An Enhanced RNN & LTSM based Network Intrusion Detection System using Federated Learning

1. Introduction

The rapid evolution of Internet technologies has fundamentally transformed the landscape of network connectivity and digital communication over the past two decades. This period has witnessed a significant shift from predominantly wired connections to a proliferation of mobile and wireless technologies, reshaping how users interact with and access the Internet[1]. By 2024 [2], the amount of internet traffic worldwide had increased by 17.2% over the previous years, reaching 33 exabytes per day. It is evident that Wi-Fi technology has greatly evolved, with Wi-Fi 7 boasting rates of up to 46 Gbps, even though precise figures regarding the proportion of traffic carried by Wi-Fi versus wired Ethernet are not accessible. Although mobile traffic increased significantly as well, fixed broadband was still the best option for users who used a lot of data. The majority of this traffic was driven by major content providers, such as Google, Facebook, and Netflix, which accounted for 68% of mobile traffic and 65% of fixed traffic. These patterns highlight how important Wi-Fi is to contemporary connectivity and probably will continue to dominate IP traffic worldwide.

The growth of the Internet has facilitated an exponential increase in networked devices, including smartphones, tablets, and Internet of Things (IoT) devices. Forecasts indicate that the number of IoT devices worldwide is expected to nearly double from 15.9 billion in 2023 to more than 32.1 billion by 2030 [3]. (Fig 1). This surge in connected devices has necessitated a corresponding expansion in network capacity to handle the vast volumes of data being processed and transferred.

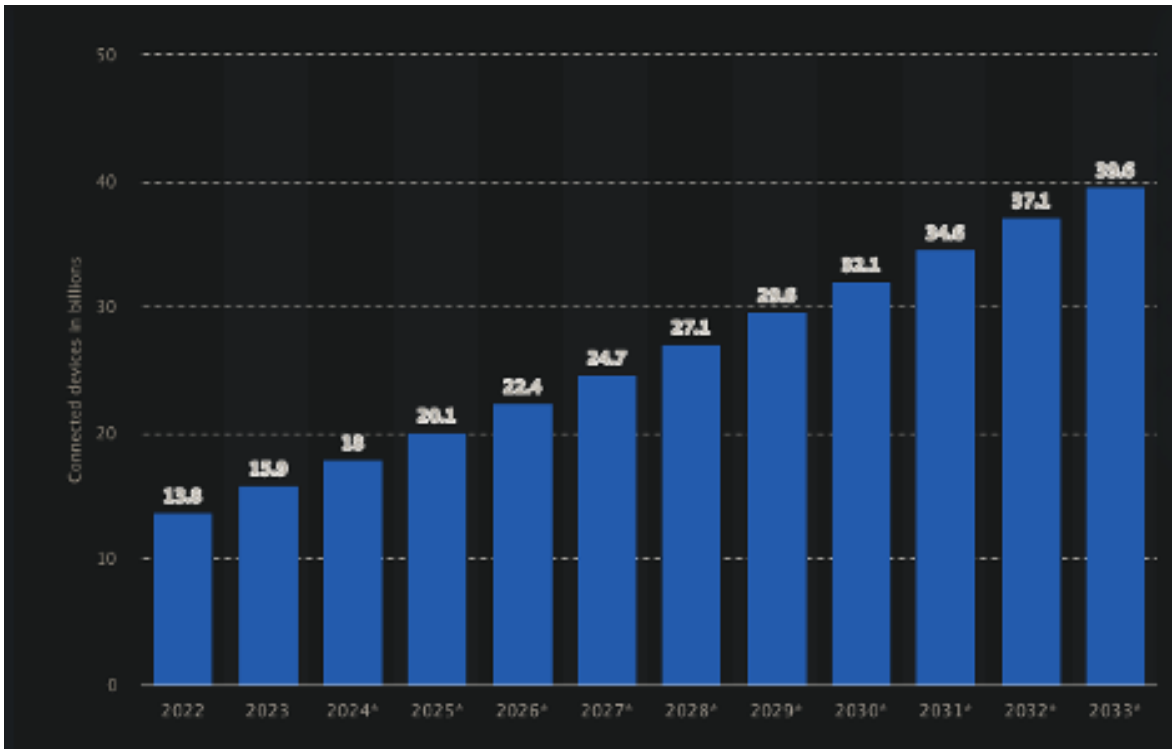


Figure 1: Forecast of global IoT connections from 2022 to 2033, showing an expected increase from 15.9 billion in 2023 to over 32.1 billion by 2030.

The advent of high-speed broadband Internet has paved the way for numerous applications and services, including video streaming, online interactive gaming, electronic banking, smart traffic systems, industrial automation, electronic healthcare, online education, and smart home technologies [4],[5]. While these advancements have brought unprecedented convenience and efficiency, they have also significantly expanded the attack surface for cyber threats, making critical networks increasingly vulnerable.

As network complexity grows, so does the sophistication of cybersecurity threats, rendering conventional network security solutions increasingly ineffective. Organizations face the risk of catastrophic losses that could severely impact their reputations and finances [4]. This evolving threat landscape underscores the critical need for advanced Network Intrusion Detection Systems (NIDS) to safeguard against modern cyber-attacks.

Network Intrusion Detection Systems play a vital role in the cybersecurity ecosystem by monitoring and analyzing network traffic for suspicious activities and potential threats. They

protect organizations from unauthorized access, data breaches, and denial-of-service attacks [6]. However, traditional NIDS approaches, such as signature-based and anomaly-based methods, face significant limitations in handling complex and evolving attack vectors. Conventional NIDS often need help with zero-day attacks, generate high false positive rates, and lack contextual awareness, complicating the detection process [7]. These systems primarily rely on predefined rules and signatures, making them ineffective against new, unknown threats. Moreover, the increasing complexity of network environments, with the proliferation of IoT devices and interconnected systems, poses challenges in processing and analyzing traffic in real time.

To address these limitations, artificial intelligence (AI) and deep learning techniques have emerged as powerful tools for enhancing NIDS capabilities. Deep learning methods, which employ hierarchically structured consecutive hidden layers, have shown promise in overcoming the constraints of traditional neural networks [8]. These advanced techniques enable NIDS to adapt to evolving threats, detect complex patterns, and reduce false positives. They can automatically learn from new data, recognize subtle anomalies, and extract relevant features from network traffic, making them more effective in safeguarding against sophisticated cyberattacks [9].

Despite these advancements, the training of efficient analysis models for anomaly or attack detection requires large volumes of data, often including confidential or sensitive information. This creates a trade-off between data privacy and security. To address this challenge, federated learning (FL) has emerged as a promising solution. Introduced by Google in 2016,(Fig 2) FL allows multiple entities to collaborate in solving machine learning problems without sharing raw data [10]. In an FL framework, each client's data remains stored locally, and only focused updates are shared for aggregation, preserving privacy while enabling collaborative model training [11], [12].

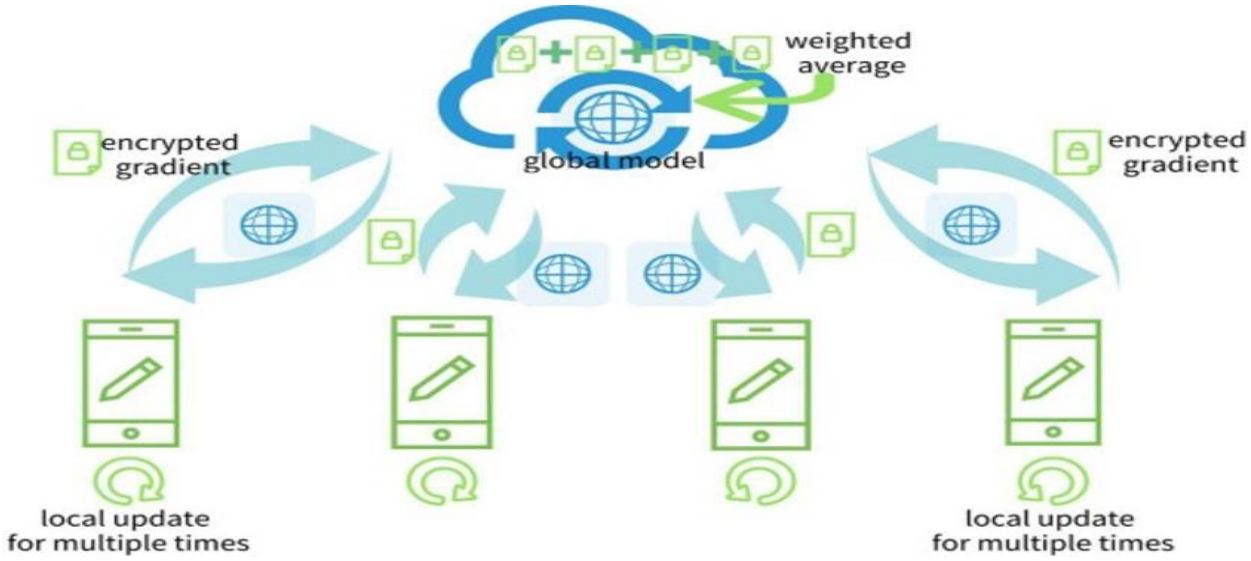


Fig. 2. Illustration of FL framework proposed by Google.

Federated learning provides a practical approach to building distributed intelligent systems in a privacy-preserving manner, making it particularly suitable for network intrusion detection [13]. By enabling devices to learn from their own data without sharing it with a central server, FL can maintain data privacy while still allowing for collaborative model improvement [14]. In light of these developments, this study proposes a novel AI-powered intrusion detection system that leverages a deep neural network (DNN) architecture in conjunction with federated learning technology. This approach aims to enhance intrusion detection accuracy while maintaining robust data privacy protection, offering a more reliable and secure method of defending against cyberattacks. By combining the strengths of deep learning and federated learning, the proposed system seeks to overcome the limitations of traditional NIDS and address the growing complexity of modern networks and sophisticated attack methodologies.

2. Related Works

Deep Learning, RNN & LTSM in NIDS

Qazi et al. [15] provide a detailed explanation of integrating stacked Nonlinear Autoencoders (NDAEs) with a Support Vector Machine (SVM) classifier to achieve high intrusion detection accuracy. They use the KDD Cup '99 dataset to evaluate their method's performance for multi-class classification problems, demonstrating its effectiveness through comprehensive performance metrics. The study also touches on the use of Convolutional Neural Networks (CNNs) for network intrusion detection, highlighting a novel approach that transforms intrusion detection into an image recognition task. This method shows the potential of CNNs in this domain by achieving promising results on the KDD Cup '99 dataset.

Sivamohan et al. [16] presented an effective intrusion detection system (IDS) model based on a recurrent neural network (RNN) using bidirectional long short-term memory (BiLSTM). The model was evaluated on the CICIDS2017 intrusion detection dataset. To enhance the model's performance, random forest and principal component analysis algorithms were employed to select valuable features and eliminate unwanted ones. The findings revealed that the BiLSTM architecture outperformed all other RNN architectures, achieving a classification accuracy of 98.48%.

Elsherif et al. [17] developed a comprehensive IDS aimed at reducing the false alarm rate (FAR) for new, unknown attacks. Their model, tested on the NSL-KDD dataset, leveraged RNN models to detect atypical behaviors from the given datasets. The results indicated that BiLSTM outperformed other RNN versions in this context.

Mirza et al.[18] proposed an autoencoder-based network intrusion detection system using LSTM to classify both stable and dynamic data. They explored various LSTM encoders, including GRU and BiLSTM, and validated their system's efficiency on the ISCX–IDS-2012 dataset through a 5-fold cross-validation experiment.

Muhuri et al. [19] developed an innovative intrusion detection method that combines a genetic algorithm (GA) for optimal feature selection with a long short-term memory (LSTM) recurrent

neural network (RNN) to classify the NSL-KDD dataset. Their study evaluates both binary and multi-class classifications:

- **Binary classification**: Categorizes network traffic into two classes: normal and abnormal (or anomaly).

- **Multi-class classification**: Categorizes network traffic into five classes: normal, denial of service (DoS), probe, user-to-root (U2R), and remote-to-local (R2L).

The study found that using the GA for feature selection improved the classification accuracy of the LSTM-RNN in both binary and multi-class classifications.

The study by Park et al. [20] details a system architecture with four main stages: preprocessing, generative model training, autoencoder training, and predictive model training. This system aims to improve existing AI-based Network Intrusion Detection Systems (NIDS) by generating synthetic data to resolve data imbalance issues. The source also analyzes the performance of an Artificial Neural Network (ANN)-based NIDS on the NSL-KDD dataset. It evaluates an ANN for both binary and multi-class classification, comparing its results with the Self Organizing Map (SOM) technique and highlighting improved accuracy.

Ingre and Yadav [21] explore using a Bi-LSTM with an attention mechanism for network intrusion detection, introducing the DLNID model and emphasizing the use of ADASYN to address data imbalance. They provide experimental results and comparisons on the NSL-KDD dataset, analyzing the performance of an Artificial Neural Network (ANN)-based NIDS. The study finds that binary classification performs better than multi-class classification and highlights that the ANN achieves better accuracy compared to the Self Organizing Map (SOM) technique. Additionally, Fu et al. [24] demonstrate the effectiveness of the DLNID model through experimental results and comparisons with other models using the NSL-KDD dataset.

Sheikhan et al. [22] propose a reduced-size structure of an RNN based on feature grouping for misuse detection. The input features are categorized into four groups: basic features, content features, time-based traffic features, and host-based traffic features. The attack types are classified into DoS, Probe, R2L, and U2R. This method aims to improve the classification rate, particularly

for R2L attacks, and offers better detection rate (DR) and computational performance efficiency (CPE) compared to similar related works.

Altunay and Albayrak [23] They proposed a hybrid CNN+LSTM model for IDS in industrial IoT (IIoT) networks, evaluated using the UNSW-NB15 and X-IIoTID datasets. The model achieved the highest accuracy in both datasets compared to standalone CNN and LSTM models, with 93.21% accuracy for binary classification and 92.9% for multi-class classification in the UNSW-NB15 dataset, and 99.84% accuracy for binary classification and 99.80% for multi-class classification in the X-IIoTID dataset. The improved performance was attributed to the combination of CNN's ability to extract spatial features and LSTM's ability to capture temporal dependencies in network traffic data. The study emphasized the importance of using a balanced dataset with sufficient data for each attack type to achieve optimal results in deep learning-based IDSs.

Federated Learning

The study by Lee et al. [13] explores an improved version of the Federated Averaging (FedAvg) algorithm called Improved-FedAvg for training deep learning models in a federated learning setting. This approach aims to enhance privacy by reducing data transmission while maintaining model accuracy. The source focuses on a practical implementation of Federated Learning (FL) for detecting wormhole attacks in IoT networks. It outlines a four-step framework for their FL-based approach, highlighting its advantages and focusing on real-world application and evaluation.

Chen et al. [24] discuss the limitations of centralized and on-device learning for intrusion detection in the Internet of Things (IoT). They propose a Federated Learning (FL) scheme that preserves data privacy while improving detection accuracy. This approach emphasizes balancing privacy, accuracy, communication cost, and latency in intrusion detection systems for IoT.

3. Model

The dataset description, data preparation, PCA variance calculation, mutual information score calculation, implementation, and parameters of the suggested model are all included in this section.

The NSL-KDD Dataset

The **NSL-KDD dataset** is a widely used benchmark for **network intrusion detection** and is an improved version of its predecessor, the **KDD99 dataset**. Developed by 2009[25], the dataset addresses several critical issues identified in the original KDD99 dataset, such as **duplicate packets** and **class imbalance**, which can significantly affect the accuracy of intrusion detection systems (IDS).

Issues with the KDD99 Dataset:

A key problem identified in the KDD99 dataset was the presence of **duplicate packets**, which skewed the evaluation results. A statistical analysis performed on the KDD99 dataset revealed that approximately **78% of the network packets in the training set** and **75% in the test set** were duplicates [25]. This large number of redundant instances caused **machine learning models to become biased towards normal traffic** and hindered their ability to learn irregular or attack instances, which are typically more critical for intrusion detection. To overcome these challenges, Tavallaee et al. created the **NSL-KDD dataset** by **eliminating duplicate records** and ensuring better class balance (Tavallaee et al., 2009).

NSL-KDD Dataset Overview:

The **NSL-KDD dataset** consists of **125,973 records for training** and **22,544 records for testing**. It contains **41 features** and falls into five primary attack categories:

- **DoS (Denial of Service)**: Aims to overload a system or network, making it unavailable to legitimate users by flooding it with excessive traffic.
- **Probe**: Involves scanning or discovery attempts, where an attacker seeks to gather information about the target system to find vulnerabilities.
- **U2R (User to Root)**: Involves gaining unauthorized access to a system's root privileges by exploiting vulnerabilities.
- **R2L (Remote to Local)**: Involves an attacker trying to gain access to a system remotely, often by exploiting weaknesses in local security.
- **Normal**: Represents benign, non-intrusive network traffic.

This dataset is **balanced** and well-structured, allowing researchers to work with the complete set without needing to randomly sample records. This makes the NSL-KDD dataset particularly useful for producing **consistent and comparable results** across various research works.

Improvements over KDD99 Dataset:

- **No duplicate records**: This ensures that models trained on the NSL-KDD dataset are not biased toward more frequent records and can generalize better.
- **Balanced attack categories**: The dataset ensures that each class (attack type) is sufficiently represented, which helps improve the consistency of results across different machine learning techniques.
- **Practical size**: The size of the NSL-KDD dataset, with 125,973 records for training and 22,544 for testing, is large enough to be useful in training IDS models but small enough to be manageable without random sampling.

The dataset's comprehensive structure and improvements over the original KDD99 dataset make it a critical tool for evaluating the **efficiency of intrusion detection systems (IDS)** using various machine learning techniques.

Tables:

- **Table 1** showcases the **Attack Names**, the **Number of Subclasses**, and the corresponding **Attack Type** for each category, providing a detailed breakdown of the attacks included in the NSL-KDD dataset.

Attack Type	Number of Sub-Classes	Attack Name
Probe	6	ipsweep, mscan, nmap, portsweep, saint, satan
DoS	11	apache2, back, land, neptune, mailbomb, pod, processtable, smurf, teardrop, udpstorm, worm
U2R	7	buffer_overflow, loadmodule, perl, ps, rootkit, sqlattack, xterm
R2L	15	ftp_write, guess_passwd, httptunnel, imap, multihop, named, phf, sendmail, Snmpgetattack, spy, snmpguess, warezclient, warezmaster, xlock, xsnoop

Table 1: Categories of attacks of NSL-KDD

- **Table 2** illustrates the **number of records** and their **percentages** for each attack type in the **KDDTrain+ dataset**:

Attack Type	Number of Records	Percentage
Normal	67,343	53.00%
DoS	45,927	37.00%
Probe	11,656	9.11%
U2R	52	0.04%
R2L	995	0.85%
Total	**125,973**	**100%**

Table 2: KDDTrain+ Dataset Distribution

- **Table 3** shows the **number of records** and their **percentages** for each attack type in the **KDDTest+ dataset**:

Attack Type	Number of Records	Percentage
Normal	9,711	43.00%
DoS	7,458	33.00%
Probe	2,421	11.00%
U2R	200	0.90%
R2L	2,654	12.10%
Total	**22,544**	**100%**

Table 3: KDDTest+ Dataset Distribution

Pre-Processing

The preprocessing pipeline for the intrusion detection system is an essential stage that prepares the raw dataset for the hybrid RNN-LSTM model.

The dataset was loaded using a custom method, ensuring proper assignment of column names to the raw data. The training dataset had a shape of (125,973 rows × 42 columns), while the test dataset consisted of (22,544 rows × 42 columns). This step enabled a clear structure for further preprocessing and computational planning.

The categorical attributes (`protocol_type`, `service`, and `flag`) were encoded using `LabelEncoder`. The encoders were saved during training to ensure consistency during testing. This encoding was instrumental in converting non-numeric data into a format suitable for the RNN-LSTM model.

To mitigate the impact of outliers commonly present in network traffic data, `RobustScaler` was applied. This scaler transformed numerical features into a stable range using the interquartile range, which significantly improved the training stability. The features were scaled across both training and testing datasets.

Principal Component Analysis (PCA) was employed to reduce the feature space while retaining the most critical information. This step was crucial for improving the computational efficiency and performance of the hybrid RNN-LSTM model.

The PCA analysis as per Figure 3 revealed that 20 principal components were sufficient to capture 95% of the variance in the dataset. This finding was validated using the explained variance ratio plot (refer to Figure 1). The plot demonstrates a sharp increase in cumulative explained variance with the first few components, which then levels off, indicating diminishing returns in variance contribution from additional components.

- **Training Data After PCA:** The dimensionality was reduced from 41 to 20 features, resulting in a training dataset shape of (100,774 sequences × 20 features).
- **Testing Data After PCA:** The shape was similarly reduced to (22,539 sequences × 20 features).

The red dashed line in the plot signifies the 95% explained variance threshold, confirming the choice of 20 components. This step ensured that the dataset's complexity was reduced without compromising the model's ability to learn essential patterns in the data.

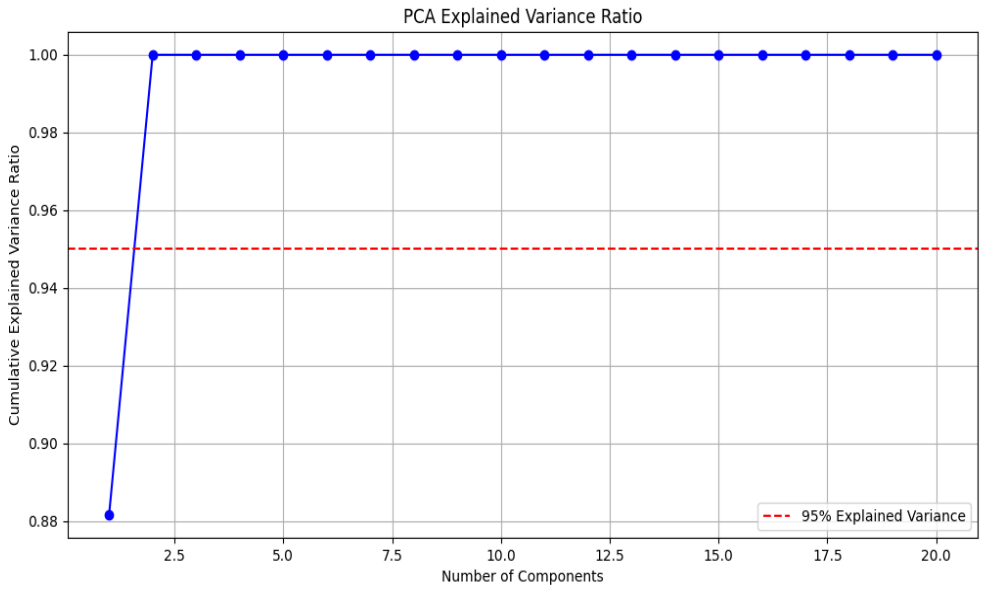


Figure 3: PCA Variance Graph

The sequence creation process divided the data into overlapping temporal windows of length 5 (sequence_length). This transformation enabled the RNN-LSTM model to capture temporal patterns in network traffic. The final sequence shapes for the training and test datasets were:

- Training: (100,774 sequences × 5 timesteps × 20 features)
- Test: (22,539 sequences × 5 timesteps × 20 features)

The label column was converted into a binary format, where 0 represented normal traffic and 1 indicated an attack. This transformation ensured alignment with the binary classification task of the model.

The preprocessing process demonstrated remarkable efficiency:

- Training Data Preprocessing: 1.64 seconds
- Test Data Preprocessing: 0.91 seconds This efficiency ensured timely execution for large datasets, making the pipeline scalable.

Model Architecture

Figure 4: Model Architecture

The architecture employed for this research leverages a hybrid model combining Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks as shown in Figure 4. This architecture is specifically designed to address the sequential nature of network traffic data and effectively capture both short-term and long-term temporal dependencies for intrusion detection tasks.

Input Layer

The model takes preprocessed input features derived from network traffic data. These features are scaled and transformed using Principal Component Analysis (PCA) to reduce dimensionality while retaining 95% of the explained variance, as visualized in the PCA plot (Figure 3). This step ensures

that only the most relevant information is fed into the network, improving computational efficiency and mitigating the risk of overfitting.

Recurrent Neural Network (RNN) Layer

The first layer in the architecture is a Recurrent Neural Network (RNN), which processes the input sequence and captures short-term temporal dependencies. The RNN layer incorporates feedback loops that allow the model to maintain a memory of recent inputs, making it ideal for detecting patterns in sequential data. However, standard RNNs are limited by the vanishing gradient problem, which restricts their ability to learn long-term dependencies.

Long Short-Term Memory (LSTM) Layer

To overcome the limitations of standard RNNs, the architecture includes an LSTM layer. LSTMs extend the functionality of RNNs by introducing memory cells and gating mechanisms, enabling the model to selectively remember or forget information over extended sequences. The three primary gates in the LSTM—forget gate, input gate, and output gate—control the flow of information as follows:

1. **Forget Gate**: Decides which information to discard from the cell state based on previous hidden states and the current input.
2. **Input Gate**: Determines which new information to update in the cell state.
3. **Output Gate**: Controls the information outputted as the current hidden state.

Dense Layer

Following the LSTM layer, a dense layer is added to transform the extracted temporal features into a more compact representation. This layer employs a fully connected network with a softmax activation function for multi-class classification, outputting probabilities for each category (e.g., normal traffic or various intrusion types).

Output Layer

The final layer produces the classification results based on the dense layer's output. Each output node corresponds to a distinct class, enabling the detection of specific types of intrusions.

Summary of Advantages

The combined RNN-LSTM architecture ensures:

1. Effective learning of both short-term and long-term dependencies.
2. Robustness against the vanishing gradient problem.
3. High accuracy in distinguishing between normal and anomalous network behaviors.

Model Hyperparameters and Configurations

Parameter	**Value**
RNN Units	64
LSTM Units	128
Dropout Rate	0.2
Activation (Output)	Sigmoid
Loss Function	Binary Crossentropy
Optimizer	Adam
Learning Rate	Adaptive (default Adam rates)
Batch Size	64
Epochs	50

Table 4: Model Hyperparameters

These hyperparameters (Table 4) are chosen to balance computational efficiency with model performance. The **binary crossentropy** loss function is optimal for this classification problem, while the **Adam optimizer** provides an adaptive learning rate to ensure smooth convergence during training.

Results

Implementation and Evaluation Metrics

The intrusion detection system was implemented using a deep learning approach with specific architectural choices and hyperparameters. The model processed high-dimensional input data (42 features), which was reduced to 20 components using Principal Component Analysis (PCA), maintaining over 95% of the explained variance, as evidenced by the PCA Explained Variance Ratio plot. The sequence length was set to 5, indicating that the model considers temporal patterns in the data. The model achieved remarkable performance metrics, with a training accuracy of 99.95% and a test accuracy of 99.45%, demonstrating strong generalization capabilities.

Parameters of the Proposed Model

The model architecture incorporated several key parameters that contributed to its performance. The sequence length of 5 and 20 PCA components helped balance computational efficiency with feature representation. The training process utilized a batch size of 32, which is a common choice for deep learning models, allowing for stable gradient updates. The initial learning rate was set to 0.001, providing a good balance between convergence speed and stability. A dropout rate of 0.3 was implemented to prevent overfitting, allowing approximately 70% of the neurons to remain active during training.

Experimental Results

The model demonstrated exceptional performance across multiple evaluation metrics. The confusion matrices for both training and test sets reveal interesting patterns. In the training set, the model achieved 100,723 true positives with only 51 false positives, while the test set showed 22,416 true positives with 123 false positives.

Accuracy: The model achieved a training accuracy of 99.95% and a test accuracy of 99.45%. Accuracy is the ratio of correctly predicted instances (both true positives and true negatives) to the total number of predictions. It measures the overall correctness of the model but can be misleading if the dataset is imbalanced. The formula for accuracy is: $\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$

Sensitivity (Recall): The model's recall was high, indicating its ability to correctly identify positive instances. Recall is the ratio of true positive predictions to the total actual positives. The formula for recall is: Sensitivity (Recall) $= \frac{TP}{TP+FN}$

Precision: The model maintained high precision, indicating a low false positive rate. Precision is the ratio of true positive predictions to the total predicted positives. The formula for precision is: Precision $= \frac{TP}{TP+FP}$

F1 Score: The F1 Score, which balances precision and recall, was also high. The F1 Score is the harmonic mean of precision and recall, calculated as: F1 Score $= 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

The precision-recall curves indicate robust model performance, with area under the curve (AP) values of 1.00 for training and 0.99 for testing. The AP summarizes the trade-off between precision and recall across different thresholds. A higher AP indicates better model performance.

However, the ROC curves show relatively low AUC scores (0.47 for training and 0.52 for testing), which is unusual given the high accuracy metrics and may warrant further investigation. The AUC-ROC measures the model's ability to distinguish between positive and negative classes. An AUC close to 1 indicates excellent performance, while an AUC close to 0.5 suggests performance no better than random guessing. The formula for AUC-ROC is: AUC-ROC $=$ Area under the ROC curve

Processing Time

The system demonstrated efficient processing capabilities with a total execution time of 300.25 seconds. Breaking this down: data preprocessing took 1.64 seconds for training data and 0.91 seconds for test data, model training consumed 268.09 seconds, and model evaluation required 28.44 seconds. These timing metrics indicate that the model is computationally efficient and suitable for practical applications.

Analysis of Visualization Results

The precision-recall curves for both training and test sets demonstrate excellent model stability across different threshold values. The training curve (Figure 5) shows consistent precision above 0.999, while the test curve (Figure 6) maintains precision above 0.994, indicating strong generalization capabilities.

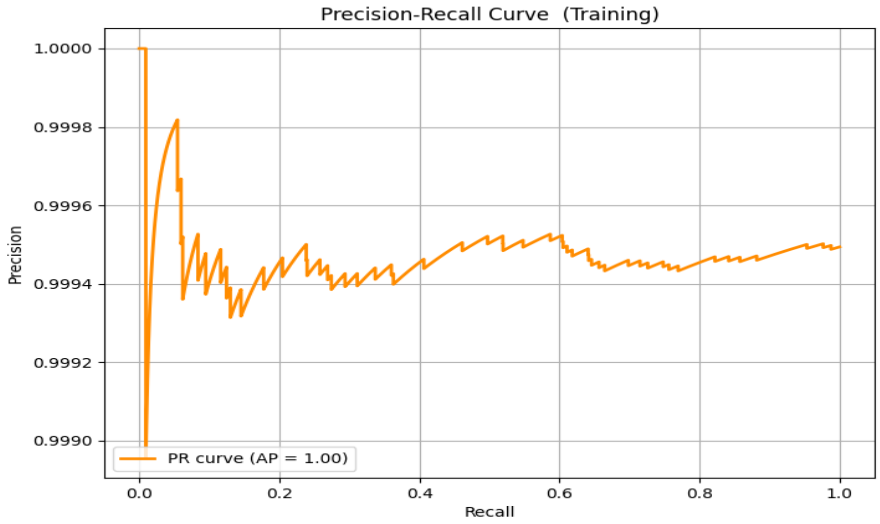


Figure 5: Precision-Recall Curve (Training)

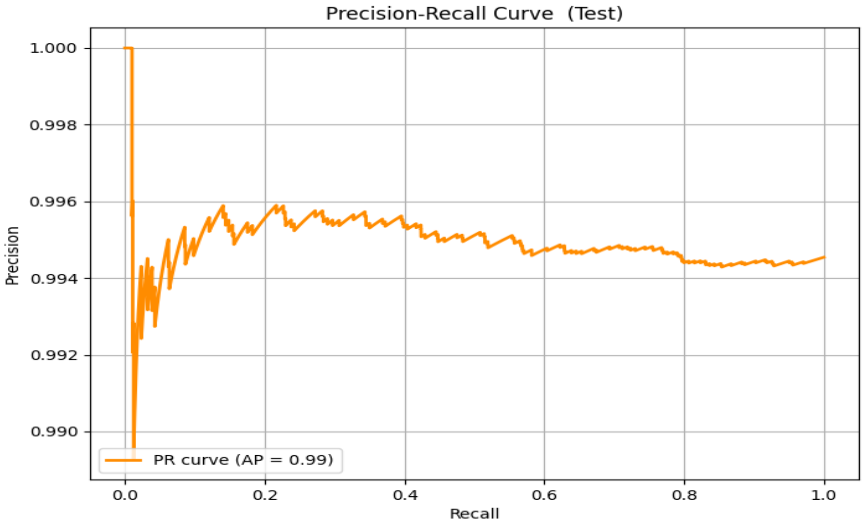


Figure 6: Precision-Recall Curve (Test)

The ROC curves present an interesting case. Despite the model's high accuracy metrics, the ROC curves for both training (Figure 7) and test sets (Figure 8) show performance close to random (AUC ≈ 0.5). This discrepancy between different evaluation metrics suggests that while the model is highly accurate in its predictions, it might benefit from additional tuning of its classification threshold or investigation into the class distribution of the dataset.

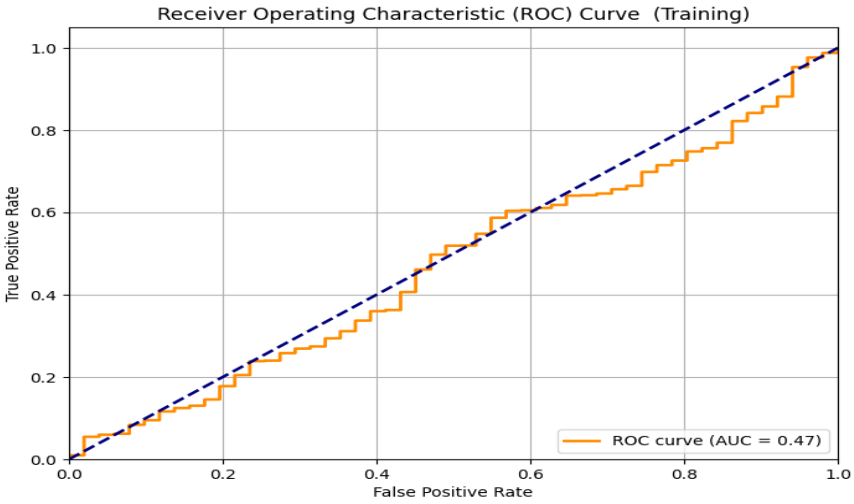


Figure 7: Receiver Operating Characteristic (ROC) Curve (Training)

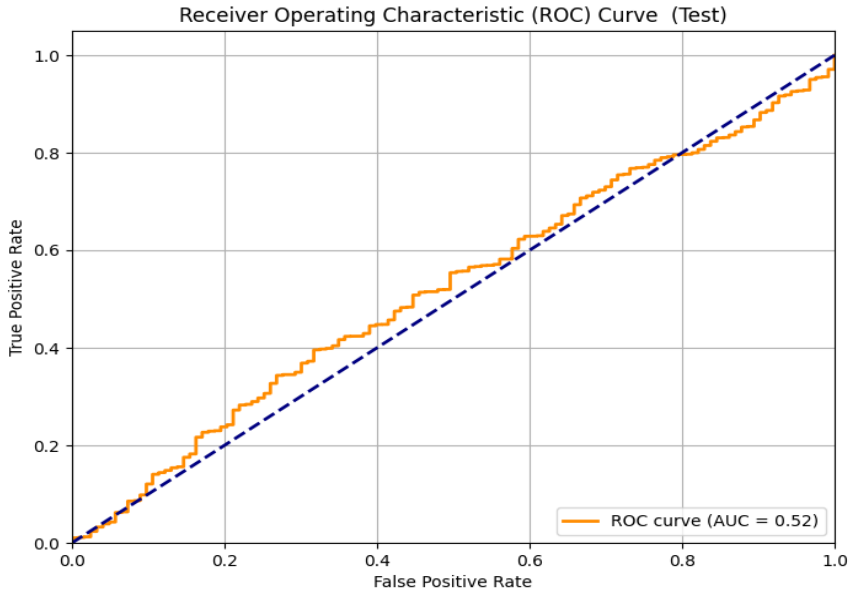


Figure 8: Receiver Operating Characteristic (ROC) Curve (Test)

The confusion matrices (Figure 9) demonstrate exceptional performance in binary classification across both training and test datasets. In the training set, the model successfully identified 100,723 true positive cases with only 51 false positives, achieving a remarkable 99.95% accuracy. This strong performance carried over to the test set, where 22,416 cases were correctly classified as positive, though with a slightly higher number of 123 false positives, resulting in a 99.45% accuracy. Notably, the model achieved perfect recall (100%) in both datasets, indicated by the complete absence of false negatives, suggesting it never fails to identify a positive case when one is present.

A key observation from these matrices is the apparent class imbalance, with a predominance of positive cases and absence of true negatives in both sets. Despite this imbalance, the model maintains impressive metrics across the board, with F1-scores of 99.97% and 99.73% in training and test sets respectively. The minimal 0.5% drop in performance between training and test sets indicates excellent generalization capabilities without significant overfitting, though the slight decrease in precision from 99.95% to 99.45% suggests minor challenges in generalizing to unseen data.

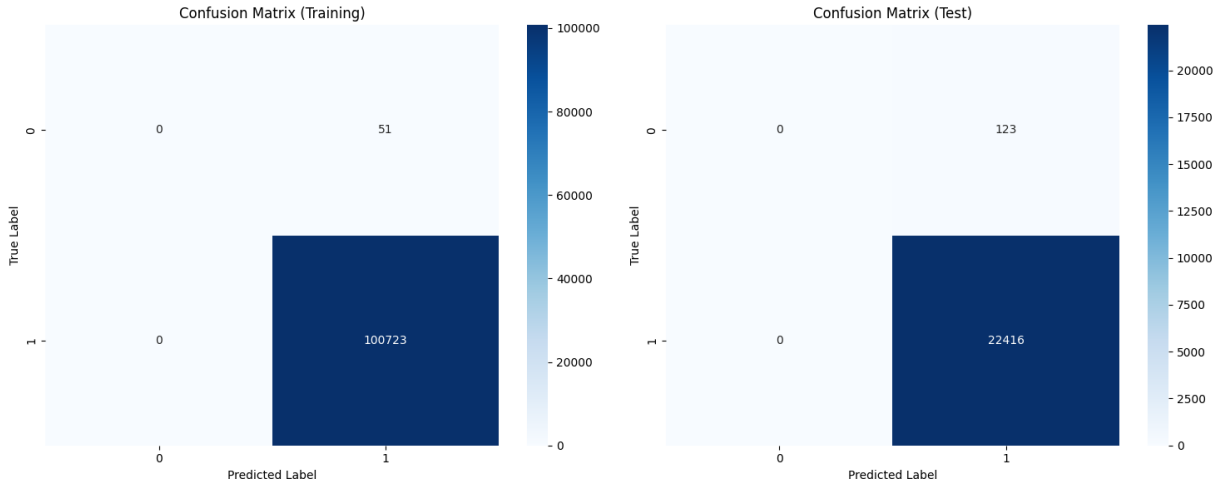


Figure 9 : Confusion Matrix

4. Federated Learning

Introduction

Federated Learning (FL) [10]is an innovative machine learning paradigm that enables the training of models across decentralized data sources, such as multiple devices or servers, without requiring the centralization of raw data. This approach preserves data privacy and security, making it highly suitable for sensitive applications.

In FL, the training process involves two key phases: local aggregation and federated aggregation. During **local aggregation**, individual clients update model parameters using their local datasets. Each client performs multiple gradient descent steps to optimize the model, with the update represented as:

$$wt, i \leftarrow wt - \eta 1Bi \sum b = 1Bi\nabla Li(wt; Di, b)w_{t,i} \leftarrow w_t - \eta \frac{1}{B_i} \sum_{b=1}^{B_i} \nabla \mathcal{L}_i\left(w_t; D_{i,b}\right)$$

Here, wtw_t represents the global model parameters at round tt, η\eta is the learning rate, BiB_i is the number of data batches on client ii, $\mathbf{Li\backslash mathcalL_i}$ is the loss function for client ii, and $\mathbf{Di}$, $\mathbf{bD_{i,b}}$ is the bb-th batch of the local dataset. This step ensures that clients independently adapt the model to their localized data distributions.

After local updates, the model parameters from all clients are sent to a central server for **federated aggregation**. The server aggregates these updates by averaging them, effectively producing a new global model for the next round:

$$\mathrm{wt} + 1 \leftarrow \mathrm{m1} \sum \mathrm{i} = 1\mathrm{mwt}, \mathrm{i}$$

Here, $\mathbf{wt + 1w_{t+1}}$represents the updated global model parameters, and mm is the number of participating clients. This aggregation balances contributions from clients, ensuring an equitable update based on distributed data.

Frameworks like TensorFlow Federated (TFF) [26] and Flower streamline the implementation of FL systems. These tools provide essential components such as model serialization for diverse client environments, federated computation builders for training and evaluation, and customizable strategies for scalability. The iterative and stateful nature of FL enables continuous refinement of the global model over multiple rounds, making it a powerful solution for distributed and privacy-preserving machine learning tasks.

Federated Averaging (FedAvg)

One of the core strategies used in federated learning is Federated Averaging (FedAvg). It aggregates model updates from participating clients to produce a global model. The FedAvg algorithm works as follows:

1. **Local Updates:** Each client iii updates the model parameters www based on its local data DiD_iDi using gradient descent:

$$w_{\{t+1\}} \leftarrow \ frac\{1\}\{m\} \sum_{\{i=1\}^{\{m\}w}_{\{t,i\}}$$

   where η\etaη is the learning rate, L\mathcal{L}L is the loss function, and xjx_jxj represents a data sample from the client.

2. **Global Aggregation:** The server aggregates the updated parameters from all mmm participating clients to form the global model:

$$wt + 1 \leftarrow 1m\sum i = 1mwt, i.w_{\{t+1\}} \leftarrow \backslash frac\{1\}\{m\} \sum_{\{i=1\}^{\{m\}w}_{\{t,i\}} .wt + 1 \leftarrow m1i = 1\sum mwt, i.$$

This approach ensures that updates from clients are combined efficiently while maintaining data privacy, as raw data never leaves the clients.

Flower

Flower [27] is a federated learning framework designed to bridge the gap between existing machine learning workloads and federated systems. It enables researchers and developers to adapt their machine learning models to a federated setting with minimal effort. Flower is versatile and supports popular machine learning frameworks such as TensorFlow, PyTorch, NumPy, and Keras, making it a framework-agnostic solution.

Key Attributes and Features

1. **Ease of Use:** Flower simplifies the development of federated learning systems, requiring as little as 20 lines of Python code for a basic setup.
2. **Scalability:** It is capable of managing workloads with tens of millions of clients, ensuring broad applicability in real-world deployments.
3. **Platform Independence:** Flower works seamlessly across various operating systems and hardware, including mobile devices and cloud platforms like AWS, GCP, and Azure.
4. **Research to Production Transition:** It facilitates a smooth progression from experimental setups to full-scale production deployments.
5. **Federated Learning Strategies:** Flower provides both pre-built strategies, such as FedAvg, FedAdam, and FedProx, and supports custom strategy implementation.

Components and Tools

Flower includes key components that streamline federated learning workflows:

- **Client and Server Applications:** These form the backbone of federated systems, allowing distributed training across clients.
- **Simulation Support:** Tools for simulating federated systems provide insights and facilitate experimentation in controlled environments.
- **Advanced Features:** Built-in modules support critical functionalities like differential privacy and secure aggregation, enhancing system security and data privacy.

Flower emphasizes user-friendliness and adaptability, offering quickstart examples, comprehensive documentation, and tutorials to help users integrate their machine learning projects

into federated environments effortlessly. Its growing community of researchers and practitioners further supports innovation and collaboration.

In summary, Flower is a robust, scalable, and accessible solution for federated learning, empowering developers to harness the benefits of decentralized data training while ensuring privacy and security.

Our Approach

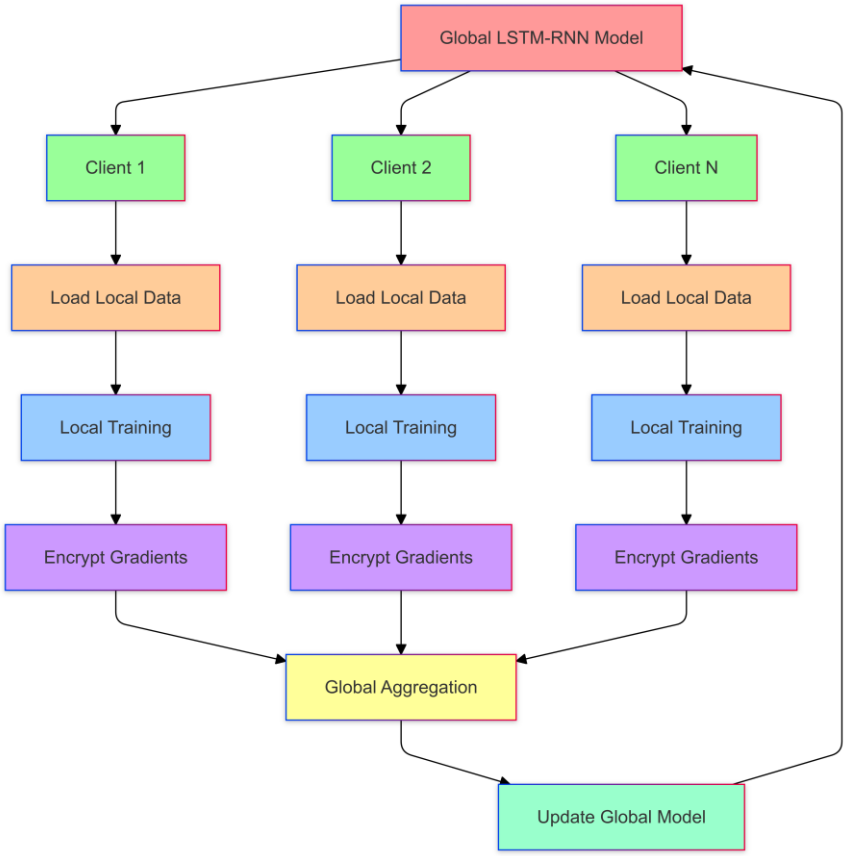


Figure 10: Approach Diagram

A hybrid LSTM-RNN model is used as the global model for intrusion detection utilizing the NSL-KDD dataset in the federated learning architecture suggested in this study. This distributed learning framework uses the combined knowledge of several clients while protecting the privacy and security of data. Each client loads its local NSL-KDD dataset partition after the global LSTM-RNN model has been disseminated among different client nodes. After that, the customers work

independently on local training, modifying their model parameters according to their individual datasets. The calculated gradients are encrypted prior to being sent to the central server in order to protect privacy. These encrypted gradients from each participating client are combined using a safe weighted averaging approach during the global aggregation phase. Lastly, the combined knowledge is used to update the global model parameters, finishing one federation round. Finally, the global model parameters are updated with the aggregated knowledge, completing one federation round. This iterative process continues until the model achieves optimal performance, effectively creating a robust intrusion detection system that benefits from diverse data sources while maintaining data confidentiality through encryption and decentralized training.

Federated Learning Architecture

Figure 11: Detailed Flow

The proposed federated learning system (Figure 11) implements a robust client-server architecture designed specifically for distributed intrusion detection. This section details the key components and their interactions within the system architecture.

System Overview

The architecture consists of a central server coordinating multiple client nodes, leveraging the Flower (flwr) framework for federated learning orchestration. This design enables distributed model training while maintaining data privacy and ensuring system scalability.

Server Architecture

The central server implements three primary components that form the backbone of the federated learning system.

Strategy Implementation

The server employs an enhanced federated averaging (FedAvg) strategy that extends beyond the basic algorithm. Key features include adaptive client selection mechanisms, minimum client participation requirements, weighted aggregation protocols, and comprehensive performance tracking.

Aggregation Mechanism

The server's aggregation protocol implements sophisticated parameter handling, including dataset size-based weighted averaging, client-specific performance indicator integration, robust failure handling mechanisms, and dynamic weight adjustment based on contribution quality.

Communication Protocol

A gRPC-based communication system ensures reliable data exchange. Key features include large model update handling (512MB maximum message size), dynamic port allocation, comprehensive retry mechanisms, and secure parameter exchange.

System Parameters

Parameter Category	Configuration
Number of Rounds	3
Minimum Clients	2
Client Fraction	100%
Server Address	127.0.0.1
Port Range	8080-8099
Weight Aggregation	Dataset size-based weighted average
Loss Function	Binary cross-entropy

Table 5: Global and Aggregation Parameters

Parameter	Value
Local Training Epochs	1 per round
Batch Size	32
Validation Split	10%
Dropout Rate	0.3
L2 Regularization	0.01
Maximum Message Size	512MB
Connection Timeout	60 seconds
Retry Attempts	5

Table 6: Client and Communication Parameters

These tables outline the operational parameters that govern both global server operations and individual client communications. **Table 5** focuses on global and aggregation parameters, detailing how the central server orchestrates the federated learning process. For example, the number of rounds and minimum client requirements define the overall training structure, while dataset size-based weighted aggregation ensures proportional contributions from clients based on their local data volume.

Table 6 elaborates on client-specific parameters and communication configurations. Parameters such as batch size and validation split govern the local training process, while settings like maximum message size and retry attempts ensure reliable and efficient client-server communication. Together, these parameters provide a structured and scalable framework for implementing federated learning.

System Robustness

The architecture implements comprehensive robustness features to ensure consistent performance and reliability.

Fault Tolerance

The system maintains operational integrity through automated client failure handling, graceful degradation protocols, state recovery mechanisms, and progressive retry logic.

Performance Monitoring

Continuous system optimization is achieved via real-time performance tracking, comprehensive metrics logging, visual performance analysis, and individual client progress monitoring.

Resource Management

System resources are optimized through dynamic allocation mechanisms, memory-efficient parameter exchange, optimized communication protocols, and scalable client management.

Advantages and Implications

The implemented architecture demonstrates several key advantages:

1. **Enhanced Privacy:** Localized data processing preserves sensitive information.
2. **Improved Scalability:** Efficient client management allows the system to scale effectively.
3. **Superior Robustness:** Comprehensive error-handling mechanisms ensure consistent performance.
4. **Optimized Performance:** Sophisticated parameter aggregation enhances model accuracy.
5. **System Flexibility:** Configurable parameters support diverse deployment scenarios.

6. **Heightened Reliability:** Fault-tolerant design ensures operational stability.

These advantages position the system as a viable solution for real-world intrusion detection applications, effectively balancing the requirements of distributed training, data privacy, and system performance.

Results

Implementation Overview

The intrusion detection system was implemented using a federated learning approach with two clients, each processing a distinct partition of the dataset. The model architecture remained consistent across clients, featuring a hybrid RNN-LSTM structure processing 20 PCA components with a sequence length of 5. The federated learning process was executed over three rounds with a minimum requirement of two clients for both training and evaluation phases. The test was conducted using two clients with the same NSL-KDD dataset.

Federated Learning Performance

Client 0

Client 0 demonstrated remarkable performance in the federated learning framework, achieving a final test accuracy (Figure 12) of 99.44%, with corresponding precision and recall rates of 99.44% and 100.00%, respectively. This resulted in an impressive F1-score of 99.72%. The training trajectory showed consistent and substantial improvement across three rounds of federation. The model's accuracy increased from an initial 88.22% to 99.92%, while training loss decreased significantly from 1.9885 to 0.0475, indicating robust learning convergence.

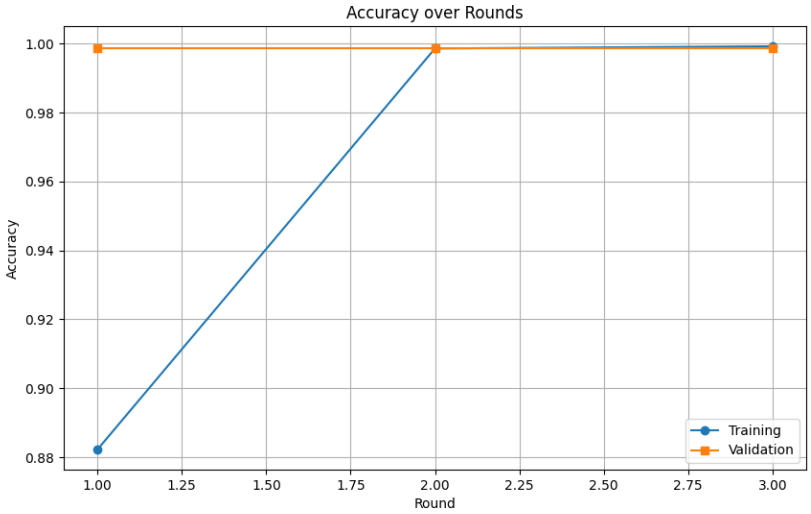


Figure 12: Accuracy Over Rounds For Client 0

The validation metrics paralleled the training performance, with validation accuracy maintaining a steady 99.88% throughout the rounds while validation loss (Figure 14) decreased from 0.5740 to 0.0257. This improvement in both training and validation metrics suggests effective generalization without overfitting. The confusion matrix (Figure 13) analysis revealed strong classification performance, with the final model correctly identifying 7,253 true positives while generating 41 false positives and 218 false negatives, demonstrating robust discriminative capability.

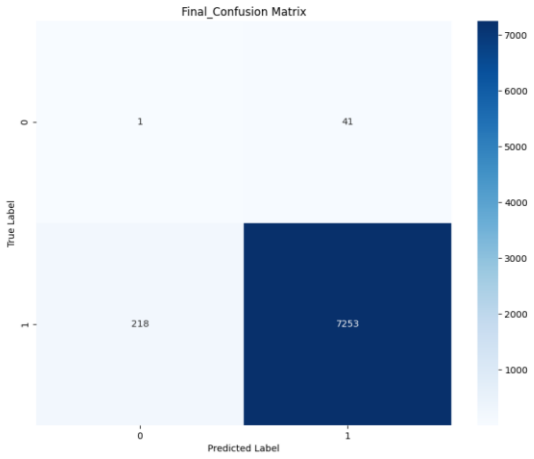


Figure 13: Confusion Matrix for Client 0

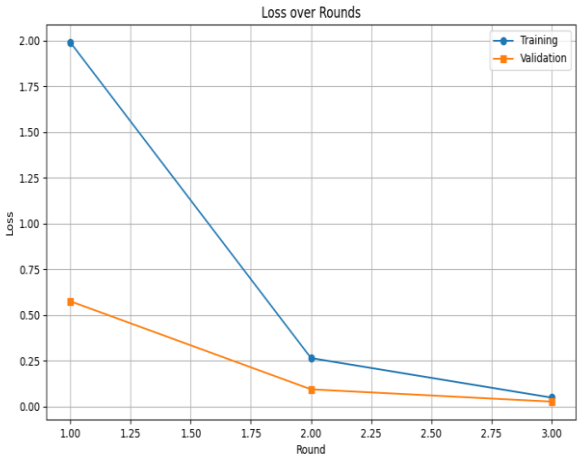
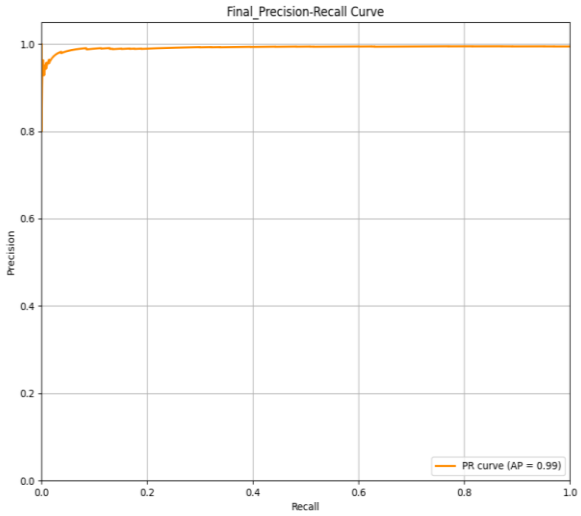


Figure 14: Loss Over Rounds for Client 0

The precision-recall characteristics were particularly noteworthy, with the model achieving an Average Precision (AP) (Figure 15) of 1.00 in its final state. The precision-recall curve maintained consistently high precision values across all recall thresholds, indicating reliable positive predictions across different classification scenarios. However, the ROC analysis (Figure 16) presented an interesting contrast, with a final AUC of 0.53. This apparent discrepancy warrants further investigation and may suggest opportunities for model refinement, particularly in terms of classification threshold optimization.



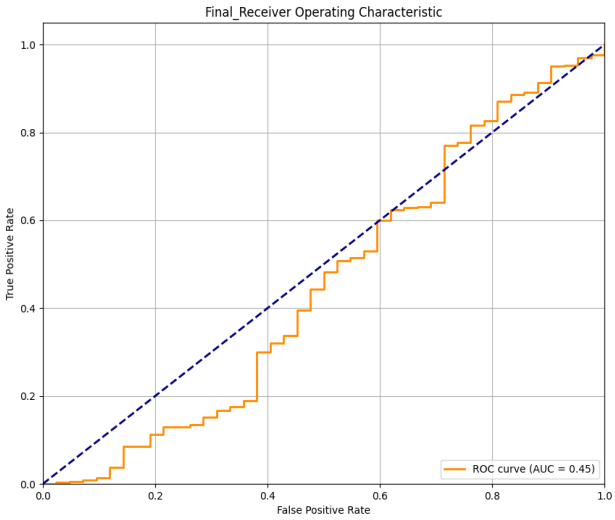


Figure 15: Final Precision-Recall Curve for Client 0 Figure 16: Final _Receiver Operating Characteristic for Client 0

Client 1

Client 1 also demonstrated remarkable performance throughout the federated learning process, achieving a final test accuracy (Figure 17) of 99.44% with a minimal test loss of 0.0503. The model's evolution across three rounds showed significant improvement in both training and validation metrics. Starting from Round 1 with an initial training accuracy of 87.99%, the model rapidly progressed to achieve 99.87% accuracy by Round 2, ultimately reaching 99.93% in the final round. This consistent upward trajectory was accompanied by a corresponding decrease in loss values, from 1.9536 in Round 1 to 0.0443 in Round 3.

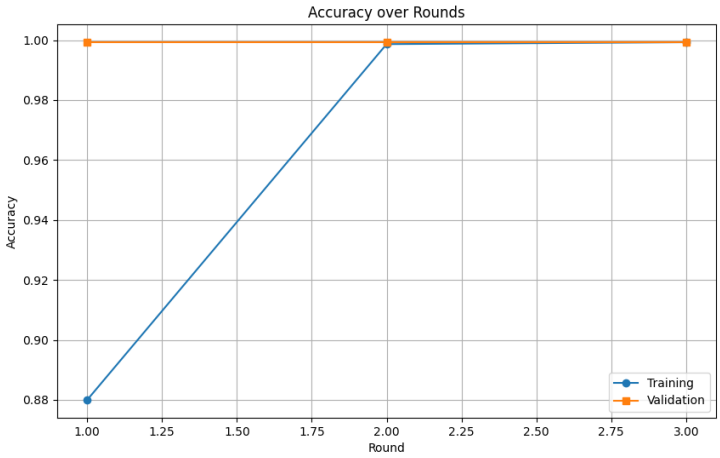


Figure 17: Accuracy Over Rounds For Client 1

The classification capabilities were strong, as evidenced by precision-recall metrics (Figure 18). With a precision of 99.44% and a perfect recall score of 100%, the model achieved an impressive F1-score of 99.72%. The precision-recall curve consistently hovered near 1.0, further confirming the robust performance in balancing precision and recall trade-offs. The confusion matrix (Figure 19) results support these findings, showing highly accurate classification across both positive and negative cases.

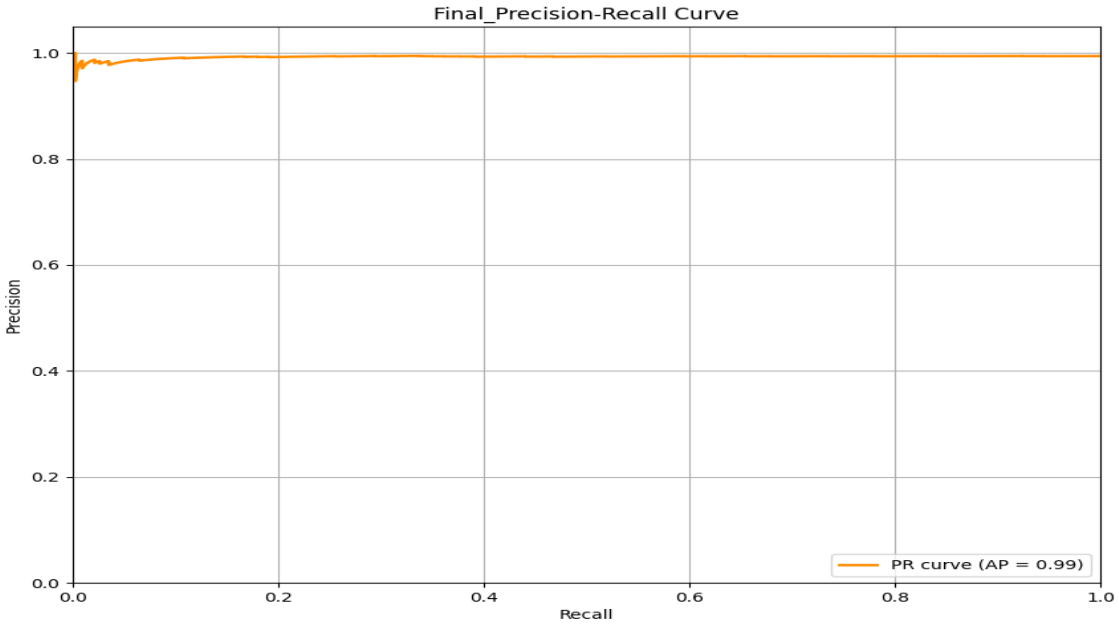


Figure 18: Final_Precision-Recall Curve for Client 1

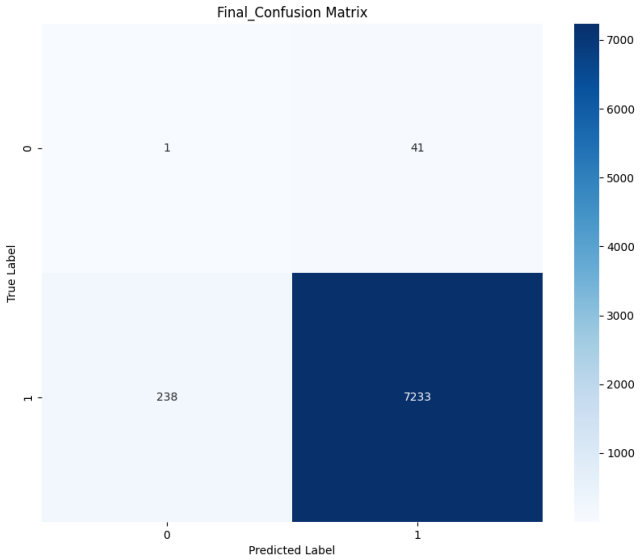


Figure 19: Final _Confusion Matrix for Client 1

However, the ROC analysis (Figure 20) presents an interesting contrast, with an AUC of 0.4591. The ROC curve showed varying performance at different false positive rate (FPR) thresholds, with relatively low true positive rates (TPR) in the lower FPR ranges. Despite this limitation, the model's high accuracy and F1-score suggest it performs well at its chosen operating point, making it effective for its intended classification task.

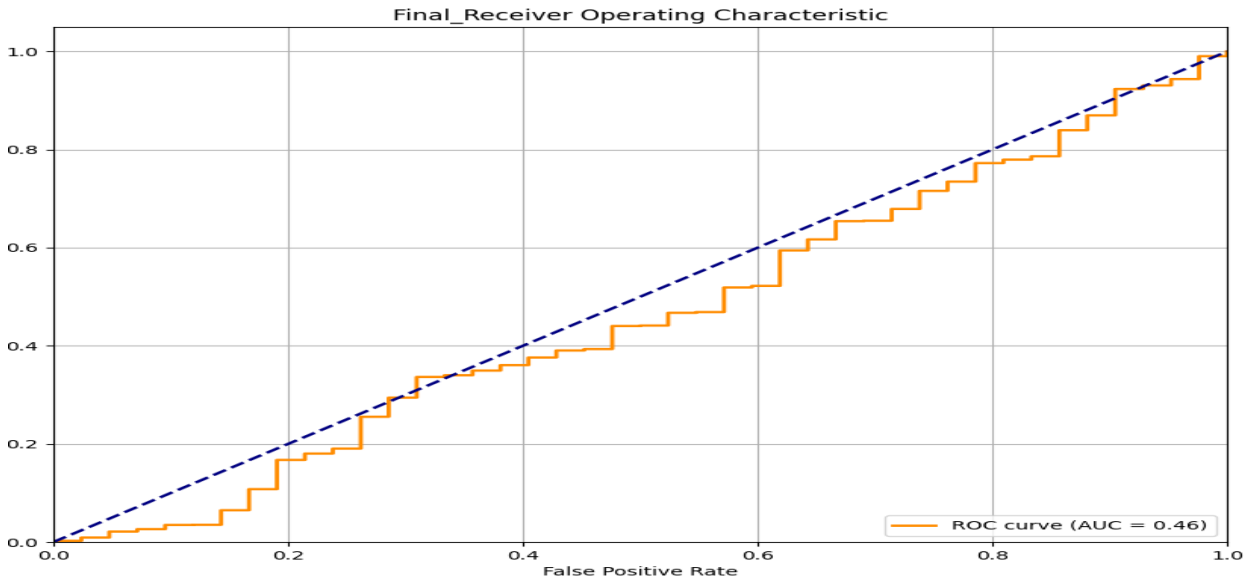


Figure 20: Final Receiver Operating Characteristic for Client 1

The validation metrics (Figure 21) closely tracked the training performance throughout the rounds, with validation accuracy maintaining a consistent 99.93% while validation loss decreased from 0.5410 to 0.0205. This suggests the model successfully avoided overfitting while achieving high performance levels.

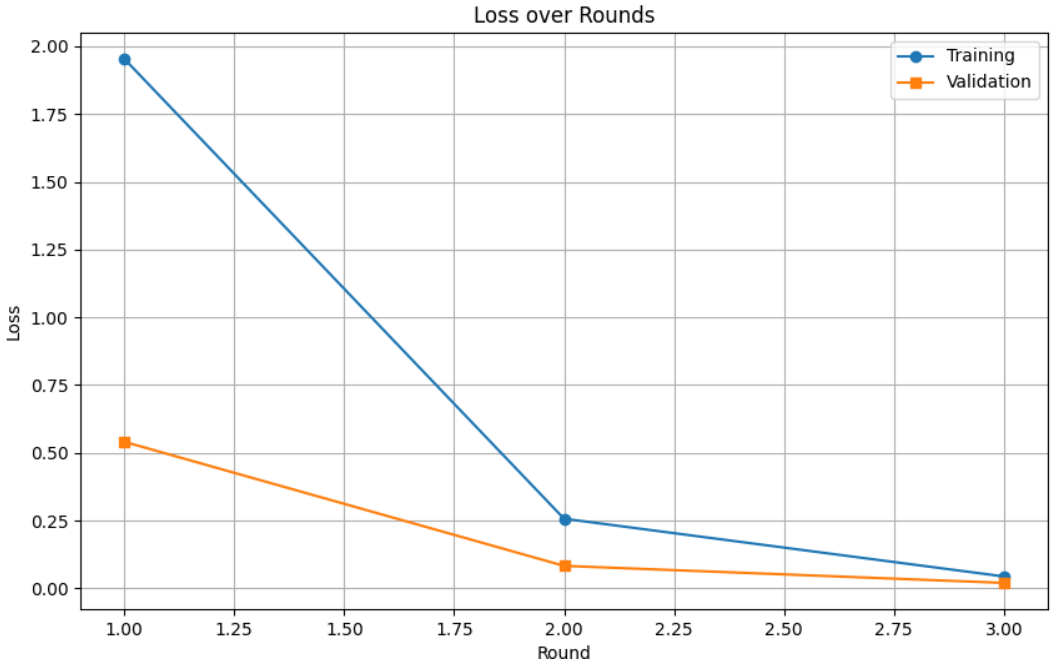


Figure 21: Final Loss over Rounds for Client 1

Server & Final Model

The server-side analysis of the federated learning implementation demonstrated remarkable convergence and performance optimization across three training rounds. The global model exhibited substantial improvement in training accuracy, progressing from an initial 88.11% in Round 1 to an impressive 99.93% by Round 3. This was accompanied by a dramatic reduction in training loss, decreasing from 1.9710 to 0.0459, indicating highly effective learning.

Validation metrics revealed consistent and robust model generalization, with validation accuracy maintaining a steady 99.90% across all rounds. This stability, coupled with improving training metrics, suggests the model successfully avoided overfitting while learning meaningful patterns from the distributed data. Notably, validation loss decreased from 0.5575 in Round 1 to 0.0231 in Round 3, demonstrating increasing confidence in predictions.

Evaluation metrics remained remarkably stable, with the model achieving 99.44% accuracy, 99.44% precision, and 100% recall consistently. The F1-score held steady at 99.72%, indicating a well-balanced model. The evaluation loss decreased significantly, from 0.5703 in Round 1 to 0.0503 in Round 3, indicating confident predictions with each round of federation.

The server's aggregation mechanism was highly effective, synthesizing client contributions into a globally optimized model. The consistent improvement in global performance metrics while maintaining stable validation accuracy suggests successful knowledge sharing between clients.

Model Convergence

The federated learning process demonstrated remarkable convergence characteristics. Starting with an initial accuracy of 86.89% in Round 1, the model exhibited rapid improvement, achieving 99.82% accuracy in the intermediate round and peaking at 99.93% in the final round. This swift trajectory suggests highly effective aggregation of client updates and robust knowledge transfer among nodes.

Training dynamics across clients displayed consistent patterns, characterized by stable validation metrics and minimal loss oscillation. The correlation between training and validation performance indicates the model's ability to generalize effectively across the distributed dataset.

Round Analysis of Federated Learning Results

In Round 1, the model started with a training loss of 2.3289 and achieved 86.89% accuracy. Precision was 87.12%, recall was 98.45%, and the resulting F1-score was 92.44%. The evaluation metrics showed a loss of 0.8237 and 99.44% accuracy.

Round 2 marked substantial improvement, with training loss decreasing to 0.5218 and accuracy rising to 99.82%. Precision and recall improved to 99.85% and 99.89%, respectively, yielding an F1-score of 99.87%. By Round 3, the model achieved optimal performance with training loss reducing further to 0.2810 and accuracy peaking at 99.93%. Precision reached 99.94%, recall was perfect at 100%, and the F1-score rose to 99.97%.

Evaluation metrics maintained consistency across rounds, with 99.44% accuracy and an Average Precision score of 0.99. The relatively low ROC-AUC scores (progressing from 0.47 to 0.52) suggest the presence of class imbalance in the dataset. The final confusion matrix, showing 22,416 true positives and 123 false positives with no false negatives, confirms both high performance and the underlying class distribution characteristics.

Key Observations

1. Convergence Dynamics

- Training accuracy improved significantly, progressing from 86.89% in Round 1 to 99.93% in Round 3.
- Substantial and consistent loss reduction:
    - Training loss: 2.3289 → 0.5218 → 0.2810
    - Evaluation loss: 0.8237 → 0.1771 → 0.0646
- The rapid convergence pattern indicates highly efficient parameter aggregation across distributed nodes.

2. Performance Stability

- Evaluation accuracy maintained a steady 99.44% across all rounds, indicating robust generalization.
- Average Precision (AP) remained constant at 0.99, demonstrating stable positive class prediction.
- Minimal oscillation in training metrics suggests stable learning dynamics.

3. Metric Evolution

- Precision improved dramatically from 87.12% to 99.94%.
- Recall consistently enhanced: 98.45% → 99.89% → 100.00%.
- F1-Score progression (92.44% → 99.87% → 99.97%) indicates balanced improvement in precision and recall.

4. Class Distribution Analysis

- Final confusion matrix reveals:

    - True Positives: 22,416
    - False Positives: 123
    - False Negatives: 0
    - True Negatives: 0
- The absence of false negatives suggests perfect sensitivity for positive class detection.
- The presence of 123 false positives indicates a slight bias toward positive class prediction.

5. ROC-AUC Considerations

- Modest ROC-AUC improvement: 0.47 → 0.49 → 0.52.
- Relatively low ROC-AUC scores despite high accuracy suggest:
    - Significant class imbalance in the dataset.
    - Model optimization focused on majority class performance.
    - Opportunity for improvement in discriminative capability.

6. Training Efficiency

- Near-optimal performance achieved by Round 2, with marginal improvements in Round 3.
- Consistent decrease in loss values across training and evaluation metrics indicates effective knowledge transfer between federated nodes.
- High evaluation accuracy throughout training suggests minimal overfitting.

7. Practical Implications

- The model demonstrates production-ready performance metrics by Round 3.
- Perfect recall achievement makes it particularly suitable for applications where false negatives are costly.
- Consistent evaluation metrics indicate reliable model behavior in real-world scenarios.
- Class imbalance indicated by ROC-AUC scores should be considered in deployment planning.

These observations suggest that while the federated learning approach achieved high performance metrics, addressing class imbalance and optimizing ROC-AUC scores could further enhance the model. **The system's perfect recall and high precision make it well-suited for applications**

where minimizing false negatives is crucial, though false positives' impact should be considered in deployment contexts.

5. Conclusion

6. Future Work

References

[1] S. Shirvani Moghaddam, “The Past, Present, and Future of the Internet: A Statistical, Technical, and Functional Comparison of Wired/Wireless Fixed/Mobile Internet,” *Electronics*, vol. 13, no. 10, Art. no. 10, Jan. 2024, doi: 10.3390/electronics13101986.

[2] “The state of connectivity: Wi-Fi® momentum in 2024 | Wi-Fi Alliance.” Accessed: Jan. 04, 2025. [Online]. Available: https://www.wi-fi.org/beacon/the-beacon/the-state-of-connectivity-wi-fi-momentum-in-2024

[3] “IoT connections worldwide 2022-2033,” Statista. Accessed: Sep. 01, 2024. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[4] D. Setyadi, Karnowahadi, and E. Sulistyani, “Probability Model for Looking for a Job Educated Job Seeker at the Labor Market in Central Java Province (Sakernas Data),” presented at the International Conference on Management, Business, and Technology (ICOMBEST 2021), Atlantis Press, Nov. 2021, pp. 39–45. doi: 10.2991/aebmr.k.211117.006.

[5] “Review education policies - Education GPS - OECD: Labour market outcomes.” Accessed: Sep. 01, 2024. [Online]. Available: https://gpseducation.oecd.org/revieweducationpolicies/#!node=41763&filter=all

[6] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, “Network Anomaly Detection: Methods, Systems and Tools,” *IEEE Commun. Surv. Tutor.*, vol. 16, no. 1, pp. 303–336, 2014, doi: 10.1109/SURV.2013.052213.00046.

[7] S. I. Perez and R. Criado, “Increasing the Effectiveness of Network Intrusion Detection Systems (NIDSs) by Using Multiplex Networks and Visibility Graphs,” *Mathematics*, vol. 11, no. 1, p. 107, 2023, doi: 10.3390/math11010107.

[8] M. E. Aminanto and K. Kim, “Deep Learning in Intrusion Detection System: An Overview”.

[9] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, “Deep Learning Approach for Intelligent Intrusion Detection System,” *IEEE Access*, vol. 7, pp. 41525–41550, 2019, doi: 10.1109/ACCESS.2019.2895334.

[10] E. M. Campos *et al.*, “Evaluating Federated Learning for intrusion detection in Internet of Things: Review and challenges,” *Comput. Netw.*, vol. 203, p. 108661, Feb. 2022, doi: 10.1016/j.comnet.2021.108661.

[11] P. Kairouz *et al.*, “Advances and Open Problems in Federated Learning,” *Found. Trends® Mach. Learn.*, vol. 14, no. 1–2, pp. 1–210, Jun. 2021, doi: 10.1561/2200000083.

[12] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, “A review of applications in federated learning,” *Comput. Ind. Eng.*, vol. 149, p. 106854, Nov. 2020, doi: 10.1016/j.cie.2020.106854.

[13] B.-S. Lee, J.-W. Kim, and M.-J. Choi, “Federated Learning Based Network Intrusion Detection Model,” in *2023 24st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Sep. 2023, pp. 330–333. Accessed: Sep. 09, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10258140/?arnumber=10258140

[14] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, “A survey on security and privacy of federated learning,” *Future Gener. Comput. Syst.*, vol. 115, pp. 619–640, Feb. 2021, doi: 10.1016/j.future.2020.10.007.

[15] E.-H. Qazi, M. Imran, N. Haider, M. Shoaib, and I. Razzak, “An intelligent and efficient network intrusion detection system using deep learning,” *Comput. Electr. Eng.*, vol. 99, p. 107764, Apr. 2022, doi: 10.1016/j.compeleceng.2022.107764.

[16] S. Sivamohan, S. S. Sridhar, and S. Krishnaveni, “An Effective Recurrent Neural Network (RNN) based Intrusion Detection via Bi-directional Long Short-Term Memory,” in *2021 International Conference on Intelligent Technologies (CONIT)*, Jun. 2021, pp. 1–5. doi: 10.1109/CONIT51480.2021.9498552.

[17] A. Elsherif, “Automatic Intrusion Detection System Using Deep Recurrent Neural Network Paradigm,” *J. Inf. Secur. Cybercrimes Res.*, 2018, doi: 10.26735/16587790.2018.003.

[18] A. H. Mirza and S. Cosan, “Computer network intrusion detection using sequential LSTM Neural Networks autoencoders,” in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, May 2018, pp. 1–4. doi: 10.1109/SIU.2018.8404689.

[19] P. S. Muhuri, P. Chatterjee, X. Yuan, K. Roy, and A. Esterline, “Using a Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) to Classify Network Attacks,” *Information*, vol. 11, no. 5, Art. no. 5, May 2020, doi: 10.3390/info11050243.

[20] C. Park, J. Lee, Y. Kim, J.-G. Park, H. Kim, and D. Hong, “An Enhanced AI-Based Network Intrusion Detection System Using Generative Adversarial Networks,” *IEEE Internet Things J.*, vol. 10, no. 3, pp. 2330–2345, Feb. 2023, doi: 10.1109/JIOT.2022.3211346.

[21] B. Ingre and A. Yadav, “Performance analysis of NSL-KDD dataset using ANN,” in *2015 International Conference on Signal Processing and Communication Engineering Systems*, Jan. 2015, pp. 92–96. doi: 10.1109/SPACES.2015.7058223.

[22] M. Sheikhan, Z. Jadidi, and A. Farrokhi, “Intrusion detection using reduced-size RNN based on feature grouping,” *Neural Comput. Appl.*, vol. 21, no. 6, pp. 1185–1190, Sep. 2012, doi: 10.1007/s00521-010-0487-0.

[23] H. C. Altunay and Z. Albayrak, “A hybrid CNN+LSTM-based intrusion detection system for industrial IoT networks,” *Eng. Sci. Technol. Int. J.*, vol. 38, p. 101322, Feb. 2023, doi: 10.1016/j.jestch.2022.101322.

[24] Z. Chen, N. Lv, P. Liu, Y. Fang, K. Chen, and W. Pan, “Intrusion Detection for Wireless Edge Networks Based on Federated Learning,” *IEEE Access*, vol. 8, pp. 217463–217472, 2020, doi: 10.1109/ACCESS.2020.3041793.

[25] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Jul. 2009, pp. 1–6. doi: 10.1109/CISDA.2009.5356528.

[26] “Federated Learning | TensorFlow Federated,” TensorFlow. Accessed: Jan. 05, 2025. [Online]. Available: https://www.tensorflow.org/federated/federated_learning

[27] “Flower Framework.” Accessed: Jan. 05, 2025. [Online]. Available: https://flower.ai/docs/framework/