefore, it avoids the need to share raw data centrally (Neto et al., 2022).
- 2 Limited network bandwidth: IoT devices may operate in environments where there is unreliable network connectivity. Therefore, uploading large datasets to a central server can be considered impractical. FLAT allows models to be trained locally, reducing the need for transmitting large datasets. Also, in normal machine learning scenarios, if the central server fails, then the entire collected IoT data is lost. Here, it is distributed locally among clients so it can be recovered (Jiyal et al., 2022; Muhammad et al., 2023).
- 3 IoT applications involve real-time or low latency decision making: Sending the entire dataset to a central server and then waiting for updates as it finishes training can result in huge unacceptable delays. FL facilitates on-device training, allowing models to adapt quickly to changing conditions (Korkmaz et al., 2022; Wang, 2020).
- 4 Interoperability across device types: Heterogeneous IoT systems require diverse device types and architectures. A centralised model can struggle to accommodate devices with various characteristics, therefore Federated Learning allows models to be trained on specific device types optimising for local conditions, furthermore, when it is optimised with adversarial training it can be trained to dynamically adapt to varying attacks (Pan et al., 2022; Lai and Yan, 2022).
- 5 In network security, adversarial attacks are a common concern (Song et al., 2018). In an attempt to trick the network intrusion detection system, attackers can try to alter the data. FLAT uses adversarially constructed instances to train models. This strengthens the model's defences against possible intrusions. Training on adversarial examples can lead to improved generalisation performance. The FLAT model learns to capture more robust and meaningful features, contributing to better performance on unseen data (Song et al., 2018; da Silva et al., 2022).
- 6 Because deep learning models are complicated and nonlinear, adversarial assaults can target them, especially neural networks. An effective way to reduce vulnerabilities and increase the model's defence against these kinds of attacks is through adversarial training.
- 7 Frequently, adversarial samples designed to trick one model extend to other models that have been trained on comparable tasks. The goal of adversarial training is to lessen the transferability of adversarial examples, which increases the difficulty of an attack's success across several models (Gungor et al., 2022).

## 7 Criteria for model selection

The paper looks into several classification models for network attacks. The two primary types of these models are machine learning and deep learning. The structure and operation of the brain serve as an inspiration for deep learning models such as CNNs and ANNs. Although they can be quite effective at analysing network data, their decision-making process can be convoluted and challenging to comprehend. A different strategy is provided by machine learning models including LR, random forest, XGBoost, and LGBM. They are ideally suited for contexts with limited resources because they are usually simpler to understand and use less processing power. Additionally, the paper also proposes FLAT, a novel strategy that combines the privacy advantages of FLAT, a method that exposes the model to intentionally created noisy data to enhance its detection capabilities of actual attacks.

The models are compared based on six key factors:

- 1 Model complexity: A complex model has many intricate parts, while a simple model only has a few. While complex models have the potential to be more powerful, they are also harder to implement and understand.
- 2 Model interpretability: Interpretable models allow you to see how they arrive at their conclusions, just the way transparent windows do. Less interpretable models are similar to ‘black boxes’ in that it is unclear why they do what they do.
- 3 Performance and scalability: Performance refers to the accuracy of the model in classifying attacks. Scalability refers to the model’s ability to process large workload without slowing down.
- 4 Computational and resource efficiency: Resource efficient models use less processing power, making them ideal for devices with limited resources.
- 5 Model robustness and adversarial defence: Consider the attempt to trick the model by using fake or noisy data. Robust models can resist adversarial attacks and maintain their accuracy.
- 6 Evaluation metrics above 85%: This simply means the model needs to perform very well on test datasets, achieving an accuracy of at least 85%. This ensures the model can reliably identify network attacks.

**Table 4** Criteria for model selection

| <i>Sl. no.</i> | <i>Criteria for model selection</i>              |
|----------------|--------------------------------------------------|
| 1              | Model complexity                                 |
| 2              | Model interpretability                           |
| 3              | Performance and scalability                      |
| 4              | Computational and resource efficiency            |
| 5              | Model robustness and adversarial defence         |
| 6              | Evaluation metrics above 85% (for both datasets) |

Table 4 provides a brief overview of the criteria used for model selection, while Table 5 elaborates on the comparison between deep learning and machine learning models based

on these criteria. A checkmark (✓) means that the model performs well on that criterion, while a cross (✗) means that the model has a weakness on that criterion.

**Table 5** Comparison of models based on criteria

| Models                          | Criteria |   |   |   |   |   |
|---------------------------------|----------|---|---|---|---|---|
|                                 | 1        | 2 | 3 | 4 | 5 | 6 |
| CNN                             | ✓        | ✗ | ✓ | ✗ | ✓ | ✓ |
| ANN                             | ✓        | ✓ | ✓ | ✓ | ✓ | ✓ |
| Random forest                   | ✗        |   | ✓ |   | ✓ | ✓ |
| XGBoost classifier              | ✗        | ✓ | ✓ | ✓ | ✓ | ✓ |
| Light gradient boosting machine | ✗        | ✓ | ✓ | ✓ | ✓ | ✓ |
| Logistic regression             | ✓        | ✓ | ✓ | ✓ | ✓ | ✓ |
| FLAT                            |          | ✗ |   | ✓ |   |   |

When it comes to network attack classification, different levels of complexity can be found in random forest, XGBoost, LGBM, and LR. Random forest and XGBoost are decision tree ensembles that strike a compromise between interpretability and performance, whereas LR is a straightforward linear model. Even if they are strong, CNN and ANN increase computational demands and complexity, particularly in deep systems, compromising interpretability in exchange for increased efficiency. The comparison table that is presented in Table 5 demonstrates that FLAT is a better model in several categories. FLAT stands apart for having a simpler model, which makes it easier to use and comprehend overall. One advantage of understanding model decisions is its greater interpretability when compared to several other models, including CNN, ANN, and LR. Performance and scalability are two key areas where FLAT shines, as they are essential for handling the large-scale distributed scenarios that come with federated learning. Moreover, it performs computationally and resource efficiently compared to models such as CNN and ANN, which makes it especially useful in contexts with limited resources. The model's ability to effectively apply adversarial training reinforces its resilience to adversarial threats. Achieving evaluation metrics above 85% for both datasets underscores FLAT's overall excellence in terms of accuracy, precision, recall, and F1-score. In summary, FLAT emerges as a compelling choice, showcasing a balanced combination of simplicity, interpretability, performance, efficiency, robustness, and decent evaluation metrics tailored to cater to various application needs.

## 8 Results and discussion

Results of the datasets obtained, along with appropriate visualisations and analysis of the data, pre- and post-FLAT integration in a centralised and decentralised setup.

### 8.1 Kitsune network attacks – man-in-the-middle attack dataset

The Kitsune network attacks dataset was chosen because of its broad applicability for detecting general active wiretapping incidents. Unlike datasets that focus on a specific type of attack, Kitsune's ability to classify packets as benign or malicious without

narrowing down to specific techniques made it suitable for the study's focus on general anomaly detection in network traffic.

### *8.1.1 Centralised pre-FLAT approach for man-in-the-middle attack dataset*

This section describes the results obtained before FLAT integration (that is, the dataset is subjected to supervised machine learning algorithms – XGB, random forest, LGBM, LR, and deep learning models: ANN and CNN. In a centralised training setup, the training of the machine learning model takes place on a central server or a centralised data processing unit. All training data is collected and stored in one location. The model is trained on the aggregated data, and the updates are sent back to the central server. Because all data is sent to and processed on a central server, privacy concerns may arise, posing potential risks, particularly when dealing with sensitive information (You et al., 2022).

### *8.1.2 Performance metrics of centralised training (pre-FLAT) – machine learning and deep learning models for man-in-the-middle attack dataset*

#### *8.1.2.1 Deep learning models*

The structure of the CNN model and the ANN model with Adam optimiser, binary cross-entropy loss function, and a callback with early stopping with patience of three epochs when validation loss exceeds train loss in a centralised pre-FLAT setup, is described in Table 6. The CNN model is trained on ten epochs with early stopping and the ANN model is trained for 20 epochs with early stopping, with a batch size of 20 on the provided training data. Finally, the performance is evaluated on the test data, and the test accuracy is printed as a percentage. The train test split is 80–20 with a random state of 42. The original epochs both models are trained for is 100, but it showed early stopping for 10 and 20 epochs respectively.

**Table 6** Configuration of deep learning models (CNN and ANN) in a centralised pre-FLAT setup

| <i>Model</i> | <i>Layer type</i>  | <i>Filters</i> | <i>Neurons</i> | <i>Kernel size</i> | <i>Activation</i> | <i>Regularisation</i> |
|--------------|--------------------|----------------|----------------|--------------------|-------------------|-----------------------|
| CNN          | 1D convolutional   | 32             | -              | 3                  | ReLU              | -                     |
|              | Max pooling        | -              | -              | 2                  | -                 | -                     |
|              | 1D convolutional   | 64             | -              | 3                  | ReLU              | -                     |
|              | Max pooling        | -              | -              | 2                  | -                 | -                     |
|              | Global max pooling | -              | -              | -                  | -                 | -                     |
|              | Fully connected    | 128            | -              | -                  | ReLU              | L2: 0.001             |
|              | Dropout            | -              | -              | -                  | -                 | -                     |
| ANN          | Dense              | -              | 128            | -                  | ReLU              | L2: 0.001             |
|              | Dropout            | -              | -              | -                  | -                 | -                     |
|              | Dense              | -              | 64             | -                  | ReLU              | L2: 0.001             |
|              | Dropout            | -              | -              | -                  | -                 | -                     |
|              | Dense (output)     | -              | 1              | -                  | Sigmoid           | L2: 0.001             |

Based on the provided metrics in Table 7 in a centralised pre-FLAT setup, CNN outperforms ANN in terms of accuracy, precision, recall, and F1-score. The CNN achieved a higher overall accuracy compared to the ANN, indicating that it made fewer mistakes in classifying both benign and attack points. The CNN's precision (95.53) is higher than that of the ANN (93.16), suggesting that the CNN has a better ability to correctly identify attack points without misclassifying benign points. The CNN's recall (99.93) is significantly higher than the ANN's recall (97.71), indicating that the CNN has a better ability to capture all actual attack points. The CNN's F1-score (97.68) is higher than the ANN's F1-score (95.39), indicating that the CNN has a better balance of precision and recall.

**Table 7** Performance metrics comparison between ANN and CNN (man-in-the-middle attack dataset) in a centralised pre-FLAT setup

| <i>Metric</i> | <i>CNN model</i> | <i>ANN model</i> |
|---------------|------------------|------------------|
| Accuracy      | 97.5%            | 95.12%           |
| Precision     | 95.5%            | 93.16%           |
| Recall        | 99.93%           | 97.71%           |
| F1-score      | 97.68%           | 95.39%           |

#### 8.1.2.2 Normal machine learning models

The machine learning models used in the centralised pre-FLAT analysis include LR, random forest, XGB, and LGBM. The dataset was split into training and testing sets for all models using an 80–20 split with a random state of 42.

#### 8.1.2.3 Comparison of all models in a centralised pre-FLAT setup

By doing a comparison of the models in a centralised pre-FLAT setup based on their performance metrics as shown in Figure 5 for man-in-the-middle attack dataset, the CNN model achieves the highest accuracy, precision, recall, and F1-score among all models, making it the best performer. Random Forest comes second, followed by the ANN.

**Table 8** Performance metrics for machine learning models in a centralised pre-FLAT setup (man-in-the middle attack dataset)

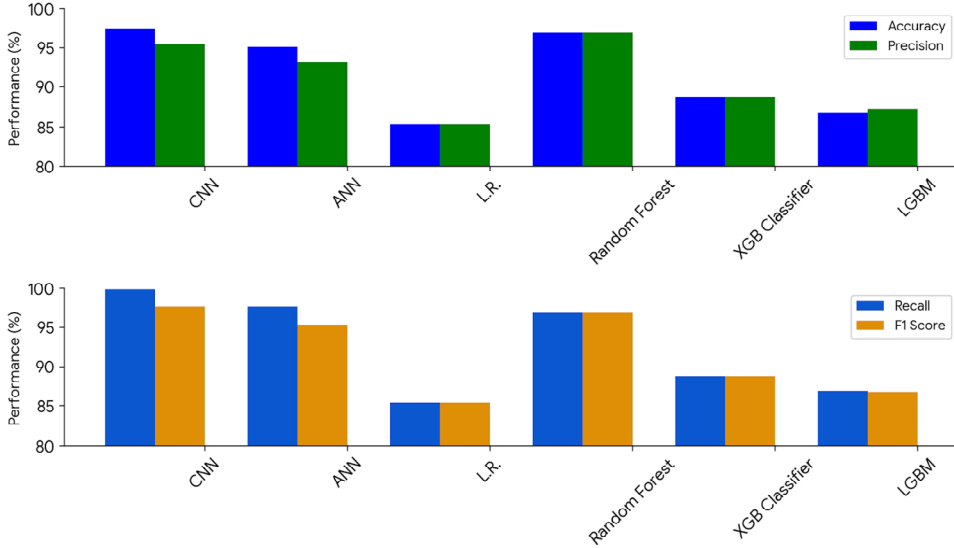
| <i>Model</i>                    | <i>Accuracy</i> | <i>Precision</i> | <i>Recall</i> | <i>F1-score</i> |
|---------------------------------|-----------------|------------------|---------------|-----------------|
| Logistic regression             | 85.39%          | 85.38%           | 85.39%        | 85.38%          |
| Random forest (ten estimators)  | 96.91%          | 96.93%           | 96.91%        | 96.91%          |
| XGB classifier (ten estimators) | 88.82%          | 88.82%           | 88.82%        | 88.81%          |
| LGBM (ten estimators)           | 86.88%          | 87.32%           | 86.88%        | 86.76%          |

Based on the performance metrics in Table 8 in a centralised pre-FLAT setup, the random forest model with ten decision trees appears to have the best overall performance for classifying attack and benign points. It has a high level of accuracy, precision, recall, and F1-score. Comparison of all models in a centralised pre-FLAT setup:

By doing a comparison of the models in a centralised pre-FLAT setup based on their performance metrics as shown in Figure 5 for the man-in-the-middle attack dataset (where LR expands to logistic regression), the CNN model achieves the highest accuracy,

precision, recall, and F1-score among all models, making it the best performer. Random forest comes second, followed by the ANN.

**Figure 5** Visualisations of performance metrics of six models for MiTM attack dataset (see online version for colours)



### 8.1.3 Device performance during centralised training (pre-FLAT) for man-in-the-middle attack dataset

In a centralised pre-FLAT setup, transmitting data to the central server introduces latency, especially if the data size is large. The time it takes to transfer data can become a bottleneck, especially in scenarios with slow or unreliable network connections. The model training is done by the centralised server, and the time it takes is determined by the complexity of the model, the size of the dataset, and the computing resources available. Training large models on extensive datasets can be time-consuming. The central server bears the brunt of computation, potentially leading to high resource utilisation. This can affect the server's responsiveness and other tasks it may be handling. Furthermore, centralised training involves centralising sensitive data, raising privacy concerns. Transmitting and storing such data centrally requires careful handling to ensure security.

Decentralised/federated learning with adversarial training (post-FLAT) for man-in-the-middle attack dataset this section details the setup for man-in-the-middle attack classification using FLAT setup. The setup for FLAT simulation using the flower framework is explained in the methodology section of the paper. In a FLAT setup, the model is trained across multiple decentralised devices or clients without exchanging raw data while the data stays on the server, and only model updates are shared (Korkmaz et al., 2022). The central server sends the current model to clients, where it is trained on local data. The updates from the clients are then sent back to the central server and aggregated to update the global model. Federated learning addresses privacy concerns by keeping data localised and only sharing model updates (Sharma et al., 2023). Adversarial training is used to improve the robustness of the model's performance by inducing

adversarial samples. The FLAT methodology makes it suitable for scenarios where data privacy and the diversity of the model in attack classification is a significant concern.

Based on the evaluation and performance metrics for over 10 FLAT rounds from the analysis of Figures 6, 7, and 8, Tables 9, 10, 11, and 12 are formulated.

From Figure 6, it is observed that in a post-FLAT setup over ten rounds for man-the-in-the-middle-attack dataset, the losses seem to be relatively consistent across the rounds, with values ranging from approximately 0.381 to 0.395. Consistent losses can suggest stable training dynamics. The absolute magnitude of the losses is relatively low, indicating that the model can fit the training data well. If the losses were to fluctuate significantly between rounds, it might indicate instability in the training process. However, the provided losses do not show drastic changes. From Figure 7, the accuracy remains consistently high across all rounds, ranging from approximately 91.35% to 92.79%. From Figure 8, precision values vary across rounds, with the highest precision observed in round 8 (94.79%). Recall values also exhibit variability across rounds, ranging from approximately 87.66% to 94.91%. F1-scores, which balance precision and recall, consistently maintain a high level, ranging from approximately 90.48% to 92.23%. The FLAT model consistently performs well in terms of accuracy, precision, recall, and F1-score throughout the ten rounds.

From Table 9 in a post-FLAT setup over ten rounds for the MiTM attack dataset, the FLAT model achieves a high overall performance level, with an average accuracy, precision, recall, and F1-score consistently above 90%. This indicates the effectiveness of the federated learning approach with adversarial training. The rounds with the highest metrics are not consistent across different evaluation criteria (accuracy, precision, recall, and F1-score). This suggests that the FLAT model's performance peaks in different aspects at different times. While there is variability in the best and worst-performing rounds, the consistently high average metrics suggest the robustness and general effectiveness of the FLAT approach. The variability in performance across rounds emphasises the dynamic nature of federated learning. The model exhibits adaptability, optimising its behaviour based on the characteristics of the data or Adversarial Training challenges present in each round. FLAT provides competitive and robust performance across multiple evaluation metrics, indicating its effectiveness in enhancing model generalisation and resistance to adversarial attacks.

From Table 10 in a post-FLAT setup over ten rounds for the MiTM attack dataset, round 7 achieved the highest accuracy, indicating that the FLAT model made the fewest overall mistakes in its predictions during this round. The FLAT model's robustness and generalisation capabilities might have been enhanced in round 7, leading to accurate predictions on both training and unseen data. Round 8 recorded the highest precision, suggesting that the FLAT model exhibited a low rate of false positives during this round. In round 8, the FLAT model was particularly effective in correctly identifying positive instances, minimising the chances of incorrectly labelling negatives as positives. Round 5 achieved the highest recall, indicating that the FLAT model effectively captured a significant portion of positive instances among the total actual positives. The FLAT model's heightened sensitivity in round 5 is crucial for scenarios where identifying all positive instances is a priority, even at the cost of some false positives. Round 1 attained the highest F1-score, reflecting a harmonised balance between precision and recall. The FLAT model in round 1 effectively managed the trade-off between minimising false positives and false negatives, showcasing a well optimised performance.

From Table 12 in a post-FLAT setup over 10 rounds for the MiTM attack dataset, it is observed that round 3 recorded the lowest accuracy, indicating that the FLAT model made more mistakes in its predictions compared to other rounds. Round 5 had the lowest precision, round 8 recorded the lowest recall, and round 3 had the lowest F1-score, reflecting challenges in balancing precision and recall. One can further conduct a thorough analysis of the data and training dynamics during round 3. Identify potential adversarial challenges or data characteristics that led to lower accuracy. One can also investigate strategies to mitigate false positives in round 5, such as adjusting the decision threshold or incorporating additional features. In summary, one should consider incorporating insights from both high and low-performing rounds to refine the FLAT model for improved overall performance.

**Figure 6** Average losses through ten FLAT rounds for MiTM dataset (see online version for colours)

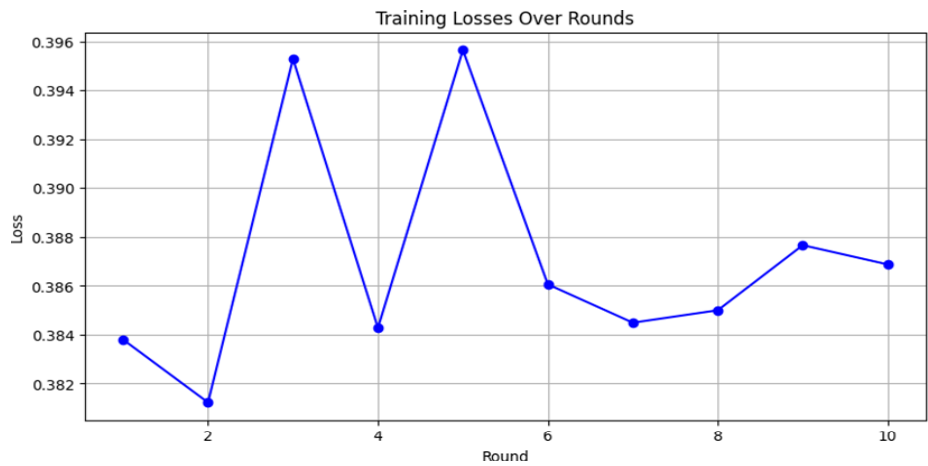


**Figure 7** Accuracy over ten FLAT rounds for MiTM dataset (see online version for colours)

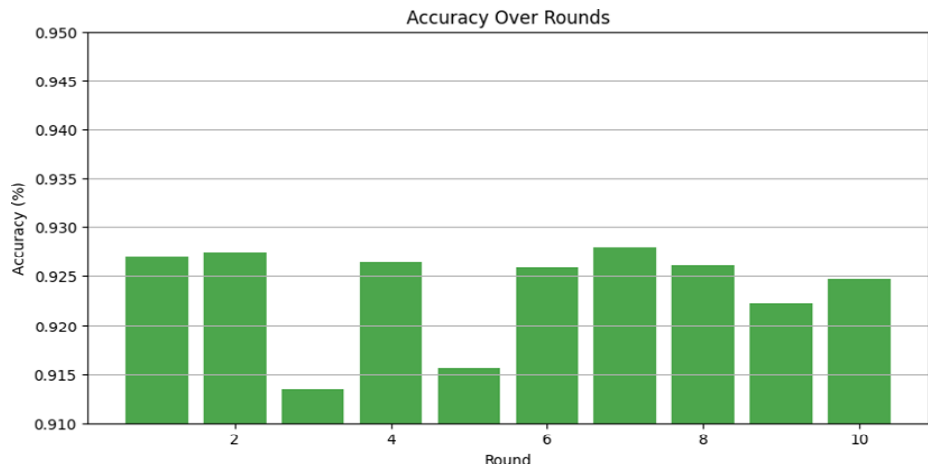
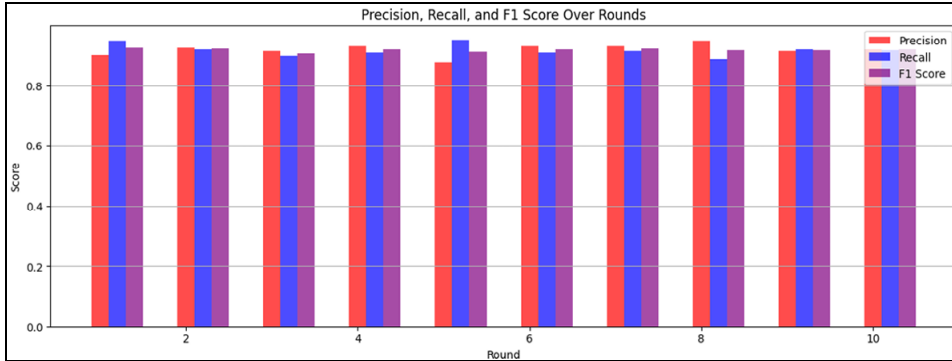


**Figure 8** Precision, recall, F1-score over ten FLAT rounds for MiTM dataset (see online version for colours)**Table 9** Average performance metrics for FLAT training (MiTM attack dataset)

| <i>Metric</i> | <i>Average value</i> |
|---------------|----------------------|
| Accuracy      | 92.37%               |
| Precision     | 91.77%               |
| Recall        | 91.61%               |
| F1-score      | 91.66%               |

**Table 10** Rounds with the highest metrics (MiTM attack dataset)

| <i>Metric</i> | <i>Highest value</i> | <i>Round</i> |
|---------------|----------------------|--------------|
| Accuracy      | 92.79%               | 7            |
| Precision     | 94.79%               | 8            |
| Recall        | 94.91%               | 5            |
| F1-score      | 92.23%               | 1            |

**Table 11** Rounds with the second-highest metrics (MiTM attack dataset)

| <i>Metric</i> | <i>Second-highest value</i> | <i>Round</i> |
|---------------|-----------------------------|--------------|
| Accuracy      | 92.74%                      | 2            |
| Precision     | 92.90%                      | 4            |
| Recall        | 94.65%                      | 1            |
| F1-score      | 92.05%                      | 7            |

**Table 12** Rounds with the lowest metrics (MiTM attack dataset)

| <i>Metric</i> | <i>Lowest value</i> | <i>Round</i> |
|---------------|---------------------|--------------|
| Accuracy      | 91.35%              | 3            |
| Precision     | 87.66%              | 5            |
| Recall        | 88.71%              | 8            |
| F1-score      | 90.48%              | 3            |

8.1.4 Individual client performance from federated learning with adversarial training (post-FLAT) setup for man-in-the-middle attack dataset

Every client in the present flower simulation setup has the same training environment and starting point. They load pre-trained weights and start with the same model definition. This guarantees that the initial model state is the same for every client. Moreover, each follows the same training program. The server provides training configurations to the ‘fit’ function; however, these configurations are applied uniformly, i.e., the learning rate, optimiser parameters, and number of local training epochs are the same for all clients. Lastly, clients would receive statistically equivalent subsets of the total data when data partitioning was done using an independent and identically distributed (IID) sampling strategy. As a result of the existing setup, all clients begin and train in the same way.

**Figure 9** Individual client losses for every FLAT round for MiTM dataset (see online version for colours)

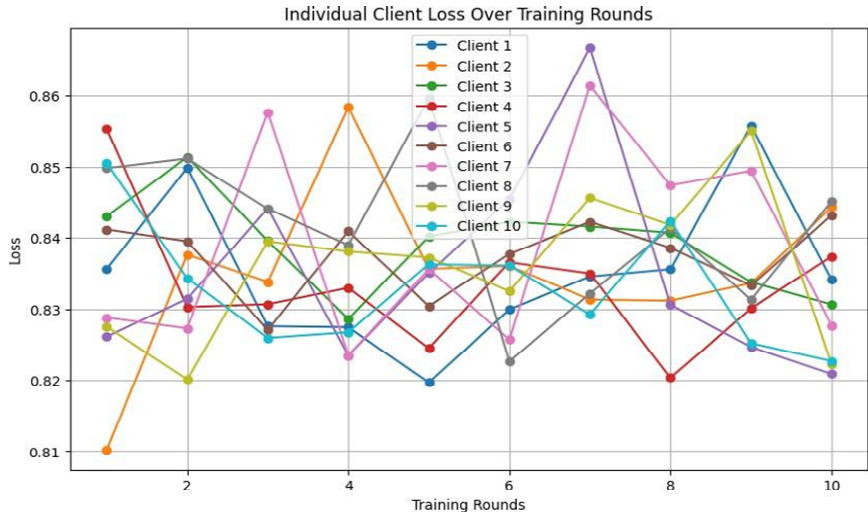


**Table 13** Average losses for every chosen client (MiTM attack dataset)

| Clients   | Average losses for every $i^{\text{th}}$ client |
|-----------|-------------------------------------------------|
| Client 1  | 0.837                                           |
| Client 2  | 0.837                                           |
| Client 3  | 0.837                                           |
| Client 4  | 0.834                                           |
| Client 5  | 0.835                                           |
| Client 6  | 0.834                                           |
| Client 7  | 0.842                                           |
| Client 8  | 0.837                                           |
| Client 9  | 0.837                                           |
| Client 10 | 0.833                                           |

From Table 13 in a post-FLAT setup over ten rounds for the MiTM attack dataset, the average loss values across the ten randomly chosen (not repeated) clients for each FLAT training round are consistent. Shared model parameters, training configurations, and IID data create a scenario where the clients have similar average losses across rounds, ranging from 0.833 to 0.842. The clients begin with the same model parameters and undergo identical training configurations, as a result of which their performance converges to similar loss values. The losses are similar and do not show significant fluctuations, indicating stability in the training process. The fact that the losses are consistent across different clients suggests that the FLAT model has converged to a similar state across various subsets of clients. This is a positive outcome, as federated learning aims to leverage the collective knowledge from diverse datasets. The consistent average losses imply that the FLAT model maintains its robustness across different training rounds. The adversarial training effectively improves the model's ability to handle perturbations, and this improvement is sustained over multiple rounds. Randomly selecting ten clients for each round contributes to exploring the heterogeneity of data present in the entire client pool. This is beneficial for ensuring that the FLAT model generalises well to various data distributions.

**Table 14** Round-wise best-performing clients (MiTM attack dataset)

| <i>Round</i> | <i>Best-performing client</i> |
|--------------|-------------------------------|
| 1            | Client 5                      |
| 2            | Client 1                      |
| 3            | Client 4                      |
| 4            | Client 8                      |
| 5            | Client 10                     |
| 6            | Client 3                      |
| 7            | Client 4                      |
| 8            | Client 6                      |
| 9            | Client 2                      |
| 10           | Client 10                     |

**Table 15** Round-wise worst-performing clients (MiTM attack dataset)

| <i>Round</i> | <i>Worst-performing client</i> |
|--------------|--------------------------------|
| 1            | Client 9                       |
| 2            | Client 4                       |
| 3            | Client 2                       |
| 4            | Client 1                       |
| 5            | Client 7                       |
| 6            | Client 10                      |
| 7            | Client 7                       |
| 8            | Client 5                       |
| 9            | Client 9                       |
| 10           | Client 1                       |

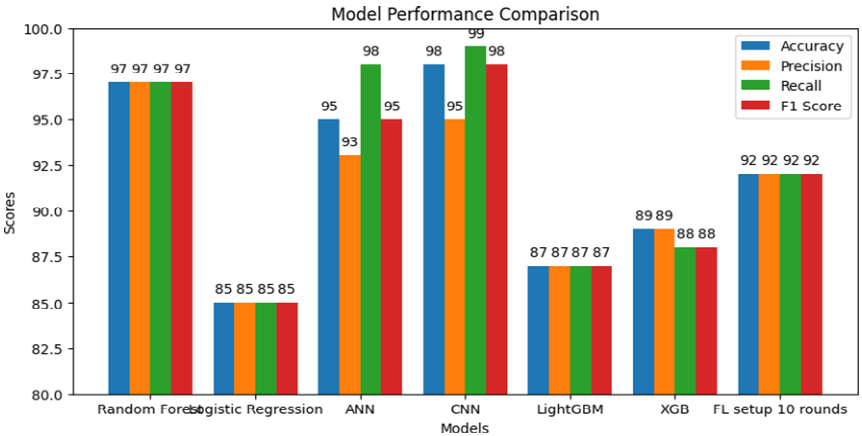
From Table 14 in a post-FLAT setup over ten rounds for the MiTM attack dataset, there is variability in the best-performing clients across different rounds, suggesting that the contribution of clients to FLAT model improvement varies throughout the training process. Different clients excel in different rounds, indicating the diversity in the quality of contributions from clients. This is beneficial for obtaining a comprehensive improvement in the FLAT model. The clients identified as best performers are adapting well to adversarial training strategies.

Similar to the best-performing clients, from the observations in Table 15, there is variability in the worst-performing clients across rounds. This variability indicates that clients who may not contribute effectively in one round might provide valuable contributions in subsequent rounds. The participation and impact of clients can change over rounds. A client who performs poorly in one round might contribute significantly to another, emphasising the dynamic nature of FLAT. The dynamic nature of client performance suggests that the federated adversarial training process is effectively leveraging the diversity of the client pool. The identified patterns can guide further investigations into why certain clients consistently perform well or poorly. The information about best and worst-performing clients provides valuable insights into the learning dynamics of the FLAT model, allowing for adjustments or targeted interventions if needed. It is important to note that while the best-performing clients contribute positively, the worst performing clients may still play a crucial role in the overall improvement of the global model as FLAT relies on collective knowledge.

### 8.1.5 Comparison of pre-FLAT and post-FLAT attack classification methodologies for man-in-the-middle attack dataset

From Figure 10, in a centralised pre-FLAT approach, it is inferred that The CNN outperforms other models in all metrics, indicating its effectiveness in capturing complex patterns. Random forest performs consistently well across metrics, making it the second-best performer.

**Figure 10** Comparison of pre-FLAT and post-FLAT (ten rounds) classification methodologies for MiTM attack dataset (see online version for colours)



The ANN shows competitive performance, particularly in recall. The worst performance is exhibited by the LR model. The decentralised post-FLAT approach performs well, demonstrating competitive metrics compared to standalone models. It offers a good balance between accuracy, precision, recall, and F1-score. In a decentralised setup, the FLAT methodology offers competitive performance in comparison to centralised traditional attack detection mechanisms. In summary, FLAT leverages edge devices for model training, without heavily relying on cloud resources and also accommodates scenarios where the dataset is distributed across devices or locations, and also provides robustness whilst classifying attacks.

## 8.2 Kitsune network attacks – active wiretap dataset

Centralised pre-FLAT approach for active wiretap dataset. The same machine learning and deep learning models in a centralised setup were used in the previous dataset.

### 8.2.1 Performance metrics of centralised training (pre-FLAT) – machine learning and deep learning models for active wiretap dataset

#### 8.2.1.1 Deep learning models

In a pre-FLAT setup, the ANN and CNN models have the same architecture as described in the MiTM attack dataset. The CNN and ANN models' performance metrics with an 80–20 train test split and random state 42 with early stopping on 20 epochs with patience of three epochs are described in Table 16.

Based on the evaluation metrics in Table 16, it is clear that CNN outperforms ANN whilst classifying attack and benign label points in the active wiretap network attack dataset in all performance.

#### 8.2.1.2 Machine learning models: the machine learning models used are XGB classifier, random forest, LGBM, and LR in a centralised pre-FLAT setup

Based on the metrics in Table 17 among supervised machine learning models in a centralised pre-FLAT setup, random forest, LGBM, and XGB classifiers show competitive and almost the same performance varying across different metrics. LR shows the least favourable performance.

**Table 16** Performance metrics for CNN and ANN models (active wiretap dataset) in a centralised (pre-FLAT) setup

| <i>Model</i> | <i>Performance metric</i> | <i>Value (%)</i> |
|--------------|---------------------------|------------------|
| CNN          | Accuracy                  | 97.52            |
|              | Precision                 | 97.37            |
|              | Recall                    | 97.24            |
|              | F1-score                  | 97.25            |
| ANN          | Accuracy                  | 95               |
|              | Precision                 | 96               |
|              | Recall                    | 95               |
|              | F1-score                  | 95               |

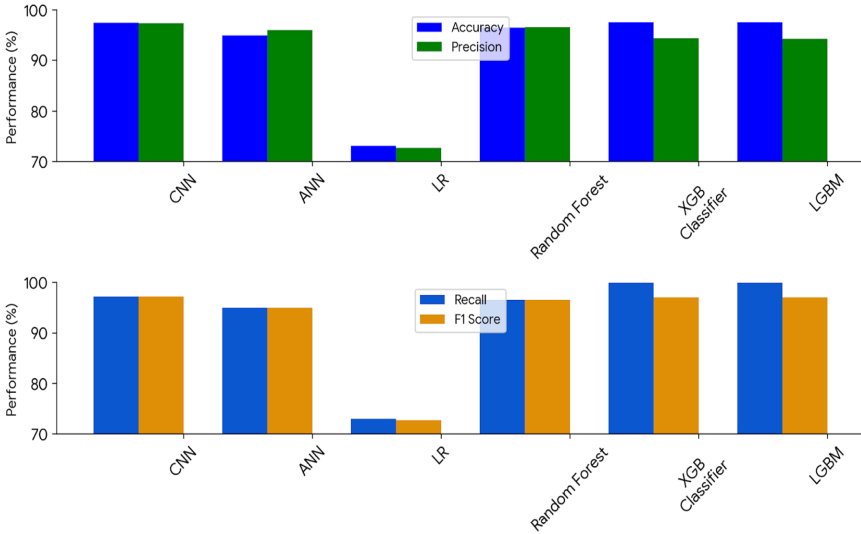
**Table 17** Performance metrics for machine learning models (active wiretap dataset) in a centralised (pre-FLAT) setup

| <i>Model</i>                    | <i>Accuracy (%)</i> | <i>Precision (%)</i> | <i>Recall (%)</i> | <i>F1-score (%)</i> |
|---------------------------------|---------------------|----------------------|-------------------|---------------------|
| Logistic regression             | 73.08               | 72.7                 | 73.02             | 72.78               |
| Random forest (ten estimators)  | 96.55               | 96.588               | 96.552            | 96.559              |
| Light gradient boosting machine | 97.5409             | 94.316               | 99.9419           | 97.0476             |
| XGB classifier                  | 97.5483             | 94.355               | 99.914            | 97.055              |

### 8.2.1.3 Comparison of all models in a centralised (pre-FLAT) setup

From Figure 11 in a centralised pre-FLAT setup, CNN has the best performance in terms of accuracy, precision, recall, and F1-score. XGBM slightly outperforms CNN in terms of accuracy only. LGBM also outperforms CNN in terms of accuracy and recall. ANN also ranks high and consistent when you take accuracy, precision, and recall into consideration. Random forest outperforms ANN in terms of all evaluation metrics. LR comes last, with low evaluation metrics in all criteria.

**Figure 11** Visualisations of performance metrics of six models (centralised pre-FLAT) for active wiretap attack dataset (see online version for colours)



### 8.2.2 Device performance during centralised training (pre-FLAT) for active wiretap dataset

In a pre-FLAT setup, device performance is comparable to any centralised setup scenario there is high latency in receiving results, and this setup will not be feasible in situations with large datasets/computing resources and in conditions where there is low network bandwidth. Furthermore, there is a single/central point of attack, leaving it susceptible to privacy and security threats and leaving sensitive user data vulnerable (Yin et al., 2022).

Decentralised/federated learning with adversarial training (post-FLAT) setup for active wiretap dataset a: performance metrics from federated learning with adversarial training (post-FLAT) setup for active wiretap dataset.

From Figures 12, 13, and 14, based on the analysis of the visualisations over ten rounds of FLAT, Tables 18, 19, 20 and 21 are formulated.

**Figure 12** Average losses through ten FLAT rounds for active wiretap dataset (see online version for colours)

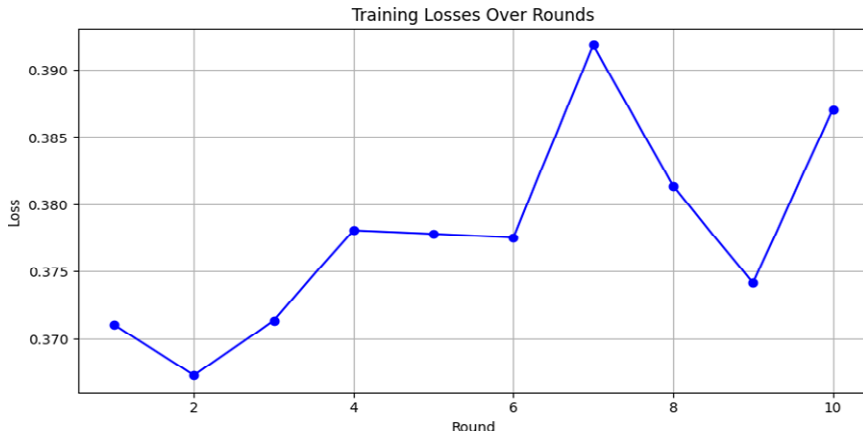


**Figure 13** Accuracy over ten FLAT rounds for active wiretap dataset (see online version for colours)

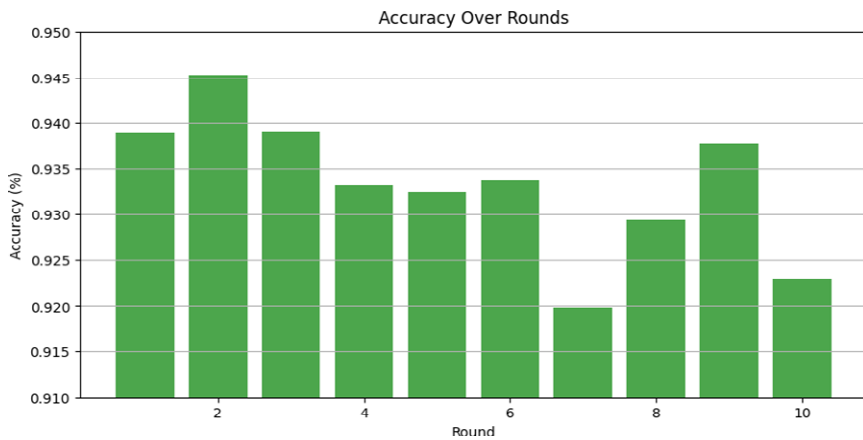


**Table 18** Average metrics for ten FLAT rounds (active wiretap dataset)

| Metric    | Average value (%) |
|-----------|-------------------|
| Accuracy  | 93.33             |
| Precision | 95.82             |
| Recall    | 87.30             |
| F1-score  | 91.35             |

**Figure 14** Precision, recall, F1-score over ten FLAT rounds for active wiretap dataset (see online version for colours)

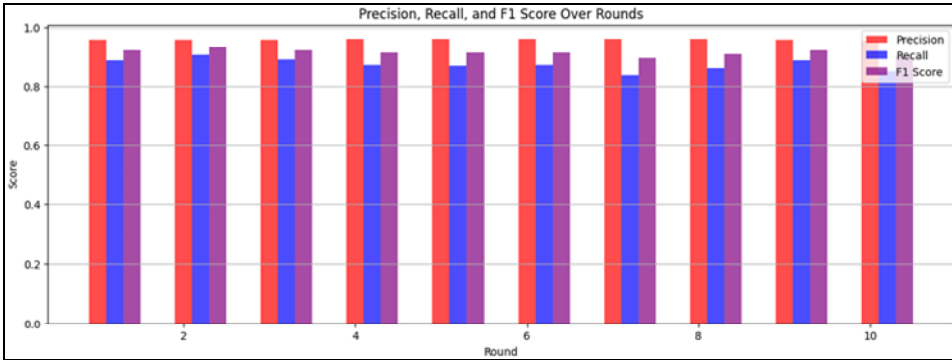


**Table 19** Rounds with highest metrics

| Metric    | Highest value (%) | Round number |
|-----------|-------------------|--------------|
| Accuracy  | 94.53             | 2            |
| Precision | 96.01             | 4            |
| Recall    | 90.52             | 2            |
| F1-score  | 93.05             | 2            |

**Table 20** Rounds with second highest metrics (active wiretap dataset)

| Metric    | Second highest value (%) | Round number |
|-----------|--------------------------|--------------|
| Accuracy  | 93.90                    | 3            |
| Precision | 96.01                    | 8            |
| Recall    | 88.98                    | 3            |
| F1-score  | 92.19                    | 3            |

From Figure 12 in a post-FLAT setup over ten rounds for the active-wiretap attack dataset, the loss values decrease over the first few rounds, indicating that the model is learning and improving. From Figure 15, the accuracy is generally high, ranging from approximately 91.99% to 94.53%.

From Figures 12, 13, and 14 in a post-FLAT setup over ten rounds for the active-wiretap attack dataset, it is observed that the fluctuations in accuracy across rounds are relatively small, suggesting stable FLAT model performance. Precision values consistently remain high, ranging from approximately 95.43% to 96.01%. High precision indicates that the FLAT model effectively minimises false positives. Recall values vary but generally stay within the range of 85.00% to 90.52%. F1-scores, which balance precision and recall, remain consistently high, ranging from approximately 89.92% to 93.05%. Slight fluctuations in losses might be attributed to various factors, such as changes in data distribution or complexity.

From Table 18 in a post-FLAT setup over ten rounds for the active-wiretap attack dataset, it is observed that on average, the FLAT model correctly predicts the class of approximately 93.33% of instances. The FLAT model achieves a high precision, indicating a low rate of false positives. Approximately 95.82% of predicted positive

instances are true positives. The recall, or sensitivity, is slightly lower at 87.30%, suggesting that the FLAT model might miss some positive instances. The F1-score, which balances precision and recall, is at 91.35%. The FLAT model demonstrates strong overall performance, particularly in terms of accuracy and precision. The F1-score, considering both precision and recall, reflects a well-balanced model.

From Table 19 in a post-FLAT setup over ten rounds for the active-wiretap attack dataset, it is observed that during the second round, the FLAT model achieved its highest accuracy, correctly predicting the class for approximately 94.53% of instances. In the fourth round, the FLAT model exhibited the highest precision, indicating a low rate of false positives. Around 96.01% of predicted positive instances were true positives. The second round saw the highest recall, or sensitivity, with the FLAT model capturing approximately 90.52% of actual positive instances. In the second round, the F1-score reached its peak at 93.05%, reflecting a good balance between precision and recall. Round 2 stands out as particularly strong, achieving the highest values across multiple metrics. High precision in round 4 indicates a low rate of false positives, while high recall in round 2 suggests effective capture of positive instances. The F1-score in round 2 indicates a well-balanced performance considering both precision and recall.

From Table 20 in a post-FLAT setup over ten rounds for the active-wiretap attack dataset, it is observed that with regards to the second highest metrics, the FLAT model achieved an accuracy of 93.90% in round 3, reflecting a strong overall predictive performance during this training iteration. In round 8, the FLAT model exhibited a high precision of 96.01%, indicating a low rate of false positives among the predicted positive instances. Round 3 saw the second-highest recall at 88.98%, suggesting a good ability to capture a significant proportion of actual positive instances. The F1-score peaked at 92.19% in round 3, showcasing a balanced performance between precision and recall during this round. Round 3 stands out as a round with consistently strong performance across accuracy, precision, recall, and F1-score. Round 8 demonstrated particularly high precision, indicating a robust ability of the FLAT to minimise false positives.

From Table 21 in a post-FLAT setup over ten rounds for the active-wiretap attack dataset, it is observed that with regards to the lowest metrics, during the seventh round, the FLAT model faced challenges, resulting in the lowest accuracy of 91.99%. This indicates a relatively higher number of incorrect predictions. In the tenth round, precision dropped to 95.45%, indicating a slightly higher rate of false positives among the predicted positive instances. Round 7 experienced the lowest recall, suggesting that the FLAT model missed a notable proportion of actual positive instances, capturing around 83.69%. The F1-score hit its lowest point in round 7 at 89.41%, indicating a temporary reduction in the balance between precision and recall. One can investigate the characteristics of the data in round 7 to identify challenges or shifts in the distribution and can consider introducing additional features to enhance overall robustness.

**Table 21** Rounds with lowest metrics (active wiretap dataset)

| <i>Metric</i> | <i>Second highest value (%)</i> | <i>Round number</i> |
|---------------|---------------------------------|---------------------|
| Accuracy      | 93.90                           | 3                   |
| Precision     | 96.01                           | 8                   |
| Recall        | 88.98                           | 3                   |
| F1-score      | 92.19                           | 3                   |

### 8.2.3 Individual client performance from federated learning with adversarial training (post-FLAT) setup for active wiretap dataset

From Figure 15, Tables 22, 23 and 24 are formulated. From Table 22 in a post-FLAT setup over ten rounds for the active-wiretap attack dataset, every 3, 4, 5, 7, and 9 clients consistently have lower losses, ranging from 0.814 to 0.817, indicating a relatively stable and effective learning process. Due to shared model parameters, IID based dataset distribution and a uniform client setup there are similar average client losses across rounds, ranging from 0.814 to 0.823. Every 1, 2, 8, and 10 clients have slightly higher losses. This suggests a bit more variability in the learning performance of these clients across different rounds. In general, the average losses are relatively low across all clients, which is a positive sign. It indicates that, on average, the clients are successfully minimising the loss during the adversarial training process. The average losses remain consistent or decrease over the 10 rounds, suggesting that the FLAT process is effective in improving the models of individual clients. From Table 23 in a post-FLAT setup over ten rounds for the active-wiretap attack dataset, when you consider client performance round-wise, client 9 appears multiple times as the best performer, suggesting a consistent ability to achieve lower losses compared to other clients in different rounds. Other clients, such as 8, 10, and 3, also make appearances as the best performers, indicating variability in which clients excel in different rounds. From Table 24 in a post-FLAT setup over ten rounds for the active-wiretap attack dataset, clients 6, 10, and 2 are identified as worst performers in multiple rounds, suggesting challenges or variability in their learning process across different rounds. Clients 1 and 9 appear as both best and worst performers in different rounds, indicating that their performance is more variable or sensitive to the training dynamics.

**Figure 15** Individual client losses for every FLAT round for active wiretap dataset (see online version for colours)

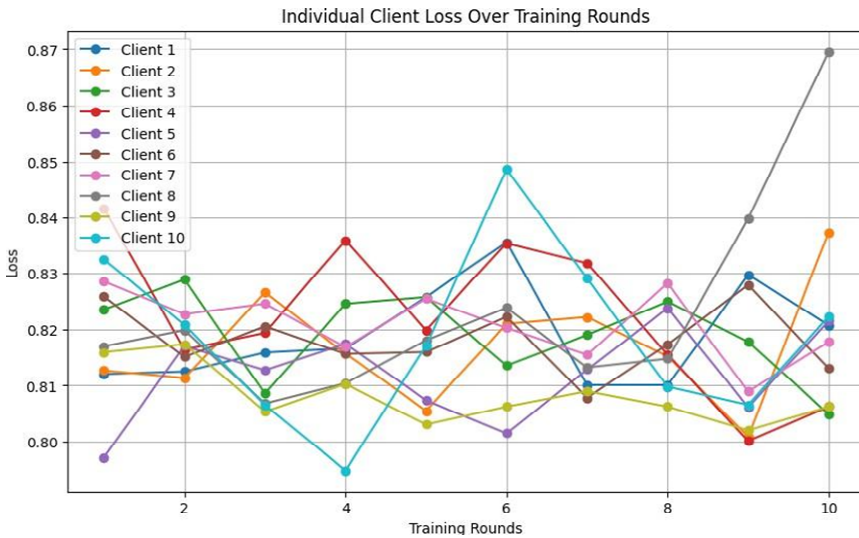


**Table 22** Average loss for every  $i^{\text{th}}$  client (active wiretap dataset)

| <i>Client</i> | <i>Average loss</i> |
|---------------|---------------------|
| Client 1      | 0.821               |
| Client 2      | 0.818               |
| Client 3      | 0.815               |
| Client 4      | 0.816               |
| Client 5      | 0.816               |
| Client 6      | 0.823               |
| Client 7      | 0.817               |
| Client 8      | 0.817               |
| Client 9      | 0.814               |
| Client 10     | 0.822               |

**Table 23** Round-wise best-performing clients (active wiretap dataset)

| <i>Round</i> | <i>Best-performing client</i> |
|--------------|-------------------------------|
| 1            | Client 8                      |
| 2            | Client 9                      |
| 3            | Client 10                     |
| 4            | Client 9                      |
| 5            | Client 1                      |
| 6            | Client 7                      |
| 7            | Client 9                      |
| 8            | Client 3                      |
| 9            | Client 9                      |
| 10           | Client 4                      |

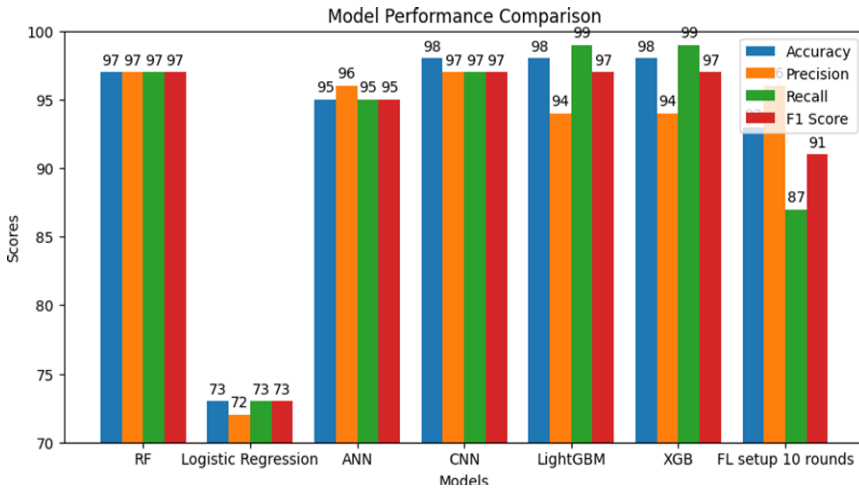
**Table 24** Round-wise worst-performing clients (active wiretap dataset)

| <i>Round</i> | <i>Worst-performing client</i> |
|--------------|--------------------------------|
| 1            | Client 6                       |
| 2            | Client 10                      |
| 3            | Client 2                       |
| 4            | Client 1                       |
| 5            | Client 8                       |
| 6            | Client 9                       |
| 7            | Client 1                       |
| 8            | Client 10                      |
| 9            | Client 2                       |
| 10           | Client 6                       |

### 8.2.4 Comparison of pre-FLAT and post-FLAT attack classification methodologies for active-wiretap attack dataset

From Figure 16, the following results can be deciphered. In a centralised, pre-FLAT approach, the best model is XGB with accuracy – 98%, precision – 94%, recall – 99% and F1-score – 97%. Almost all the evaluation metrics are high and they show a favourable recall. The second best is ANN with accuracy – 95%, precision – 96%, recall – 95% and F1-score – 95%. The third best is LGBM: accuracy – 98%, precision – 94% recall – 99% and F1-score – 97%. The post-FLAT decentralised model offers competitive performance in attack classification, with accuracy – 93%, precision – 96%, recall – 87%, and F1-score – 91%.

**Figure 16** Comparison of pre-FLAT and post-FLAT (ten rounds) classification methodologies for active wiretap attack dataset (see online version for colours)



## 9 Conclusions and future work

The FLAT model revolutionises model training by decentralising the process across devices, preserving data privacy locally, and minimising the need for centralised data consolidation. The model aims to show the increased security benefits of a cohesive model tackling the fundamental security paradigms of confidentiality, availability and integrity in the following ways. The FLAT approach offers competitive evaluation metrics in terms of accuracy, precision, recall, and F1-score in the network attack classification of the MiTM attack dataset and active wiretap dataset with a majority of the metrics being above 90%. Individual clients can perform attack classification on constrained resources and still offer stable performance with no sudden spike in losses. Furthermore, additional insights can be gained from the best and the worst performing rounds and the best and the worst performing clients. Clients with comparably higher losses and rounds that perform poorly in the FLAT setup can be investigated further to see if the client/round is affected by any adversary. Rounds where the FLAT model shows a drop in performance can also be identified. This approach not only addresses

privacy concerns but also reduces communication overhead, particularly beneficial in bandwidth-limited environments. FLAT enables model updates to be shared locally, optimising bandwidth usage and enhancing computing efficiency on edge devices. Adversarial training further strengthens models against attacks. Well-suited for distributed datasets in large-scale or geographically dispersed settings, FLAT adapts to dynamic environments, allowing real-time adjustments without constant retraining. In IoT setups, FLAT's decentralised approach ensures data privacy, accommodates heterogeneous devices, and provides a low-latency system aligned with edge computing, offering an extra layer of privacy and robustness during client training. The inclusion of increased accuracy, reliability, adaptability and better latency provided by these results show us a stronger security framework that could be implemented in place of existing methodologies. Although federated learning with adversarial training has significant privacy benefits for internet of things devices, resource constraints make it difficult to use on small devices. These gadgets usually feature a little memory capacity, a short battery life, and a poor computing power. Large data transfers or the training of complicated models can quickly deplete their resources and put a burden on network capacity. Adversarial training itself may also result in additional computing overhead. Even if federated learning at the edge and model distillation appear promising, the state-of-the-art may not be appropriate for all small IoT devices. It is imperative to thoroughly assess the compromises between privacy, performance, and resource usage prior to implementing federated learning with adversarial training on these kinds of devices. To further improve upon this work, one could explore the effectiveness of federated learning in training with clients possessing varying local training datasets. This exploration could be particularly relevant in a multi-label attack classification scenario where some clients have extensive datasets while others have relatively smaller datasets, potentially impacting the model's ability to generalise to labels with fewer samples. This investigation would shed light on how federated learning can enhance performance in such diverse scenarios.

## References

- Bebin Josey, T and Misbha, D.S. (2022) 'Detection of attacks using attention-based Conv-LSTM and bi-LSTM in industrial internet of things', *International Conference on Automation, Computing and Renewable Systems (ICACRS)*, Pudukkottai, India, pp.402–407, DOI: 10.1109/ICACRS55517.2022.10029012.
- Beutel, D.J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Li, K.H., Parcollet, T., de Gusmão, P.P.B. and Lane, N.D. (2020) *Flower: A Friendly Federated Learning Research Framework*, arXiv preprint arXiv:2007.14390.
- Chauhan, P. and Atulkar, M. (2021) 'Selection of tree-based ensemble classifier for detecting network attacks in IoT', *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*, Pune, India, pp.770–775, DOI: 10.1109/ESCI50559.2021.9397033.
- Chen, X. and Zhang, N. (2020) 'Layer-wise adversarial training approach to improve adversarial robustness', *2020 International Joint Conference on Neural Networks*, Glasgow, UK, pp.1–8, DOI: 10.1109/IJCNN48605.2020.9206760.
- da Silva, R.C.C., Oliveira Camargo, M.P., Sanches Quessada, M., Claiton Lopes, A., Diassala Monteiro Ernesto, J. and Pontara da Costa, K.A. (2022) 'An intrusion detection system for web-based attacks using IBM Watson', in *IEEE Latin America Transactions*, February, Vol. 20, No. 2, pp.191–197, DOI: 10.1109/TLA.2022.9661457.

- Gungor, O., Rosing, T. and Aksanli, B. (2022) 'Dense-defense: diversity promoting ensemble adversarial training towards effective defense', *IEEE Sensors*, Dallas, TX, USA, pp.1–4, DOI: 10.1109/SENSOR52175.2022.9967204.
- Hussain, F. et al. (2021) 'A two-fold machine learning approach to prevent and detect IoT BotNet attacks', in *IEEE Access*, Vol. 9, pp.163412–163430, DOI: 10.1109/ACCESS.2021.3131014.
- Jiyal, S., Sheetlani, J. and Saini, R.K. (2022) 'Internet of things: a survey on air pollution monitoring, challenges and applications', *2022 7th International Conference on Computing, Communication and Security (ICCCS)*, Seoul, Korea, pp.1–6, DOI: 10.1109/ICCCS55188.2022.10079635.
- Korkmaz, A., Alhonainy, A. and Rao, P. (2022) 'An evaluation of federated learning techniques for secure and privacy-preserving machine learning on medical datasets', *2022 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, DC, USA, pp.1–7, DOI: 10.1109/AIPR57179.2022.10092212.
- Kousar, H., Mulla, M.M., Shettar, P. and N.D.G. (2021) 'DDoS attack detection system using Apache Spark', *2021 International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore, India, pp.1–5, DOI: 10.1109/ICCCI50826.2021.9457012.
- Lai, W. and Yan, Q. (2022) 'Federated learning for detecting COVID-19 in chest CT images: a lightweight federated learning approach', *2022 4th International Conference on Frontiers Technology of Information and Computer (ICFTIC)*, Qingdao, China, pp.146–149, DOI: 10.1109/ICFTIC57696.2022.10075165.
- Li, M., Gu, W., Chen, W., He, Y., Wu, Y. and Zhang, Y. (2018) 'Smart home: architecture, technologies and systems', *Procedia Computer Science*, Vol. 131, pp.393–400, DOI: 10.1016/j.procs.2018.04.219.
- Ling, Y. et al. (2022) 'Real-time detection of DDoS attacks based on Hurst Index', *2nd International Conference on Networking Systems of AI (INSAI)*, Shanghai, China, pp.42–45, DOI: 10.1109/INSAI56792.2022.00018.
- Mothukuri, V., Khare, P., Parizi, R.M., Pouriyeh, S., Dehghantanha, A. and Srivastava, G. (2022) 'Federated-learning-based anomaly detection for IoT security attacks', in *IEEE Internet of Things Journal*, 15 February, Vol. 9, No. 4, pp.2545–2554, DOI: 10.1109/JIOT.2021.3077803.
- Muhammad, K., Ullah, H., Khan, S., Hijji, M. and Lloret, J. (2023) 'Efficient fire segmentation for internet-of-things-assisted intelligent transportation systems', in *IEEE Transactions on Intelligent Transportation Systems*, November, Vol. 24, No. 11, pp.13141–13150, DOI: 10.1109/TITS.2022.3203868.
- Neto, E.C.P., Dadkhah, S. and Ghorbani, A.A. (2022) 'Collaborative DDOS detection in distributed multi-tenant IoT using federated learning', *19th Annual International Conference on Privacy Security and Trust (PST)*, DOI: 10.1109/PST55820.2022.9851984.
- Nikhani, N. and Wu, J. (2022) 'Entropy-KL-ML: enhancing the entropy KL-based anomaly detection on software-defined networks', in *IEEE Transactions on Network Science and Engineering*, 1 November–December, Vol. 9, No. 6, pp.4458–4467, DOI: 10.1109/TNSE.2022.3202147.
- Pan, Z., Geng, H., Wei, L. and Zhao, W. (2022) 'Adaptive client model update with reinforcement learning in synchronous federated learning', *32nd International Telecommunication Networks and Applications Conference (ITNAC)*, Wellington, New Zealand, pp.1–3, DOI: 10.1109/ITNAC55475.2022.9998360.
- Pontara da Costa, K.A. (2022) 'An intrusion detection system for web-based attacks using IBM Watson', in *IEEE Latin America Transactions*, February, Vol. 20, No. 2, pp.191–197, DOI: 10.1109/TLA.2022.9661457.
- Rahman, S.A., Tout, H., Talhi, C. and Mourad, A. (2020) 'Internet of things intrusion detection: centralized, on-device, or federated learning?', in *IEEE Network*, November/December, Vol. 34, No. 6, pp.310–317, DOI: 10.1109/MNET.011.2000286.

- Sahingoz, O.K., Çekmez, U. and Buldu, A. (2021) 'Intrusion detection system using IoT devices for safety and security', *Journal of Web Engineering*, September, Vol. 20, p.17211760, DOI: 10.13052/jwe1540-9589.2062.
- Shahid, O., Mothukuri, V., Pouriye, S., Parizi, R.M. and Shahriar, H. (2021) 'Detecting network attacks using federated learning for IoT devices', *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, Dallas, TX, USA, pp.1–6, DOI: 10.1109/ICNP52444.2021.9651915.
- Sharma, D., Kumar, A. and Battula, R.B. (2023) 'FedBeam: federated learning based privacy preserved localization for mass-beamforming in 5GB', *2023 International Conference on Information Networking (ICOIN)*, Bangkok, Thailand, pp.616–621, DOI: 10.1109/ICOIN56518.2023.10048980.
- Singh, S., Fernandes, S.V., Padmanabha, V. and Rubini, P. (2021) 'MCIDS – multi classifier intrusion detection system for IoT cyber attack using deep learning algorithm', *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, Tirunelveli, India, pp.354–360, DOI: 0.1109/ICICV50876.2021.9388579.
- Song, C. et al. (2018) 'MAT: a multi-strength adversarial training method to mitigate adversarial attacks', *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Hong Kong, China, pp.476–481, DOI: 10.1109/ISVLSI.2018.00092.
- Sun, Y., Ochiai, H. and Esaki, H. (2022) 'Decentralized deep learning for multi-access edge computing: a survey on communication efficiency and trustworthiness', in *IEEE Transactions on Artificial Intelligence*, December, Vol. 3, No. 6, pp.963–972, DOI: 10.1109/TAI.2021.3133819.
- Trajanovski, T. and Zhang, N. (2021) 'An automated and comprehensive framework for IoT BotNet detection and analysis (IoT-BDA)', in *IEEE Access*, Vol. 9, pp.124360–124383, DOI: 10.1109/ACCESS.2021.3110188.
- Wang, A. (2020) 'Internet of things computer network security and remote control technology application', *2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE)*, Harbin, China, pp.1814–1817, DOI: 10.1109/ICM-CCE51767.2020.00398.
- Xu, J. (2020) 'Generate adversarial examples by Nesterov-momentum iterative fast gradient sign method', *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China, pp.244–249, DOI: 10.1109/IC-SESS49938.2020.9237700.
- Yin, C., Zhang, S., Wang, J. and Xiong, N.N. (2022) 'Anomaly detection based on convolutional recurrent autoencoder for IoT time series', in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, January, Vol. 52, No. 1, pp.112–122, DOI: 10.1109/TSMC.2020.2968516.
- You, M. et al. (2022) 'FuzzDocs: an automated security evaluation framework for IoT', in *IEEE Access*, Vol. 10, pp.102406–102420, DOI: 10.1109/ACCESS.2022.3208146.
- Zou, D. et al. (2022) 'FedMC: federated reinforcement learning on the edge with meta-critic networks', *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, Austin, TX, USA, pp.344–351, DOI: 10.1109/IPCCC55026.2022.9894336.