MÄLARDALEN UNIVERSITY
SCHOOL OF INNOVATION, DESIGN AND ENGINEERING
VÄSTERÅS, SWEDEN

Thesis for the Degree of Bachelor of Science in Engineering – Computer Network Engineering – 15 credits

ANALYZING COMMUNICATION–ACCURACY TRADE-OFFS IN FEDERATED LEARNING-BASED NETWORK INTRUSION DETECTION SYSTEMS

Hampus Holmberg
hhg21004@student.mdu.se
Hugo Elfstrand
hed23003@student.mdu.se

Examiner: Svetlana Girs
Mälardalens universitet, Västerås, Sverige

Supervisor: Maryam Vahabi
Mälardalens universitet, Västerås, Sverige

18 June 2026

Abstract

Modern smart grid technology improves system efficiency and reliability, but it also increases the attack surface. This threat to smart grid security creates a need for intrusion detection systems (IDS) to help combat the problems. Centralized machine-learning-based IDS is one solution, but it comes with drawbacks in terms of privacy, scalability, and communication. Federated learning (FL) is an emerging technology that aims to address these limitations and provide better privacy and security for the system. This thesis investigates the communication-accuracy trade-off in network-based intrusion detection systems (NIDS) using federated learning and compares three neural network architectures: CNN, LSTM, and a hybrid architecture, CNN-LSTM. To investigate this, an experiment was conducted in a simulated FL environment with one server and three clients, all using Flower as the FL framework. Each client was trained and evaluated using the Edge-IIoT dataset, with testing at 1, 3, 5, 7, and 10 rounds of federated learning. Network conditions were emulated using tc-netem, introducing packet loss ranging from 1% to 10% during federated learning. Results indicate that the CNN-LSTM architecture had the best communication-accuracy trade-off, showing high accuracy and low communication costs. CNN reached high accuracy quickly, but fluctuations prevented the model from stabilizing. The largest model was LSTM, which also had the highest communication costs but still achieved strong classification performance. Packet loss caused a disproportionately large increase in communication overhead. Findings in this thesis suggest that important factors in FL NIDS include communication cost, convergence, accuracy, and network stability.

Table of contents

1. Introduction 6

2. Background 7

2.1. Machine Learning 7

2.1.1. Deep Learning 7

2.2. Intrusion Detection System and Network-based Intrusion Detection System 7

2.2.1. Machine Learning in Network Intrusion Detection Systems 8

2.2.2. Challenges in Distributed Environments 8

2.3. Federated Learning 8

2.3.1. FL workflow and client-server training 9

2.3.2. Federated Averaging 9

2.3.3. Privacy and distributed learning 9

2.3.4. Communication in Federated Learning 10

2.3.5. Model synchronization 10

2.3.6. Bandwidth consumption 10

2.3.7. Unreliable communication 10

2.3.8. Client heterogeneity 10

2.4. Neural Network Architectures 11

2.4.1. CNN 11

2.4.2. LSTM 11

2.4.3. Hybrid CNN-LSTM architecture 11

2.5. Edge-IIoT Dataset 12

3. Problem formulation 13

4. Related work 14

5. Method 16

6. Ethical and Societal Considerations 17

7. Experimental Design and Implementation 18

7.1. Federated Learning System Architecture 18

7.2. Neural Network Architectures 18

7.2.1. CNN 19

7.2.2. LSTM 19

7.2.3. CNN-LSTM 20

7.2.4. Parameter counts and model sizes 21

7.3. Dataset and Preprocessing 22

7.3.1. Dataset description 22

7.3.2. Feature preprocessing 22

7.3.3. Data splitting 22

7.3.4. Scaling 22

7.3.5. Label handling 23

7.4. Federated Training Configuration 23

7.5. Experimental Scenarios 23

7.5.1. Packet loss 23

7.5.2. Experimental setup 24

7.5.3. Definitions and performance metrics 24

8. Results 26

8.1. Communication Cost Across Federated Architectures 26

8.2. Convergence and Detection Performance in Federated Learning 27

8.3. Communication-Accuracy Trade-Off Analysis 29

8.4. Federated Learning under Packet Loss Conditions 30

9. Discussion 34

10. Conclusions 36

11. Future work 37

References 38

1. Introduction

A recent development in modern electrical infrastructure is smart grid technology. Smart grid technology is the integration of advanced communication and monitoring systems into the electricity grid [1]. This directly leads to a more efficient and reliable grid system.

The increase in communication among smart grid units expands the attack surface, making them more susceptible to attacks [2]. Because of this, there is a need for cybersecurity in smart grid technology, and one possible solution is intrusion detection systems. Most intrusion detection system solutions for smart grids rely on centralized learning, in which data is stored on a central server, introducing privacy risks, scalability issues, and communication overhead.

Federated learning (FL) is a promising approach because it is decentralized, with no transfer of private or sensitive data among networked devices during model training [3]. In this work, we will specifically compare three different models: Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) and a hybrid architecture, CNN-LSTM. These systems aim to maintain low communication frequency, but model accuracy is also important. Accuracy is the model's estimate of whether a given traffic is an attack or normal traffic. Investigating these trade-offs in accuracy and communication makes this thesis research both new and interesting.

The purpose of this work is to investigate and analyze the trade-offs between communication and detection accuracy, and to compare CNN, LSTM, and CNN-LSTM in the same federated learning setup. The intended audience for this thesis is similarly educated people in network engineering with an interest in federated learning, smart grid systems, and intrusion detection systems.

Other studies have been conducted in this area of federated learning and intrusion detection, but with some key differences. Some research focuses on scaling in federated learning environments [6]; others focus on improving privacy [4] and on model degradation [5], but few studies investigate communication-accuracy trade-offs.

By setting up a simulated environment, we performed experiments to answer these questions. The environment consists of four virtual machines, one acting as a central server and three acting as nodes to that central server. Flower is used for the federated learning framework, and the publicly available Edge-IIoT dataset is used to train our models.

The key findings of this research indicate that CNN-LSTM is the most effective architecture for reducing communication costs while still achieving high accuracy.

2. Background

In this section, we provide the necessary information to understand the rest of the work.

2.1. Machine Learning

Machine learning (ML) is a subfield of artificial intelligence (AI) that focuses on developing algorithms and models capable of learning from data [7]. These algorithms and models allow computers to learn from data and make decisions or predictions based on identified patterns.

Common types of machine learning include supervised, unsupervised, self-supervised, reinforcement, and semi-supervised learning [7], [8]. Supervised learning is trained from labeled datasets to predict outcomes on new data by learning to recognize correlations between input and output data. The trained model is then evaluated on test data to see if it has been trained correctly. Unsupervised learning is trained on unlabelled data to learn patterns and underlying structures. Reinforcement learning focuses on trial and error, using feedback from its actions to optimize its performance. Self-supervised learning is an approach in which the model generates its own supervisory signals from the data's structure. It is a mix of supervised and unsupervised learning. Semi-supervised learning is another hybrid that combines supervised and unsupervised learning. It is trained using a smaller amount of labeled data and a larger set of unlabelled data to achieve the same results as supervised learning, but with fewer labels.

2.1.1. Deep Learning

Deep learning (DL) is a branch of machine learning that uses neural networks inspired by the human brain [9]. These neural networks can learn patterns from large amounts of data without needing manual input. Deep learning models require substantial computational power for training. It works by using multilayer neural networks to process data, and each layer extracts features from the input. To learn, the model adjusts weights to improve prediction accuracy.

2.2. Intrusion Detection System and Network-based Intrusion Detection System

Intrusion Detection System (IDS) is a type of protection system against cyber-attacks. It monitors network or host activity to detect potential violations of security policies and other suspicious activity [10]. It monitors both inbound and outbound traffic and is usually strategically placed within a network to better identify and track it. Some examples of attacks that IDS can detect include spoofing, bot/botnet attacks, Denial of Service attacks, Probing attacks, and Sinkhole attacks. One of the earliest forms of IDS was host-based IDS (HIDS), which monitors activities and events on individual host machines. A major limitation of HIDS is its inability to detect network attacks happening outside of the HIDS.

Network-based IDS (NIDS) is a technique in which an IDS is deployed at the network edge to monitor and analyze network traffic for potential intrusions [10]. Network traffic is the key to NIDS, and this system can monitor an entire network, unlike HIDS.

Within IDS, there are two detection approaches, signature-based and anomaly-based [10], [11]. In signature-based intrusion detection, the attack patterns are stored in a database and compared against observed network traffic. This technique is not effective for detecting unknown attacks and requires maintaining a large dataset. However, it gives high accuracy for known attack types. In contrast, anomaly-based intrusion detection operates by learning normal system behavior and detecting significant deviations from that behavior. This technique gives the system a high chance of detecting unknown attacks, but it may also generate false alarms when legitimate changes occur in normal traffic behavior.

2.2.1. Machine Learning in Network Intrusion Detection Systems

The use of ML to advance IDS was introduced in 1987 [10]. Traditional IDS suffer from very high false alarm rates (FAR), and two approaches used to address this problem are DL and ML. Learning from data is more dynamic than relying on predefined rules, as in normal IDS. This leads to higher accuracy in most cases, but algorithms perform differently depending on the dataset used and the type of attack. There are many developments in machine learning and deep learning algorithms.

2.2.2. Challenges in Distributed Environments

An issue with deep neural network-based NIDS is that attackers can exploit manipulated inputs to trick models, posing a serious security concern [11]. Adversarial attacks are methods that cause the DL algorithm to misclassify the input, and they have inspired researchers to develop countermeasures.

Another challenge in NIDS is concept drift [12]. Over time, network traffic distributions change and new types of attacks emerge, which invalidate older models trained on outdated datasets. The performance degrades over time.

Models trained on one type of network environment usually fail when introduced to another [8]. This is a problem for cases with heterogeneous network traffic.

2.3. Federated Learning

Federated learning is an emerging, decentralized technique, in contrast to traditional centralized machine learning model training [13]. Within a network comprising several nodes and a centralized server, the nodes train the model on local data. The central server distributes the global model to the nodes and aggregates their updates to improve it. Federated learning is

a growing technique because nodes retain important, potentially sensitive data locally. As a result, that removes the potential risk of transferring sensitive data to the centralized server.

Flower is a federated learning framework designed to connect simulation-based research and real-world deployment on edge devices [14]. The framework orchestrates training rounds involving a subset of clients, local data, aggregation, and parameter updates. Flower provides built-in strategies such as FedAvg and FedProx. ClientProxy is an object that means the server does not communicate with clients directly, making Flower communication-agnostic. Flower is also language-agnostic because of its language-independent protocol.

2.3.1. **FL workflow and client-server training**

A Federated Learning system usually consists of a group of clients and a central server [15]. An epoch is a complete processing of the local dataset.

The FL workflow starts with the central server initializing the global model's parameters, then broadcasting it to all clients. The cycle starts with clients using the global model to update their local models, then training the local models on their own data. After the local model has completed training for a predefined number of epochs, the client sends its updated model parameters to the central server. All clients do this, and after receiving all local model parameters, the central server executes aggregation and finally updates the global model. The new global model is sent to all clients. This is a round of training in federated learning. And the cycle usually continues until the model converges.

2.3.2. **Federated Averaging**

Federated Averaging Algorithm (FedAvg) [16] is a commonly used aggregation algorithm in FL that combines locally trained model updates from multiple clients to produce a shared global model. Instead of sharing raw training data, clients transmit only updated model parameters to the central server, thereby improving data privacy and reducing the need for centralized data collection. The central server collects the model updates by computing a weighted average of the client parameters.

2.3.3. **Privacy and distributed learning**

Network traffic data can contain sensitive information, so a centralized approach introduces risks to both security and legal aspects [16]. Federated learning solves this by ensuring that local data never leaves its client.

2.3.4. **Communication in Federated Learning**

The biggest constraint in federated learning is communication costs, since each round of training requires distributing the global model to clients and transmitting their locally updated model parameters back to the server [16]. FedAvg helps reduce communication costs by allowing clients to perform more local training epochs before sending their updates, thereby reducing the total number of training rounds.

2.3.5. **Model synchronization**

Standard FL uses a synchronous training model in which the server selects clients, broadcasts the global model, and repeats the cycle as mentioned before. An issue with this is the straggler problem [17]. The straggler problem arises when clients differ in training speed; slower clients can slow down the entire process because the central server waits for them to upload parameters. This problem worsens with scale, and the solution is asynchronous FL (AFL). With AFL, the central server aggregates updates as they are received. The efficiency of training improves with AFL, but communication overhead increases because clients with greater computational power upload their updates multiple times per training round before other clients have finished.

2.3.6. **Bandwidth consumption**

To mitigate communication costs, compression techniques can reduce the size of updates [18]. Examples of these techniques are quantization, pruning, and sparsification. Another way is to select clients who participate in each round, meaning only a fraction of clients communicate in each round and therefore save on communication costs.

Using more epochs to shift more training to local clients can reduce communication costs [16]. The local hardware performs the computation, reducing the number of rounds needed to reach convergence; fewer rounds mean less communication.

2.3.7. **Unreliable communication**

Real network environments are unreliable, particularly IoT [16]. Disruptions such as packet loss and latency degrade performance, making them a large bottleneck for FL [18]. This can affect FL performance.

2.3.8. **Client heterogeneity**

Federated learning systems typically operate under non-independent and identically distributed (non-IID) conditions, meaning each client has different characteristics and attacks in its dataset, leading to slower convergence and lower detection accuracy [19].

There are two parts to client heterogeneity: latency heterogeneity, the differences in edge devices' hardware and bandwidth speeds [20]. The second part is data heterogeneity, meaning that local data on edge devices may have non-IID data distributions. The difference can result in slower convergence and worse overall performance.

2.4. Neural Network Architectures

There are many neural network architectures, but common categories are feedforward neural networks, convolutional neural networks, autoencoders, transformer-based models, and recurrent neural networks [21]. CNN, LSTM, and CNN-LSTM are covered in this background section because of their relevance to this work.

2.4.1. CNN

Convolutional neural networks (CNNs) are deep learning models designed to detect features within images [22], [23]. They process two- or three-dimensional data for image classification and object recognition. CNNs use three layers in their architecture: convolutional, pooling, and fully connected.

Before being processed by the model, input images are preprocessed to ensure a consistent size and format across the dataset. [23]. In the convolutional layers, filters are added to extract features of the given image. In pooling layers, the number of parameters is reduced, and a filter without weights is applied to the input. This leads to reduced complexity, less risk of overfitting, and improved efficiency at the cost of some spatial information loss. After pooling layers, the downsampled feature maps are processed in the fully connected layers to produce output in the form of classification labels based on features found in the input.

2.4.2. LSTM

Long short-term memory (LSTM) is a type of recurrent neural network (RNN) designed to process long sequences of data and learn from them [24]. The key distinction is that LSTMs are designed to retain information over long sequences, whereas standard RNNs struggle with long-range dependencies due to the vanishing gradient problem. It retains information over long sequences by storing and selectively updating a memory cell. There are three parts of the memory cell in the LSTM architecture: the input gate, forget gate, and output gate. The input gate controls which information is added to the memory cell. The forget gate controls what is removed from the memory cell, and the output gate controls the output.

2.4.3. Hybrid CNN-LSTM architecture

CNN-LSTM is a hybrid architecture of CNN and LSTM, built with layers from both neural network architectures [25]. The CNN layers in this hybrid architecture extract features from the input data, which are then reshaped and fed into LSTM layers. This way, CNN finds

important local patterns, and LSTM learns sequential patterns and how patterns change over time.

2.5. **Edge-IIoT Dataset**

The Edge-IIoT dataset consists of realistic data from IoT and IIoT networks and is intended for use with federated and distributed machine learning algorithms [26]. Various IoT and IIoT sensory devices were used to build the network and generate the dataset. Sensors include temperature, humidity, and pH sensors, among others. Traffic was recorded at the edge, fog, and cloud computing layers to reflect realistic IoT network conditions. The dataset consists of 157801 to ~2.2 million data points, depending on the version, 61 features, and 15 classes. Data points represent the total number of traffic connections during the testbed experiments [27]. Each data point is one instance of communication between devices. Features are attributes extracted from each traffic connection; basic examples include packet length, port number, and protocol type. Classes are labeled for each data point, which indicates the type of traffic [26]. In Edge-IIoT, there are 14 attack classes and a normal traffic class, for a total of 15 classes.

Datasets contain large amounts of organized data in various formats such as CSV, JSON, or database tables [28]. This is important for things like ML and data analysis where large amounts of reliable data are needed. The data is often structured in rows and columns with defined fields, aiding readability and analysis. Analyzing datasets can reveal underlying patterns in the data.

There are five attack categories in the Edge-IIoT dataset: DoS/DDoS, information gathering, Man-In-The-Middle, injection, and malware [26]. In those categories are 14 types of cyberattacks: TCP SYN Flood, UDP Flood, HTTP Flood, and ICMP Flood for DoS/DDoS. Port Scanning, OS Fingerprinting, and Vulnerability Scanning for information gathering. DNS & ARP Spoofing and XSS for Man-In-The-Middle. SQL Injection, Uploading, and Backdoor for injection. Password cracking and ransomware for malware.

3. Problem formulation

Modern smart grids integrate digital communication, monitoring, and control technologies to improve the efficiency and reliability of electrical power systems. The large number of interconnected devices continuously generates operational and network traffic data, thereby increasing the attack surface and creating new cybersecurity challenges.

Traditionally, machine learning models have been trained using centralized learning approaches. Data gets collected from multiple users or devices and processed on a central server. While this approach simplifies model training, it also introduces several challenges related to privacy, security, and communication overhead.

For environments such as industrial and smart grid systems, transmitting large amounts of potentially sensitive data to a centralized server may be undesirable or impractical. These limitations have led to increased interest in distributed learning approaches such as federated learning.

One solution that has gained traction for addressing privacy issues is to decentralize machine learning. Decentralizing allows the data itself not to be sent directly to the server. This is where Federated Learning (FL) comes in. FL is an approach to machine learning that enables clients to collectively train a model without sending data to a central server, sending only model updates. This would effectively solve the privacy issue that a centralized approach faces.

This thesis aims to investigate how much communication is required between clients and the server to build an effective model while maintaining an acceptable level of accuracy.

The research questions (RQ) that this thesis aims to answer are the following:

> RQ1: How does the communication cost evolve during federated training when using different neural network architectures?
>
> RQ2: How do CNN, LSTM and CNN-LSTM models differ in terms of convergence speed and detection accuracy in a federated learning setup?
>
> RQ3: What trade-offs can be observed between communication cost and achieved model accuracy for the considered architectures?

This thesis is limited by the scale of the environment and the number of model architectures evaluated. The study focuses on two commonly used neural network architectures for intrusion detection and a hybrid combining them. and evaluates them in a federated learning setup comprising one server and three clients.

4. Related work

Di Xiao et al. [4] compare ECPFL, their proposed FL framework, against advanced federated learning algorithms, analyzing accuracy, communication overhead, and privacy budget. They also analyze the communication overhead and privacy budget of ECPFL, an FL framework. They argue that most FL systems optimize only one to two of accuracy, privacy, and communication efficiency, and that their proposed method is a solution. The ECPFL framework is meant to optimize model accuracy and communication efficiency while guaranteeing privacy. The system works like a traditional federated learning architecture but with a few modifications. The system builds on a traditional federated learning architecture with three key modifications: a contrastive-learning-inspired regularization term to reduce client drift, a bidirectional top-k sparsification method to compress communication, and a shuffle-model-based differential privacy mechanism that leads to reduced privacy budget growth. From their results, they report a 90% reduction in communication costs, improved robustness, and improvements in accuracy. A problem is that clients train on different datasets, causing drift in various directions and ultimately leading to lower accuracy, slower convergence, and unstable aggregation. Their solution to this is one of the parts mentioned earlier: a contrastive-learning-inspired regularization term. They add a regularization term to the model's local training to keep it close to the current global model, so the regularizer penalizes harmful drift. They report an average improvement of about 3% in accuracy over FedAvg. Another problem is massive models, which are commonplace today and cause communication issues due to their size. Their proposed solution is based on the top-k sparsification method, which only sends the largest parameters to reduce communication costs. Bidirectional top-k sparsification means they compress both directions of traffic, uplink and downlink. The last problem they address is privacy and how it is typically handled in FL systems. In traditional differential privacy federated learning (DP-FL), each client adds noise to its gradients, leading to an explosion in the privacy budget. The proposed solution is to combine local differential privacy to prevent the exposure of raw gradients, subsampling amplification, and a shuffle model, in which the server sees updates but not which client produced them. Their work addresses the same core subject we investigate in this thesis: balancing communication costs and model performance in federated learning. In Di Xiao et al.'s work, they use the ECPFL framework, whereas in our setup, we use standard neural network architectures without modifications. Their findings on communication cost reduction give an idea of how much can be optimized in federated learning.

Dos Santos et al. [5] analyze the degradation over time that happens in traditional ML-based intrusion detection due to changes in network traffic. Retraining the models is costly and fraught with privacy concerns. Organizations do not want to share their network traffic, and when models age, false positives and negatives increase. They propose an approach using federated learning and a reject-option classifier, which improves false-positive and false-negative rates without retraining the model. Their results show an improvement of 12-13% for

the false-positive rate and 9,6% for the false-negative rate. 3,6% of events needed rejection, and 0,3% of events were needed for model updates. Incoming network flows are classified by an IDS model and assigned a confidence score, which is either accepted or rejected. Rejected traffic is used for retraining locally, and model updates and weights are sent to a central server. Compared to traditional IDS — which retrains on full datasets and always predicts — this approach retrains selectively and can abstain from classifying low-confidence events. For evaluation, they use the MAWIFlow dataset, which contains over 7 terabytes (TB) of real traffic. In their experiment, they train on old traffic and test on future traffic, which lessens concept drift, and they criticize other papers using shuffled datasets, which hide the problem of concept drift. This paper focuses on aging models, reliability, costs, and privacy, while other papers prioritize optimizing accuracy. Like our thesis, this paper addresses IDS in FL, making it relevant to our work. However, a key difference is that they focus on model degradation, privacy, and reliability. Using a dataset containing real traffic, combined with their criticism of shuffled datasets, makes for valid points worth acknowledging as a limitation in our thesis.

Albanbay et al. [6] analyze and evaluate the performance in federated learning-based intrusion detection. The work focuses on the Internet of Things, which differs from this work, though many similarities exist. Their work uses the CICIoT2023 dataset, which they argue is appropriate due to its scale, diversity, high fidelity, and recentness. The federated setup consists of four different sizes, 10, 50, 100, and 150 devices, and the dataset was equally partitioned between every device. DNN, CNN, and CNN+BiLSTM are the models used and evaluated in this setup. In the experimental setup, 10 epochs and 20 rounds are used. The study examines how different neural architectures behave under FL, the effect of client scaling on performance, and the impact of constrained hardware on model quality. The experiment here follows the standard FL architecture, and as the client count increases, each client receives less local data, synchronization becomes harder, and data heterogeneity rises. Their results show that as the number of clients rises and local data shrinks, performance drops, convergence becomes slower, and smaller local datasets hurt generalization. CNN remains more stable than DNN and CNN-BiLSTM as the client count scales. CNN was the best practical model they tested, with good scalability, efficient inference, and strong accuracy. They found that more clients do not necessarily mean better performance, so there is a scaling limit. They conclude that federated learning can preserve privacy and achieve strong IDS performance without centralizing data. Practical deployment requires balancing model complexity, communication cost, and resources. Albanbay et al.'s work is related to this thesis as it evaluates different neural network architectures in a federated learning environment for intrusion detection. Their findings are important, most notably that CNN scales best, which makes for an interesting comparison to our own findings about CNN. The key difference between this work and ours is that they focus on client scaling, whereas we investigate communication overhead and the trade-offs between communication cost and achieved model accuracy.

5. Method

This thesis will use two scientific methods: a literature review and an experimental evaluation. The purpose of a literature review is to get a deeper understanding of Federated Learning and establish the required knowledge for this thesis. In addition to Federated Learning, the review covers three distinct machine learning models and the Flower framework. This knowledge provides a solid foundation for building the thesis.

The performative analysis is a method commonly used in engineering. Performative analysis aims to quantify performance by looking at different measurable performative indicators. The indicators this thesis will examine are accuracy, communication efficiency, network load, and convergence performance.

To obtain the data for the analysis, an experiment will be conducted in a controlled offline environment. This ensures that the results are not impacted by external factors such as a faulty connection. The experiment will be run multiple times across different setups and round counts.

6. Ethical and Societal Considerations

This work is based upon a publicly available dataset (Edge-IIoT), and no collection, storage, or processing of personal or sensitive data is used in this thesis. We believe our work does not involve any questions in research ethics.

We acknowledge that a core aspect of our work is intrusion detection, which includes security and privacy. Federated learning is a technique that addresses privacy and security weaknesses in centralized machine learning, so our work is of interest in that regard. The results of this thesis work are to be handled responsibly.

We emphasize that the results of this work should be used to advance future work in networking and federated learning.

7. Experimental Design and Implementation

This section is used to provide all the details of the experiments.

7.1. Federated Learning System Architecture

The environment is configured to use three clients and one server, all running Debian. The setup was built with Proxmox. In this setup, the virtual machines are allocated 4 GB of RAM and 1 CPU core. To enable hardware-accelerated machine learning in the setup, the CPU must be set to `host` type. The number of clients was limited by hardware resources.

Static IP addresses were assigned to simplify network configuration and communication between the federated learning clients and server. The Flower framework communication ports were configured to allow client-server communication without firewall filtering because the packets must not be filtered by a firewall. The experiments were implemented using the Flower framework. Flower was chosen because it is framework-agnostic. This allowed multiple different frameworks to be used, such as PyTorch, TensorFlow, and many others, while not locking the project into a single framework. Figure 1 depicts the setup of the experiment. At the top is the server, with the clients being at the bottom portion of the figure.

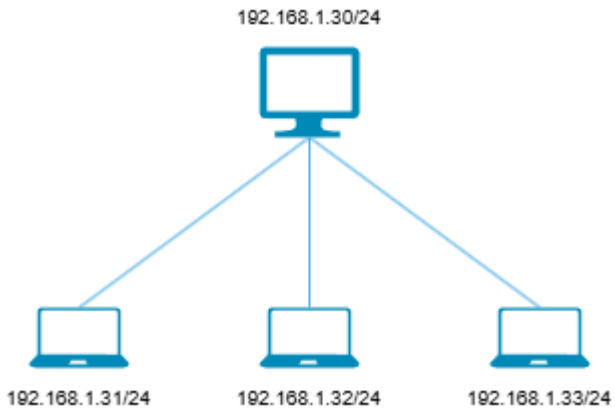

Figure 1: The figure depicts the setup for this project.

7.2. Neural Network Architectures

There are currently over 30 different machine learning architectures, not counting any hybrid architectures such as CNN-LSTM. Based on our research, the models deemed most suitable for this thesis were CNN, LSTM, and CNN-LSTM. CNN was chosen for its strengths in extracting local patterns, LSTM for sequential and temporal dependencies, and CNN-LSTM to see whether the individual strengths translate into a higher-performing model. This section describes the neural network architectures evaluated in the federated learning experiments, including the CNN, LSTM, and hybrid CNN-LSTM models.

7.2.1. **CNN**

Figure 2 displays the CNN architecture used in this work. A one-dimensional CNN (1D CNN) architecture was used since the intrusion detection task is based on sequential network traffic features rather than two-dimensional image data. The model consists of three convolutional layers with 32, 64, and 128 filters, respectively. Each convolutional layer is followed by max-pooling. The extracted features are then flattened and passed through dense and dropout layers before producing the final classification output.

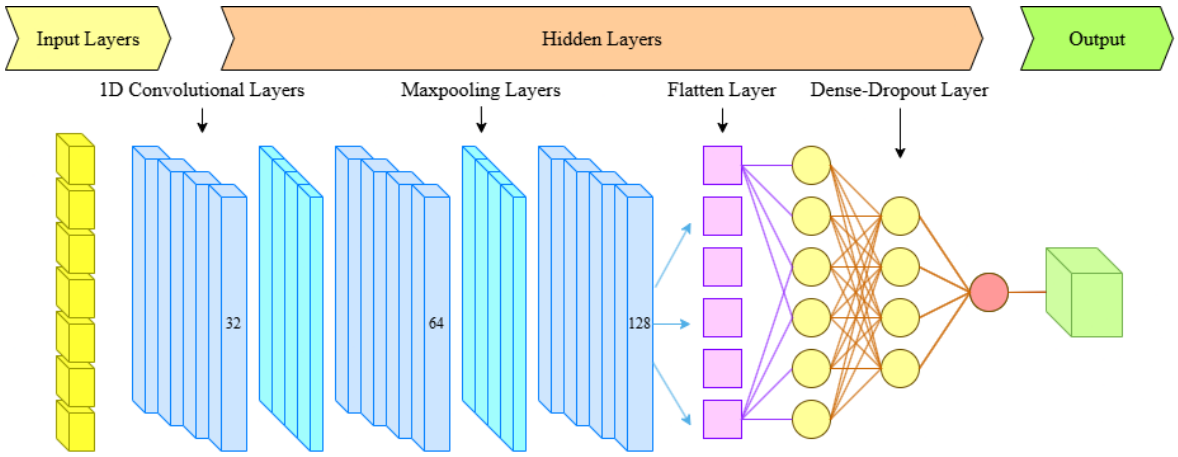

Figure 2: Visual representation of the CNN architecture

The CNN architecture was selected because convolutional networks are effective at extracting local patterns from sequential data such as network traffic flows, making them suitable for intrusion detection tasks.

7.2.2. **LSTM**

The LSTM architecture used in this thesis is illustrated in Figure 3. The model consists of two LSTM layers with 128 and 64 units, respectively, followed by dropout layers and dense layers for regularization and classification. The final dense layer produces binary classification output for intrusion detection.

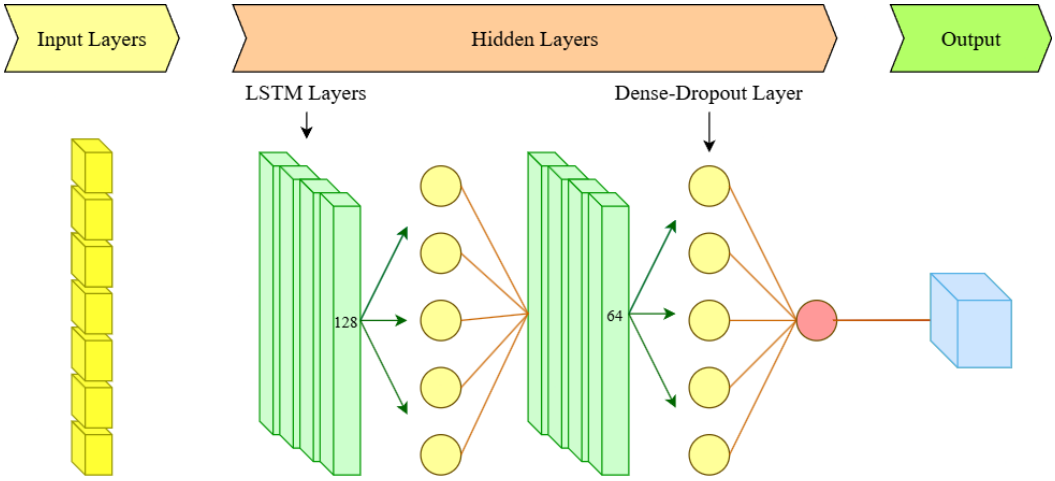

Figure 3: Visual representation of LSTM architecture

LSTM was selected because LSTM networks are designed to capture sequential and temporal dependencies, which makes them suitable for analyzing patterns in network traffic flows.

7.2.3. **CNN-LSTM**

CNN-LSTM is a hybrid of the previous two models, consisting of two CNN layers alongside one LSTM layer. The CNN layers are used for local feature extraction, while LSTM is used to capture sequential dependencies in the network traffic data. The architecture consists of two 1D convolutional layers followed by max-pooling and an LSTM layer. The LSTM layer is followed by a dropout layer and a dense layer. The architecture was included to evaluate whether the individual strengths of CNN and LSTM improve overall performance in a federated learning setup.

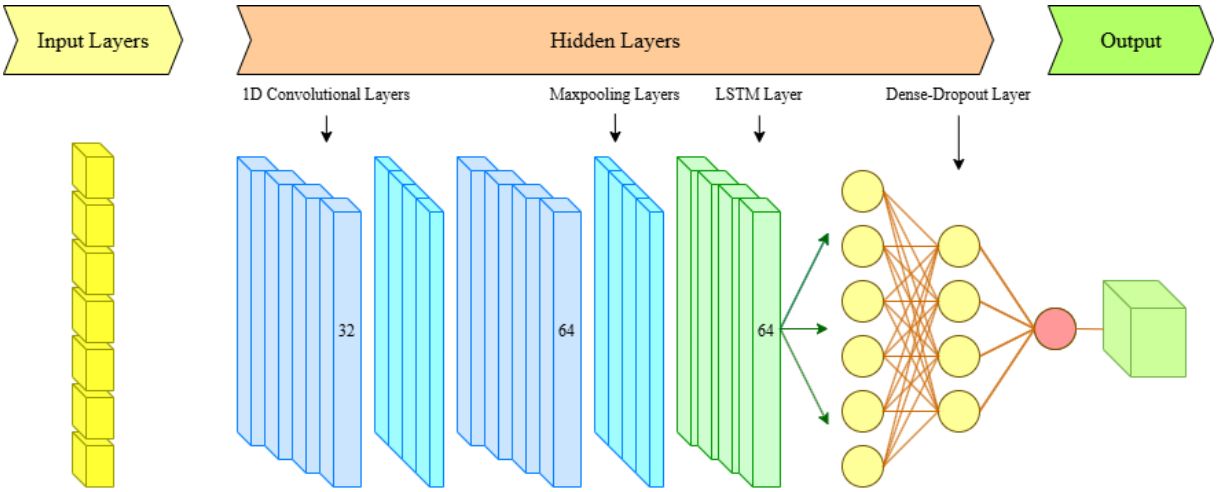

Figure 4: Visual representation of CNN-LSTM architecture

7.2.4. **Parameter counts and model sizes**

Table 1 summarizes the parameter counts and model sizes for the evaluated neural network architectures. These characteristics are important in federated learning, since a larger model entails higher communication costs due to transmitting model parameters between the server and clients during each round. The CNN-LSTM model consists of considerably fewer total parameters than the CNN and LSTM models, which may result in lower communication cost during federated training.

Hyperparameter tuning was systematically performed using Bayesian optimization [29], with previous results informing the selection of the next settings to test.

As a result of hyperparameter tuning to optimize the models, the optimal filter sizes (layers) for CNN-LSTM were smaller than those for either CNN or LSTM, as shown in Table 1. This should not have an impact on the overall results, and it has been considered when evaluating the different models. If the number of parameters were normalized within a certain range, it might affect the accuracy of the model by causing overfitting or underfitting. The models were optimized for accuracy rather than for model size.

	Layer 1	**Layer 2**	**Layer 3**	**Parameters**	**Model Size**
CNN	32	64	128	105025	410.25 KB
LSTM	128	64		118081	461.25 KB
CNN-LSTM	32	64	64	41473	162.00 KB

Table 1: Overview of the size of each model's layers, parameter count, and model size

7.3. Dataset and Preprocessing

In this section, the dataset and preprocessing steps will be explained.

7.3.1. Dataset description

The Edge-IIoTset [22] is a dataset designed for intrusion detection systems, which was generated from a realistic industrial IoT testbed with multiple layers. The data consists of different IoT sensory devices communicating with each other. The dataset contains multiple attack categories, such as DoS/DDoS, information gathering, man-in-the-middle, injection, and malware, as well as normal traffic, making it well suited for this thesis.

7.3.2. Feature preprocessing

During feature preprocessing, several features were removed. The reason for removing them was that they were deemed irrelevant for intrusion detection, could introduce bias, or could cause overfitting. Some of the features that were removed included IP addresses, timestamps, ports, and payload information, among others. The remaining non-numeric values were hashed deterministically, allowing them to be used for training.

7.3.3. Data splitting

For training, each client reserves 20% of the local dataset for testing, and the remaining 80% is divided into training and validation subsets. This results in an overall split ratio of 64:16:20, where 64% was used for training, 16% for validation, and 20% for testing. The split ratio is used to provide both enough data for local training and to preserve separate datasets for validation and evaluation.

To emulate heterogeneous local data distributions, different parts of the Edge-IIoT dataset were distributed amongst the federated learning clients. Each client was assigned a different combination of files containing attack and normal traffic, resulting in variation between the local datasets used during training.

The virtualized environment’s available memory and hardware resources constrained the amount of data partitioned to each client.

7.3.4. Scaling

Feature normalization was performed using StandardScaler. The scaler was fitted after the data split to avoid data leakage between training and evaluation datasets.

7.3.5. **Label handling**

Binary label encoding was applied to represent normal and attack traffic as separate classes for binary classification. The encoded labels were used together with the binary cross-entropy loss function during model training.

7.4. **Federated Training Configuration**

The federated learning experiments were conducted using 1, 3, 5, 7, and 10 communication rounds. Most models we tested started degrading after ten rounds, which caused us to deem it an appropriate round number to end the tests on. A batch size of 64 and three local epochs were used for all experiments. The Adam optimizer with a learning rate of 0,001 was selected based on preliminary testing.

For local validation, the client side reserved 20% of the data for testing. Model validation on the server side was performed using a small, randomized subset of the DNN-EdgeIIoT-dataset.csv file to evaluate centralized accuracy and loss. The data from DNN-EdgeIIoT-dataset.csv was unseen, making it appropriate for validating the accuracy and loss. Each test produced a log file containing relevant performance metrics and data.

7.5. **Experimental Scenarios**

This section describes the experimental scenarios and network conditions evaluated in the federated learning experiments.

7.5.1. **Packet loss**

Packet loss was implemented using tc-netem (Traffic Control Network Emulator), a Linux module widely used in networking research for its availability, maturity, and high accuracy [30]. Packet loss levels of 1%, 2%, 3%, 5%, and 10% were evaluated during the experiments of this work.

There are many factors affecting packet loss in a wireless network, such as interference, signal strength, and protocol [31] [32]. Therefore, a single static value does not suffice. In Wi-Fi networks, packet loss of ~1-5% is common, and it can exceed 10% in high-interference scenarios.

The 1-3% packet loss range reflects realistic network conditions in wireless networks, and the higher tests of 5% and 10% are included to evaluate federated learning under degraded network conditions caused, for example, by congestion, interference, or poor signal coverage.

7.5.2. **Experimental setup**

The experiment was performed by conducting 1, 3, 5, 7, and 10 rounds of training for each model architecture. The experiments were repeated across multiple packet loss configurations to evaluate how degraded network conditions influenced federated learning performance, communication cost, and training efficiency for the evaluated model architectures.

7.5.3. **Definitions and performance metrics**

There are many communication measurements that could be used and a handful were picked because of their importance to this thesis. Communication overhead is defined as the additional bandwidth required to perform a task. Network statistics were collected using iptables to monitor the transmitted and received traffic between the federated learning server and clients during the experiments. Communication cost is the total amount of bytes sent and received, including retransmissions. Communication efficiency is calculated as the static Flower cost divided by the communication cost per round. Convergence is the point at which the model reaches near-final performance and no longer shows a consistent improvement trend in subsequent training rounds. The following are the formulas used for evaluating communication performance:

$$Communication\ Cost\ per\ Round = \frac{Total\ Bytes\ Transmitted}{Number\ of\ Rounds}$$

$$Communication\ Efficiency = \frac{Baseline\ Flower\ Cost}{Communication\ Cost\ per\ Round}$$

$$Communication\ Overhead = Communication\ Cost\ per\ Round - Flower\ payload$$

Flower payload is the raw model weights transmitted between the server and the client during federated learning.

Similarly, many metrics exist in ML, and a subset was selected due to their relevance to this study. During each experiment, multiple performance and system-level metrics were collected, including decentralized and centralized accuracy and loss, precision, recall, F1-score, communication cost, training time, model size, overhead, CPU and RAM usage. Other metrics exist, such as logarithmic loss. Logarithmic loss could be used to measure the uncertainty in a model's predictions. The metrics collected are standard and deemed sufficient for the scope of this thesis project. CPU and RAM usage were deemed irrelevant to this thesis but could be included in future studies. Training time data were collected but not used because other, more relevant performance metrics were better suited to investigating the research questions. The classification performance was evaluated using the following metrics:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Where TP, TN, FP, and FN stand for true positives, true negatives, false positives, and false negatives, respectively [7]. Precision shows real positive results across the model's predictions, meaning that high precision indicates a low number of false alarms. Recall shows how many misses the model has made from all positive predictions. F1-score is used as a balancing measurement between precision and recall. Accuracy alone can be misleading in the event of an imbalanced dataset.

Training loss displays how wrong the model's predictions are on training data. Decreasing training loss over rounds means the model is learning. The training loss function used is Binary Cross-Entropy (BCE), which has the following formula:

$$BCE = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

N is the number of observations, y_i is the actual binary label (1 or 0) of the i^{th} observation, p_i is the predicted probability of the i^{th} observation being in class 1 [33].

8. Results

In this section, we present the experimental results using CNN, LSTM, and CNN-LSTM.

8.1. Communication Cost Across Federated Architectures

The parameter count is directly proportional to communication cost. As models increase in size and complexity, communication cost increases proportionally. Different architectures scale at different rates with increasing parameter count. Table 2 shows the number of parameters across the different architectures and how much communication was needed on average per round in an ideal scenario.

Model	Total number of parameters	Communication cost per round (MB)	Model size
CNN	105025	2,639	410,25 KB
LSTM	118081	3,019	461,25 KB
CNN-LSTM	41473	1,163	162,00 KB

Table 2: Overview of each model's parameter count, average communication cost per round, and model size

Our results indicate that increasing model size leads to a respective increase in communication cost. This is caused by a larger number of parameters needing to be transmitted during training. Part of the communication cost seems to stem from a fixed baseline cost, regardless of the model architecture.

In Table 2, there is information on the model's size, communication cost per round, and total number of parameters. There is a correlation where LSTM has the highest cost as the largest model, and CNN-LSTM has the lowest cost as the smallest model. The relationship between the number of parameters and communication cost varies across architectures. This is based on results showing CNN having relatively many parameters and a lower cost than the LSTM model.

Communication cost per round under ideal conditions, in this thesis, the baseline being 0% packet loss, shows that LSTM has the highest communication cost per round and CNN-LSTM has the lowest communication cost per round. These results are in line with Table 2, which shows the effect parameter count has on neural network architectures.

Round 1 consistently produced the lowest communication efficiency across architectures. A reasonable explanation is that the initial TCP three-way handshake contributed proportionally

more to the total communication cost during the first round of training in the federated learning setup.

8.2. Convergence and Detection Performance in Federated Learning

Figure 5 shows the accuracy across federated learning rounds for the evaluated architectures. The CNN-LSTM model reached a stable accuracy level after approximately two rounds and maintained relatively consistent performance throughout the remaining training rounds. The LSTM model converged more gradually, showing a steady increase in accuracy before reaching a relatively stable state around rounds 6–7. In contrast, CNN reached high accuracy in early rounds but appears to fluctuate more than the other architectures. CNN never converges in our experiment. All three models reach high accuracy, but with very different trajectories to reach it.

CNN results show that the model reached high detection accuracy early in training and kept high performance up to 10 rounds, even with minor fluctuations as seen in Figure 5. The convergence behavior differs between the three architectures. These results suggest model architectures influencing both accuracy and convergence dynamics during federated learning.

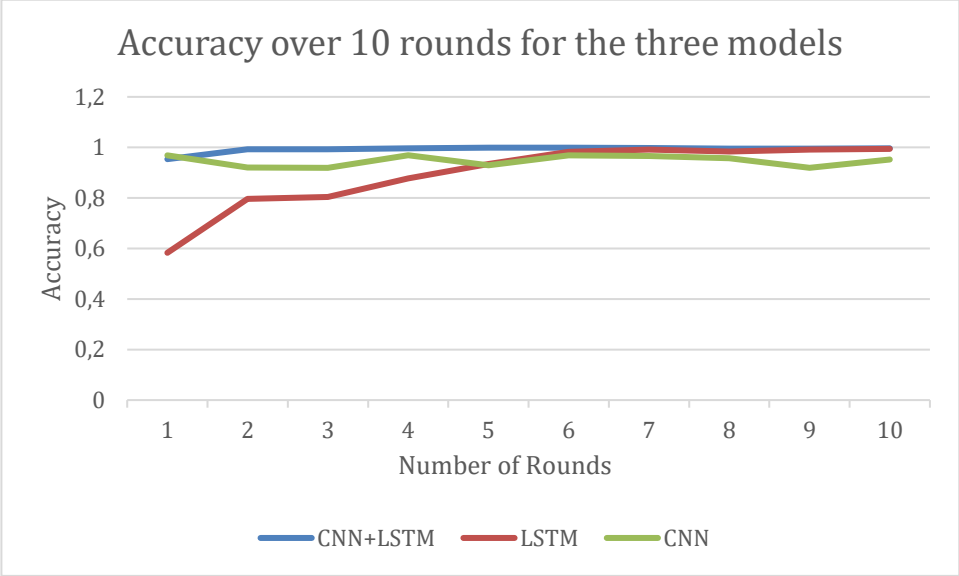

Figure 5. Detection accuracy across federated learning rounds for the evaluated CNN, LSTM, and CNN-LSTM architectures

Table 3 summarizes the performance of the three neural network architectures across various metrics. The values are collected from runs after 10 rounds, and while LSTM reached perfect performance in classification, it also produced the highest communication costs. CNN-LSTM exhibited near-perfect performance while having a much lower total communication cost, leading to a strong balance between communication and accuracy.

Measurements	CNN	LSTM	CNN-LSTM
Precision	0,983933	1	0,9999
Recall	0,9826	1	0,9999
Accuracy	0,982633	1	0,9999
F1-Score	0,9828	1	1
Communication Cost (MB)	26,302	29,9455	11,522

Table 3: Summary of five measurements, Precision, Recall, Accuracy, F1-score, and Communication Cost after 10 rounds

Figure 6 shows the training loss over 10 federated learning rounds for the three architectures. LSTM exhibits a consistent decline in training loss, achieving the lowest final training loss of the three evaluated architectures despite starting at the highest point. CNN and CNN-LSTM do not exhibit the same consistency; in the case of CNN, this is consistent with CNN's inconsistent accuracy trajectory. Training loss for CNN declines steadily from round 5 to round 10. Since training loss indicates whether the model is learning, we can see that the LSTM model learns through the last round. CNN and CNN-LSTM exhibit unstable training loss, which could indicate a learning rate that is too high for those models.

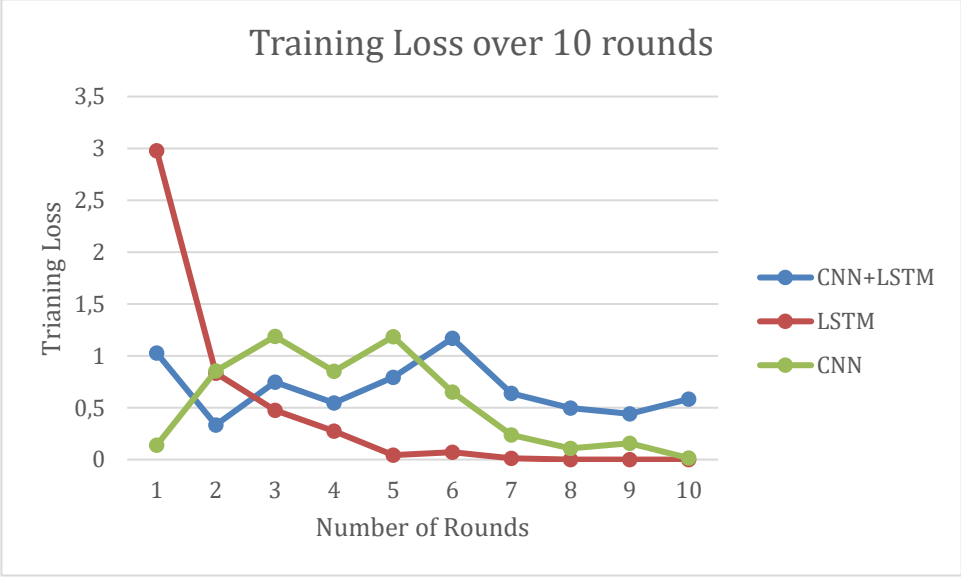

Figure 6: Training loss over 10 rounds for CNN-LSTM, LSTM, and CNN

8.3. **Communication-Accuracy Trade-Off Analysis**

To determine the communication-accuracy trade-off across different architectures, we use accuracy per MB as the metric. This metric quantifies communication efficiency across the different architectures. Table 4 presents the accuracy per MB across federated learning rounds for the architectures. It shows diminishing returns across rounds. CNN receives its highest score of 0,667 in round 1 and drops to 0,0694 in round 10. LSTM stays the most consistent across all rounds but is the least efficient overall. Overall, CNN-LSTM is the most efficient, as it had the highest accuracy per MB across 4 of 5 rounds. The effects of diminishing returns mean that the first round is the most efficient in terms of accuracy per MB. This is not the case for overhead, which is the least efficient in the first round of training. One of the main reasons for this is TCP, which establishes the three-way handshake in the first round.

Rounds:	**1**	**3**	**5**	**7**	**10**
CNN	0,667	0,2072	0,1385	0,0924	0,0694
LSTM	0,1491	0,0947	0,0488	0,0466	0,0327
CNN-LSTM	0,8152	0,1905	0,1724	0,1236	0,0868

Table 4: Accuracy per MB

Figure 7 illustrates the relationship between cumulative communication cost and detection accuracy for the architectures under the baseline network conditions. Each point in Figure 7 represents the achieved accuracy and cumulative communication cost after 1, 3, 5, 7, and 10 rounds of federated learning.

CNN-LSTM achieved high levels of accuracy with comparatively low communication cost. This aspect places CNN-LSTM in the most favorable region of the communication-accuracy-trade-off space among the evaluated models. LSTM converged more gradually but achieved high accuracy after three rounds, while remaining stable at a relatively high communication cost compared to CNN-LSTM.

CNN has slower accuracy growth and similar communication cost to LSTM. The graph also shows diminishing returns, with accuracy notably high in early rounds (3 rounds for LSTM). The communication cost continues to increase while accuracy has stagnated, yielding marginal improvements at substantial communication cost. As shown in Table 4, the accuracy-per-MB in round 1 highlights differences among architectures. CNN-LSTM achieves twice the accuracy per MB of CNN and more than twice that of LSTM.

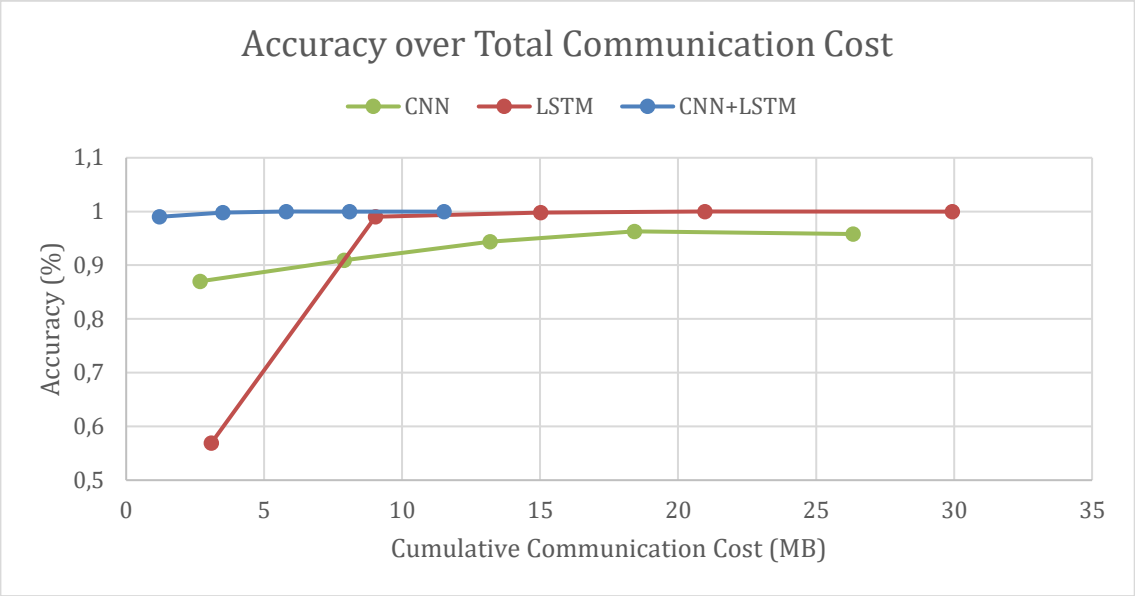

Figure 7: Accuracy over total communication cost in an ideal network scenario where each dot represents rounds 1,3,5,7 and 10 in order

8.4. **Federated Learning under Packet Loss Conditions**

The behavior of communication and accuracy both changed noticeably when packet loss was introduced to the federated learning environment. Packet loss was emulated at ranges of 1% to 10%.

Results show that communication overhead increased with increasing packet loss, but this increase was not proportional to the configured packet loss percentage. The impact of packet loss also varied between the evaluated architectures and between individual experimental runs.

Figure 8 illustrates the communication-accuracy relationship under 10% packet loss conditions. Compared to the baseline scenario in Figure 7, the models required noticeably higher cumulative communication costs to reach the same level of detection accuracy. CNN-LSTM shows the most favorable communication-accuracy trade-off among the evaluated models.

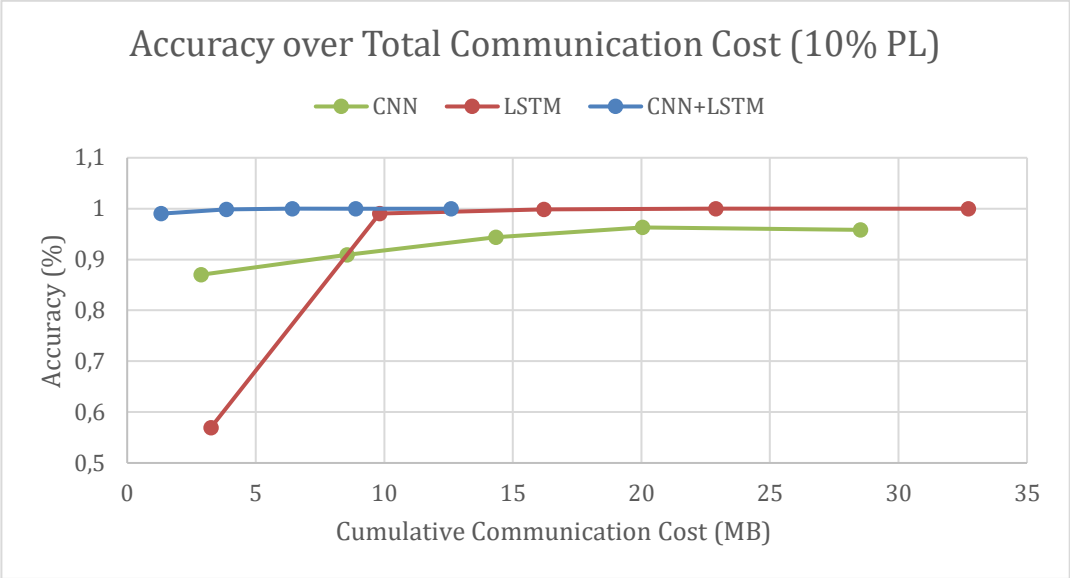

Figure 8: Accuracy over total communication cost in a non-ideal network scenario where each dot represents rounds 1,3,5,7 and 10 in order

Figure 9 illustrates how the total communication cost increases for each architecture as packet loss increases. It shows an ideal scenario without any packet loss and one scenario with 10% packet loss for each model and number of rounds.

Packet loss results in a proportional increase in communication costs. Table 5 demonstrates that the communication costs of 10 rounds of training increase proportionally with packet loss across the different architectures, without affecting detection accuracy. At 10% packet loss, the communication cost increased by approximately 9% across all evaluated models.

One possible explanation is the reliability of TCP. In the event a packet is dropped, the TCP session ensures retransmission. All model updates are preserved, allowing federated learning to continue. The main metric affected is the communication cost. The retransmissions did increase the total transmitted data, resulting in higher communication costs.

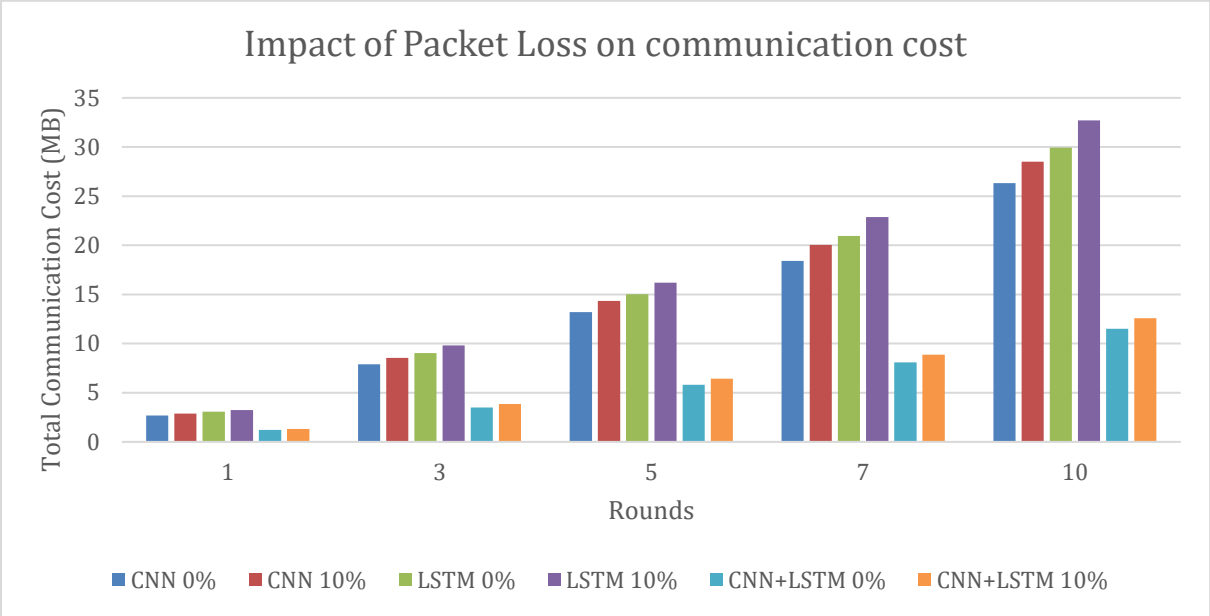

Figure 9: Total communication cost for each model with 0% packet loss and a scenario of 10% packet loss

An important takeaway from Table 5 is that the increase in packet loss results in very similar increases in communication cost, with all three models reaching roughly a 9.3% increase at 10% packet loss. This stays consistent throughout our experiments. The communication costs make sense given the model sizes, and in Table 6, we can see that the growing costs due to packet loss also make sense because losing traffic requires retransmissions, which increase communication costs.

Packet Loss	**0% Baseline**	**1%**	**2%**	**3%**	**5%**	**10%**
CNN Communication Cost per Round	2,62	2,671 (2,0%)	2,682 (2,4%)	2,693 (2,8%)	2,73 (4,6%)	2,854 (9,3%)
LSTM Communication Cost per Round	2,996	3,025 (1,0%)	3,055 (2,0%)	3,092 (3,2%)	3,114 (3,9%)	3,271 (9,2%)
CNN-LSTM Communication Cost per Round	1,152	1,171 (1,6%)	1,176 (2,1%)	1,197 (3,9%)	1,209 (4,9%)	1,259 (9,3%)

Table 5: Communication cost per round in MB and the % increase with rising packet loss

Table 6 shows the per-round communication overhead under increased packet-loss conditions for the evaluated models. These results show that communication overhead increased substantially as network conditions degraded. The LSTM architecture displayed the largest proportional increase in communication overhead per round, followed by the CNN model and then the CNN-LSTM architecture. Given the relative model sizes and communication requirements of evaluated architectures, this behavior is generally consistent.

The CNN model shows some irregularities at 1% to 3% packet loss, and the cause for this could not be determined from the data collected during the experiments.

	Communication Overhead		
Packet loss rate (%)	**CNN**	**LSTM**	**CNN-LSTM**
0% (baseline)	0,2833 MB	0,3114 MB	0,2144 MB
1%	0,2911 MB (2,7%)	0,3385 MB (8,7%)	0,2306 MB (7,6%)
2%	0,2828 MB (-0,2%)	0,3656 MB (17,4%)	0,2442 MB (13,9%)
3%	0,3020 MB (6,6%)	0,3851 MB (23,7%)	0,2516 MB (17,4%)
5%	0,3570 MB (26%)	0,4372 MB (40,4%)	0,2704 MB (26,1%)
10%	0,4598 MB (62,3%)	0,5550 MB (78,2%)	0,3284 MB (53,2%)

Table 6: Per-Round Communication Overhead under Different Packet Loss Levels. Values are presented as MB followed by the relative percentage increase compared to the baseline condition in parentheses.

9. Discussion

The Results from 8.1 revealed that a larger number of parameters led to higher communication costs. The communication cost/model size scaled linearly with a greater number of parameters, consistent across the three evaluated architectures. Chapter 8.1 also revealed that the first round is the least efficient round. The explanation is that, as the TCP three-way handshake is established, it contributes a proportionally larger share of the communication cost over fewer rounds.

8.2 provided insight into how the different models converged, an interesting takeaway being that CNN-LSTM converged fast and CNN struggled to converge due to fluctuations in performance. A possible reason for CNN-LSTM behaving that way may be the combination of LSTM layers stabilizing the model and CNN layers. Resulting in the hybrid model reaching higher accuracy in fewer rounds compared to the other neural network architectures. This would explain the differences in behavior among the CNN, LSTM, and hybrid CNN-LSTM models. For the communication-accuracy trade-off, the overarching finding was that CNN-LSTM was the most efficient model in terms of accuracy per MB transmitted. The data showed that CNN was more efficient than LSTM early in training, but LSTM achieved better performance at the maximum number of rounds.

Experiments involving degraded network conditions with packet loss resulted in a steady increase in communication cost, but it did not impact the model's evaluation performance. Because of TCP, degraded network conditions should not result in a lower quality evaluation performance since any dropped packets would be retransmitted. An interesting point is the effect of packet loss on CNN, as shown in Table 6. At 10% packet loss, the CNN model experienced 62,3% overhead, while at 1-3% packet loss, the overhead ranged from -0,2% to 6,6%. This was vastly different from how LSTM and CNN-LSTM reacted to packet loss in the experiments. We could not determine the cause of this, but we can only speculate that it is due to a lack of additional experimental data, which could be obtained with more time. This thesis set out to investigate three research questions.

RQ1 was answered with results displayed in 8.1, and the effects of packet loss on communication costs and overhead were answered in 8.4. The question set out to answer how communication costs evolve. Based on the data presented, there is no indication that it evolves outside of the first round, having a higher communication cost than the other rounds. Beyond that, the communication cost remains steady and increases linearly across the different architectures.

For RQ2, it was found that CNN-LSTM converged the fastest and maintained high accuracy, whereas LSTM took longer to converge but was very stable during convergence. CNN may have converged but was unstable at high accuracy, as it fluctuated more than the other ML

architectures. Training loss results indicate that the LSTM learns consistently across rounds, whereas the CNN and CNN-LSTM do not show the same level of learning in this setup. This means LSTM is the strongest at learning.

RQ3 investigated the trade-offs, and the most important findings from it are that CNN-LSTM has the highest overall performance and is consistently more efficient than the LSTM and CNN architectures. Between LSTM and CNN, LSTM proved to be more consistent and more efficient at the end of training, with CNN being more efficient in the beginning.

Among the neural network architectures investigated in this thesis, the hybrid CNN-LSTM model consistently outperformed CNN and LSTM on the evaluated metrics. Results collected in this work indicate the hybrid model as being the most efficient, being least affected by worsened network conditions, and having one of the best training times. Despite being the smallest model in terms of size and parameters, it still demonstrated high, stable accuracy and fast convergence.

The generalizability of the findings may be limited by the computational power of the evaluated experimental setup and by the extent to which the Edge-IIoT dataset was used. Different datasets will yield different results, and scaling the experiment to more clients would also result in new findings.

10. Conclusions

In this thesis, we have investigated communication-accuracy trade-offs in federated learning for network intrusion detection systems. We have compared CNN, LSTM, and the hybrid architecture CNN-LSTM. The results showed that communication costs scale with model size and packet loss, while the evaluated architectures exhibited substantially different convergence behaviors and communication characteristics.

The first round is the least efficient due to the initial TCP handshake. We also found that CNN-LSTM converges faster than LSTM and CNN after only two rounds. CNN was inconsistent and never truly converged, while LSTM had a slow start but converged after 6-7 rounds. Results indicate that CNN-LSTM was the most efficient neural network architecture overall. CNN was efficient in early rounds, and LSTM in later rounds. Packet loss increased communication costs by ~9,3% at 10% loss, but it did not affect accuracy due to TCP retransmissions. CNN-LSTM used the smallest model, had the highest, most stable accuracy, and the lowest communication cost. LSTM achieved the lowest training loss, improving its performance each consecutive round, while the other two models showed inconsistent performance. This implies that our LSTM model was the strongest at learning out of the three neural network architectures.

The results demonstrated that packet loss mostly affected communication overhead rather than final detection accuracy for each federated setup, considering TCP communication. To conclude, the findings in this report suggest that communication cost, convergence, and network stability are important considerations when designing and servicing a federated learning system for intrusion detection.

11. Future work

For future studies, it is possible to expand on this thesis by analyzing client hardware performance. That could provide some interesting insights, as in the study by Albanbay et al. [6]. Such experiments could provide new insight regarding how client heterogeneity influences communication overhead, convergence behavior, and overall training efficiency.

Further ways to expand this work are outlined in [6]. One idea is to investigate the scalability of the FL setup by increasing the number of clients to observe how communication overhead evolves in larger environments.

Another way to expand this work is to consider architectures beyond those evaluated in this thesis. Different numbers of parameters, configurations, and network setups, along with greater packet loss, would also provide insight into communication-accuracy trade-offs in federated learning.

As seen, there are ways to expand upon this thesis, including using a different dataset, as in the work of Dos Santos et al. [5], which used MAWIFlow, a large dataset of real traffic collected over a year. Using a different dataset can yield different results, and using more data from a dataset can also generate interesting new results.

References

[1] R. Bayindir, I. Colak, G. Fulli, and K. Demirtas, "Smart grid technologies and applications," *Renewable and Sustainable Energy Reviews*, vol. 66, pp. 499–516, 2016.

[2] W. Wang and Z. Lu, "Cyber security in the smart grid: Survey and challenges," *Computer Networks*, vol. 57, no. 5, pp. 1344–1371, 2013.

[3] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, *Federated Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 13, no. 3. Morgan & Claypool Publishers, 2019, pp. 1–207, doi: 10.2200/S00960ED2V01Y201910AIM043.

[4] D. Xiao, X. Fan, L. Chen, M. Li, and M. Zhang, "Communication-privacy-accuracy trade-offs in federated learning for non-IID data with shuffle model," *Knowledge-Based Systems*, vol. 324, p. 113872, 2025, doi: 10.1016/j.knosys.2025.113872.

[5] R. R. dos Santos, E. K. Viegas, A. O. Santin, and P. Tedeschi, "Federated learning for reliable model updates in network-based intrusion detection," *Computers & Security*, vol. 133, p. 103413, 2023, doi: 10.1016/j.cose.2023.103413.

[6] N. Albanbay, Y. Tursynbek, K. Graffi, R. Uskenbayeva, Z. Kalpeyeva, Z. Abilkaiyr, and Y. Ayapov, "Federated Learning-Based Intrusion Detection in IoT Networks: Performance Evaluation and Data Scaling Study," *J. Sens. Actuator Netw.*, vol. 14, p. 78, 2025, doi: 10.3390/jsan14040078.

[7] J. Murel and E. Kavlakoglu, "What is feature engineering?," *IBM Think*, [Online]. Available: https://www.ibm.com/think/topics/feature-engineering. [Accessed: May 21, 2026].

[8] S. Sah, "Machine learning: A review of learning types," *Preprints*, 2020, doi: 10.20944/preprints202007.0230.v1.

[9] D. Bergmann, "What is Deep Learning?," IBM Think. [Online]. Available: https://www.ibm.com/think/topics/deep-learning . [Accessed: May 25, 2026].

[10] M. S. Habeeb and T. R. Babu, "Network intrusion detection system: A survey on artificial intelligence-based techniques," *Expert Systems*, vol. 39, no. 9, 2022, doi: 10.1111/exsy.13066.

[11] K. He, D. D. Kim, and M. R. Asghar, "Adversarial machine learning for network intrusion detection systems: A comprehensive survey," *IEEE Communications Surveys and Tutorials*, vol. 25, no. 1, pp. 538–566, 2023, doi: 10.1109/COMST.2022.3233793.

[12] M. Wang, N. Yang, D. H. Gunasinghe, and N. Weng, "On the robustness of ML-based network intrusion detection systems: An adversarial and distribution shift perspective," *Computers*, vol. 12, no. 10, p. 209, 2023, doi: 10.3390/computers12100209.

[13] J. Sahoo, M. Ouaissa, and A. K. Nair, *Federated Learning: Principles, Paradigms, and Applications*. Palm Bay, FL: Apple Academic Press, 2025.

[14] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A Friendly Federated Learning Research Framework," *arXiv:2007.14390*, 2020. [Online]. Available: https://doi.org/10.48550/arxiv.2007.14390

[15] H. Zhang, J. Ye, W. Huang, X. Liu, and J. Gu, "Survey of federated learning in intrusion detection," *Journal of Parallel and Distributed Computing*, vol. 195, Art. no. 104976, 2025, doi: 10.1016/j.jpdc.2024.104976.

[16] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," *arXiv*, submitted Feb. 2016, revised 2023. [Online]. Available: https://arxiv.org/abs/1602.05629.

[17] J. Xu, B. Guo, Y. Shen, and F. Chen, "A two-layer asynchronous federated learning for heterogeneous IoT devices," *Future Generation Computer Systems*, vol. 178, Art. no. 108255, 2026, doi: 10.1016/j.future.2025.108255.

[18] M. Asad, S. Shaukat, D. Hu, Z. Wang, E. Javanmardi, J. Nakazato, and M. Tsukada, "Limitations and future aspects of communication costs in federated learning: A survey," *Sensors*, vol. 23, no. 17, p. 7358, 2023, doi: 10.3390/s23177358.

[19] N. Arimanda, R. Vaishnavi Radhakrishnan, and P. U., "FL-IDS++: A dynamic federated learning framework for intrusion detection with personalized non-IID data, adversarial resilience and energy-efficient lightweight models," *Future Generation Computer Systems*, vol. 177, Art. no. 108234, 2026, doi: 10.1016/j.future.2025.108234.

[20] H. Vardhan, X. Yu, T. Rosing, and A. Mazumdar, "Client selection in federated learning with data heterogeneity and network latencies," *arXiv*, 2025.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[22] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv*, 2015. [Online]. Available: https://arxiv.org/abs/1511.08458.

[23] IBM, "Convolutional Neural Networks," *IBM Think*, [Online]. Available: https://www.ibm.com/think/topics/convolutional-neural-networks [Accessed: May 19, 2026].

[24] J. Noble, "What is long short-term memory (LSTM)?," *IBM Think*, [Online]. Available: https://www.ibm.com/think/topics/lstm. [Accessed: May 21, 2026].

[25] S. S. Bamber, A. V. R. Katkuri, S. Sharma, and M. Angurala, "A hybrid CNN-LSTM approach for intelligent cyber intrusion detection system," *Computers & Security*, vol. 148, Art. no. 104146, 2025, doi: 10.1016/j.cose.2024.104146.

[26] T. Al Nuaimi, S. Al Zaabi, M. Alyilieli, M. AlMaskari, S. Alblooshi, F. Alhabsi, M. F. B. Yusof, and A. Al Badawi, "A comparative evaluation of intrusion detection systems on the edge-IIoT-2022 dataset," *Intelligent Systems with Applications*, vol. 20, Art. no. 200298, 2023, doi: 10.1016/j.iswa.2023.200298.

[27] I. Belcic, C.Stryker, "What is Feature Selection?," IBM Think. [Online]. Available: https://www.ibm.com/think/topics/feature-selection . [Accessed: May 25, 2026].

[28] IBM, "Datasets," *IBM Think*, [Online]. Available: https://www.ibm.com/think/topics/dataset [Accessed: May 20, 2026].

[29] F. Archetti and A. Candelieri, *Bayesian Optimization and Data Science*, 1st ed. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-030-24494-1.

[30] S. Becker, T. Pfandzelter, N. Japke, D. Bermbach, and O. Kao, "Network Emulation in Large-Scale Virtual Edge Testbeds: A Note of Caution and the Way Forward," arXiv:2208.05862, Aug. 2022. [Online]. Available: https://doi.org/10.48550/arxiv.2208.05862

[31] J. Zheng, Y. Li, L. Chen, F. Wang, B. Gu, Q. Sun, X. Gao, and F. Zhou, "Effects of packet loss on neural decoding effectiveness in wireless transmission," *Brain Sciences*, vol. 15, no. 3, p. 221, Mar. 2025. doi: 10.3390/brainsci15030221.

[32] Minnovation, "Acceptable packet losses for IoT networks," minnovation.com.au. [Online]. Available: https://minnovation.com.au/quality-assurance/acceptable-packet-losses-for-iot-networks/ [Accessed: Jun. 17, 2026].

[33] GeeksforGeeks, "Binary cross entropy / log loss for binary classification," geeksforgeeks.org. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/binary-cross-entropy-log-loss-for-binary-classification/ [Accessed: Jun. 17, 2026].