Leveraging Federated Learning and XAI For Privacy-aware and Lightweight Edge Training In Network Traffic Classification

Azizi Ariffin
Faculty of Computer Science and Information Technology
Universiti Malaya
Kuala Lumpur, Malaysia
mazizi@fskm.uitm.edu.my

Faiz Zaki
Faculty of Computer Science and Information Technology
Universiti Malaya
Kuala Lumpur, Malaysia
faizzaki@um.edu.my

Nor Badrul Anuar
Faculty of Computer Science and Information Technology
Universiti Malaya
Kuala Lumpur, Malaysia
badrul@um.edu.my

***Abstract*—The exponential growth of internet traffic causes significant challenges for network traffic classification, such as maintaining data privacy and requiring more computing resources. To address such challenges, moving the processing to data instead of the opposite shows some promise. Related technologies, such as edge computing, present a potential solution as they reduce latency and bandwidth and consider data privacy. However, edge computing often runs on resource-constraint devices, thus making complex model training like deep learning challenging. As such, this paper proposes a distributed and lightweight edge training method for network traffic classification using federated learning and XAI, which are currently underexplored in the domain. In doing so, we managed to run deep learning on edge devices and preserve data privacy. We evaluate the proposed method using the ISCXVPN2016 dataset on three NVIDIA Jetson devices to emulate edge devices. The results show a 36.71% improvement in F1 score, training time, memory, and GPU utilization when benchmarked against the more conventional centralized training.**

Keywords—Network Traffic Classification, Deep Learning, Edge Training, XAI, Federated Learning, Encrypted Traffic

I. INTRODUCTION

The exponential growth of applications and services on the Internet resulted in a significant increase in network traffic and protocols. According to Cisco's report [1], there will be an estimated 5.3 billion internet users in 2023, representing a 36% increase from 3.9 billion users in 2018. The report also states that the average global broadband speed doubled from 45.9 Mbps in 2018 to 110.4 Mbps in 2023. This growth of network traffic has created significant challenges for administrators to classify traffic and enforce policies such as content filtering, quality of service (QoS), and identification of malicious traffic [2] due to increased processing, storage and privacy requirements. To address these challenges, the network traffic classification (NTC) solution needs to leverage technologies such as cloud, big data, machine learning, deep learning, and edge computing to improve NTC efficiency and performance with the growth of network traffic.

The introduction of edge computing (EC) has brought a new paradigm in network traffic classification (NTC), enabling the monitoring and management of network traffic at the network edge, such as the router [3]. This paradigm distributes processing among multiple end devices closer to the user or data source instead of a centralized host. This approach offers several benefits, including reduced processing latency, bandwidth, and power consumption [4]. Furthermore, processing on edge helps avoid unwanted disclosure of private data over the network [5].

However, performing NTC on edge using deep learning (DL) techniques can be computationally challenging due to the resource-constrained nature of edge devices. Despite this challenge, the use of DL in NTC gained traction among practitioners, as it achieves excellent learning ability with large data and high accuracy without performing feature engineering beforehand [6]. Nevertheless, training an NTC DL model on edge devices using a conventional approach where it is done centrally on a single edge device can lead to issues such as:

- Training a complex DL model for NTC is computationally demanding, with high CPU, memory, and GPU resource requirements. Additionally, creating an accurate NTC model requires a high volume of traffic data, which overloads a single resource-constrained edge host [7].
- Transferring large amounts of data for centralized training adds latency and bandwidth consumption. Large datasets for model training between edge hosts require additional processing, storage and consume in-band bandwidth, making it costly for metered connections.
- Centralized training of NTC models on edge devices can compromise data privacy and security [8]. Raw data transmitted over the network can be intercepted, and centralizing training requires disclosing sensitive data to the central entity. Collaboration between organizations bound by data privacy regulations such as GPDR is also restricted.

Thus, model training for NTC on edge requires a lightweight and distributed DL training method. To this end, several works [6], [9], [10] proposed lightweight approaches for NTC. [6] proposes a preprocessing packet scheme called e-cut to reduce training overhead and an attention-based long short-term memory (LSTM) called ABL-TC. The results show that the e-cut approach reduced training time by 69% with only a 3.1% reduction in accuracy. However, the complexity of the preprocessing task introduced additional latency during edge operation, and extensive preprocessing to create a handcrafted dataset negated the benefits of DL.

Furthermore, the proposed ABL-TC is a centralized training approach where a central entity discloses data, reducing data privacy preservation. The processing task focuses only on a single host. In [9], a lightweight approach uses multi-head attention for parallel learning and ID Convolutional Neural Network (CNN) layers to perform feature extraction for a single packet. The results show a 90% reduction in model weight size and a 49.7% reduction in training time while maintaining comparable model

2023 IEEE International Conference on Computing (ICOCO) | 979-8-3503-0268-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICOCO59262.2023.10397836

performance. However, as in previous work, the centralized DL training method discloses user data, and the approach requires feature engineering in its preprocessing step.

Similarly, the approach in [10] uses a multi-head attention mechanism during flow-level feature extraction to reduce model training time with a simpler process during preprocessing. Although the proposed method is lightweight and can reduce model training time, the centralized approach discloses endpoint data. It is worth noting that all three methods discussed require further evaluation in edge devices with resource-constrained setups.

Several techniques proposed distributed DL training approaches, including distributed processing [11], gossip learning [12], and federated learning (FL) [8], [13]. FL is the most promising approach, allowing the local model to be trained on the edge device using local data. Then, only the parameters of the local model are sent to the aggregation server to form a global model using multiple local models. This ensures that local data privacy is preserved and the training is done locally on the edge node. In [8], FL was used for a distributed DL-based traffic classifier, enabling the use of local data for model training while preserving IoT data privacy. The evaluation shows that the proposed FL method achieved a 15.3% higher F1 score on the Cambridge dataset but required 97% longer training time to achieve that level of accuracy, indicating that the proposed method is computationally intensive.

Meanwhile, the work of [13] leverages the FL technique for classifying internet traffic among multiple clients while preserving privacy. The evaluation results show that the method achieved 92% accuracy but was not benchmarked with any other centralized learning method. Moreover, both methods require further evaluation in an edge environment.

This paper proposes a distributed edge training method for NTC that preserves the data privacy of end devices while keeping DL training lightweight based on identified gaps in previous approaches. The proposed method leverages FL to distribute DL training on each edge device, forming a local model using local data. The edge devices then send the local model weight to the aggregation server to form an accurate global model while maintaining data privacy. Additionally, to keep the training process lightweight, the proposed method utilizes an XAI technique called DeepSHAP [14] for feature reduction, requiring only simple preprocessing as raw packet bytes are used as features. The proposed method performs feature reduction on the byte's numbers based on the inference made by the initial model after it has been trained. The ISCXVPN2016 dataset [15], which contains real network traffic, was used to evaluate the proposed method on Nvidia Jetson nano as edge devices. The evaluation results show that MLP training had a 1.31% increase in F1-score, a 4.64% reduction in training time, a 36.71% reduction in memory utilization, and a 13.33% reduction in GPU utilization compared to centralized training. For CNN training, there was a 3.77% increase in F1-score, a 7.16% reduction in memory utilization, and a 5.32% reduction in GPU utilization, but a slight 1.7% increase in training time. Additionally, the proposed method preserves endpoint data privacy for MLP and CNN training. The contributions of this work can be summarized as follows:

- Develop a distributed DL training using the FL approach to preserve data privacy on edge devices.
- Develop a lightweight DL training method for the edge, which only requires simple preprocessing and feature reduction using the XAI technique.
- Benchmark the proposed method on real resource-constraint edge devices and network environments.

II. RELATED WORK

This section discusses the related work and state-of-the-art in NTC, edge training, FL, and XAI.

A. Network Traffic Classification at The Edge

NTC involves identifying and categorizing the type of traffic based on packet and flow characteristics. According to [2], there are six categories of NTC techniques: port-based, Deep Packet Inspection (DPI), supervised classification, unsupervised clustering, semi-supervised, and DL. The use of DL in NTC has recently gained traction as it exhibits excellent learning quality without handcrafted features. In [16], a DL-based classifier was proposed to classify encrypted traffic using Multi-Layer Perceptron (MLP), CNN, and stacked autoencoders (SAE) for feature reduction. Similarly, the work of [17] proposes a DL-based multitask learning framework that predicts bandwidth requirement and flow duration along with the traffic class to solve the data labelling issue in DL. Both works demonstrate high accuracy in classifying network traffic. However, both were designed to be implemented on a conventional host with ample resources and required disclosure of training data to a centralized host.

Performing DL training on edge devices or nodes is known as edge training. It reduces latency and bandwidth utilization issues. However, performing edge training is still uncommon [4] compared to DL inference on edge. For instance, in [18], the model is trained on another host and converted into a lightweight model using TensorFlow Lite before deploying it on edge. This prohibits the edge from developing the model using localized data. Due to limited computational resources on edge devices, the DL technique must be lightweight before the model can be trained on the edge. In [6], a packet preprocessing scheme was proposed to reduce the overhead of DL training. Nonetheless, the method was designed for centralized training and risks leaking endpoint data.

B. Federated Learning

The utilization of FL as a distributed learning scheme that permits multiple nodes or devices to train a shared model without exchanging or centralizing their local data has been demonstrated by [13] and [8]. [13] utilized FL to distribute DL training for the internet traffic classifier, while [8] employed FL for distributed training of the IoT traffic classifier. The works display high classifier accuracy and the ability to maintain data privacy. However, both works lack consideration towards lightweight training in FL.

C. Explainable AI

Various techniques have been developed to aid users in understanding how DL models make inferences. Among these techniques is Shapley Additive exPlanations (SHAP) [14], a Model-agnostic technique that allows users to acquire local and global explanations of DL models. Another known XAI technique is local interpretable model-agnostic explanations (LIME) [19], which provides a local explanation by generating a new dataset consisting of perturbed samples and the corresponding predictions of the DL model. These XAI techniques help users understand the decision-making process

of a DL model, which can be useful in various applications such as feature reduction. Table I summarizes and maps the approach or method used in related work regarding deep learning, lightweight training, data privacy preservation, and edge training based on the review conducted.

TABLE I. SUMMARY OF RELATED WORK

Approach/Method	*DL*	*LW*	*PP*	*ET*
P. Wang et al., 2018 [16]	✓	×	×	×
S. Rezaei and X. Liu, 2019 [17]	✓	×	×	×
E. Chen and A. Perez-Pons, 2022 [18]	✓	✓	×	×
W. Wei et al., 2022 [6]	✓	✓	×	×
H. Mun and Y. Lee, 2021 [13]	✓	×	✓	×
M. Abbasi,A. Taherkordi, and A.Shahraki,2022[8]	✓	✓	✓	×
This paper	✓	✓	✓	✓

DL – Deep Learning, LW – Lightweight, PP – Privacy Preserving, ET – Edge Training

III. METHODOLOGY

This section discusses the proposed method in detail, including the system design, NTC model training via deep learning, local and global model training and feature reduction using XAI techniques. Figure 1 shows the overall research flowchart of this paper.

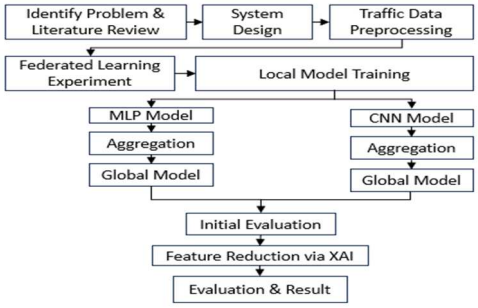

Fig. 1. Research Flowchart

The research starts by identifying the problem and conducting literature reviews. Subsequently, this paper designs a system to tackle the problem. Before performing experiments to assess the system, the traffic data undergoes preprocessing. The research then conducts local model training using either MLP or CNN approaches. The local model aggregation forms a global model, which undergoes initial evaluation. The model then undergoes feature reduction through XAI. The research concludes by re-evaluating the model with reduced features and reporting the results.

A. System Design

The proposed NTC method distributes local model training on edge devices to create a local model with local data. The trained local model weight is then sent to the aggregation server to form an accurate global model using Federated Averaging (FedAvg) with other local models. Packet byte features are reduced using the explainable AI technique called DeepSHAP to keep the training process lightweight. The proposed method is designed for edge and networked environments, allowing different domains or organizations to collaborate using their local traffic data to form more accurate NTC models. Figure 2 displays the system design diagram of the proposed method. The diagram depicts the interaction between edge devices and the aggregation server for uploading local models and updating the global NTC model via a server-client architecture during training. Communication is secured via a secure socket layer (SSL) to ensure the model update's security. The edge devices utilized in the proposed method, such as firewall, gateway router, and Wi-Fi router, vary depending on the local network setup.

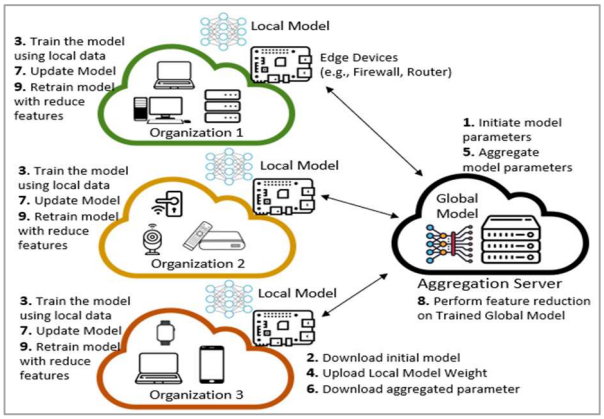

Fig. 2. System Design Diagram

Based on Figure 2, the training process of the NTC model involves several steps:

- **Step 1 (Server):** The aggregation server creates an initial model with empty weight W = 0 based on MLP or CNN. The techniques were chosen as they exhibit high accuracy for a tabular dataset. The server also set the training hyperparameters such as round number r, training epochs E, learning rate η, batch size β and training technique for all clients.
- **Step 2 (Network):** Download the initial model. The client downloads the initial model with the hyperparameters via SSL.
- **Step 3 (Edge Client):** Train the model using local data. The edge client trains the local model using the initial model $W_r^k = 0$ using local dataset shard $\mathbb{X}_k$.
- **Step 4 (Network):** Upload local model weight. The edge client uploads the trained model weight W_{r+1}^k To server via SSL.
- **Step 5 (Server):** Aggregate model parameters. The server aggregates the local model weight using a federated averaging algorithm, as shown in E.q. 2.
- **Step 6 (Network):** Download aggregated parameters. The server sends the aggregated model weight back to all client $W_{r+1}^k \rightarrow$all k. Client downloads via SSL.
- **Step 7 (Edge Client):** Update the model. The edge client updates their local model based on the new model weight, thus learning a new traffic class. Client and server repeat steps 3-7 until they reach maximum round number r.
- **Step 8 (Server):** Perform feature reduction. Once the max round r is reached, the server performs feature reduction on the global model via DeepSHAP. The dataset bytes (features) are reduced to $0 < \mathbb{X} \leq 100$.
- **Step 9 (Edge Client):** Retrain the model with reduced features. The server-client repeats the FL process from steps 1-7 with fewer features for training.

B. Network Traffic Classification Model via Deep Learning

The method provides the flexibility to train the model using various DL techniques, provided that the model weight can be obtained. In this paper, Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN) are chosen as both techniques have demonstrated high accuracy as network

traffic classifiers [16], [17] for tabular data. The preprocessing of the proposed method is as follows.

1) Preprocessing: The process is loosely based on the work [16] that transforms raw bytes (.PCAP) files into packet byte matrix. The four steps involve Parsing, Padding/Truncating, Matrix conversion, and Normalization. In parsing, the method will read the raw traffic (e.g. from . PCAP file) and remove the data link or ethernet layer header as it is only local significance and unhelpful in traffic classification. After that, the method fixed the size of each data packet x, where $0 < x \leq 740$ bytes, by padding or truncating the packet. The 740 bytes is half of the maximum transmission unit (MTU) and adequate to represent a packet in the edge environment generally. Each byte becomes the input feature for the DL training. The result of padding/truncating all packets in the dataset is packet byte vector (PBV), $X_i = \{x_{i1}, x_{i2} \ldots x_{ij}\}$, where i is the dataset and j is j-th byte X_i. Each PBV is associated with traffic label L (e.g. Skype, Netflix). After that, the PBV is processed into a packet byte matrix (PBM) by combining PBV from all processed raw datasets with its label. The matrix conversion process is as $\mathbb{X} = \{X_1^T, X_2^T \ldots X_i^T\}^T$, where i is the number of PBV datasets. After matrix conversion and label association, the raw dataset is processed into a form shown in E.q.1. The last step involves normalizing the PBM value into [0, 1] by column axis for faster classification, and some DL technique requires the input value to be normalized.

$$\mathbb{X} = \begin{bmatrix} X_1 \\ \ldots \\ X_i \end{bmatrix} \leftarrow \begin{bmatrix} \mathrm{L}_1 \\ \ldots \\ \mathrm{L}_i \end{bmatrix} \quad (1)$$

2) Multi-Layer Perceptron (MLP): The MLP was chosen to represent a simple DL technique for NTC. In the proposed method, the training epochs were set to $E = 6$ for mini-batch and $E = 36$ for multi-epochs. The *learning rate is* $\eta = 0.001$, and the *batch size* β*=64*. The layers setup for the MLP is listed in Table II.

TABLE II. MLP TRAINING LAYERS SETUP

Layer	*Output Shape*	*Activation Function*
Input	x	-
Hidden FCN	6	ReLU
Hidden FCN	6	ReLU
Output	(10,y)	Softmax

3) Convolutional Neural Network (CNN): The CNN technique represents a more complex DL model with FL in this work. For CNN, the training epochs were set to $E = 6$ for mini-batch and $E = 36$ for multi-epochs. The *learning rate is* $\eta = 0.0001$, and the *batch size* β*=64*. The layers setup for the CNN is listed in Table III.

TABLE III. CNN TRAINING LAYERS SETUP

Layer	*Output Shape*	*Activation Function*	*Configuration*
Input	(x,t,F)	ReLU	-
Hidden 1D-CNN	32	ReLU	KS=3, Padding=Same
Hidden 1D-CNN	32	ReLU	KS=3, Padding=Same
MaxPool	32	ReLU	PS=2, Padding=Same
Flatten	128	ReLU	-
Hidden FCN	256	ReLU	-
Output	(10,y)	Softmax	-

KS – Kernel Size, PS – Pool Size, t – Timeseries, F-Features

C. Local and Global Model Training in FL

The FL process facilitates the training of local and global models for NTC to preserve the privacy of the endpoint data. This paper used Flower Framework [20] to implement FL using Python3. The global model is formed by aggregating multiple local models from the edge devices and averaging the model's weight using the FedAvg algorithm[21], as E.q. 2.

$$\mathrm{W_{avg}} = \sum_{i=1}^{k} \frac{\mathrm{n_i}}{\mathrm{N}} \mathrm{W_i} \quad (2)$$

From the equation, $\mathrm{W_1}, \mathrm{W_2}, \ldots \mathrm{W_k}$ is the weight of the models from edge devices, each with local dataset $\mathbb{X}$ size $\mathrm{n_k}$ and N is the total dataset size $\mathrm{N}=\mathrm{n_1},\mathrm{n_2},\ldots\mathrm{n_k}$. The averaging is done for each round r, with k number of clients. This method sets the round number to three because averaging the model once leads to poor performance across all data [22]. The detailed federated learning process of the proposed method is shown in Algorithm 1.

ALGORITHM 1: Federated Averaging

Input: The number of clients k, the number of local training epochs E, learning rate η, batch size β
Output: Aggregated model weight W_{r+1}^k

Initialize the global model parameters $\mathrm{W} = 0$, r=3
For each round r do:
  Select a subset of clients to participate in the training round.
  For each client k do:
    Send current global weight to client k: $W_r^k = W$
    Client k trains the model on its dataset $\mathbb{X}_\mathrm{k}$ using
    W_r^k *and* generate local model parameters:
    $\mathrm{W_{r+1}^k}$=argmin($\mathrm{W_r^k}$, loss function($\mathrm{W_r^k}$))
    Client k send W_{r+1}^k to the aggregation server.
  End For
  The aggregation server aggregates the local model
  from clients and compute weighted average $\mathrm{W_{r+1}^k}$= Eq. (2)
  The aggregation server distributes the updated global
  model parameters to clients $W_{r+1}^k \rightarrow all\ k$
End For
Repeat until max round r is reached

In FL, the global model is aggregated from the local model via two approaches: 1) Mini-Batch (MB) or 2) Multi-Epoch (ME) aggregation [23]. The MB aggregation is where the model is trained with mini-batch epochs $E = 6$ in each round r before been aggregated. While in ME approach the model is trained for all epochs E in each round r, thus in ME the $E = 36$, r=3.

D. Feature Reduction using the XAI Technique

After training the initial model, the proposed method reduces the number of raw bytes to keep the DL training lightweight. This feature reduction is achieved using the DeepSHAP [14] technique to identify the impact or importance of each feature when the model makes the prediction. DeepSHAP is a game theory-based variant of the SHAP method. To determine the importance of each feature (bytes), the Shapley value for each feature is calculated using a simplified formula, as shown in E.q.3.

$$SHAP_{j(x)}=\left(\frac{1}{M}\right)\left(\sum_{i=1}^{S} f(x\_s\cup\{j\})\text{-}f(x\_S)\right)(g(S\cup\{j\})\text{-}g(S)) \quad (3)$$

From the equation, M is the set of all features, S is a subset of M without j, f is the deep learning model being explained, and g is the model's gradient function. This paper randomly chooses 5000 instances of the training dataset, and 1000 instances of the testing dataset are fed into DeepSHAP. Based on the SHAP value, we choose 100 features (Bytes) with the highest impact on model inference, which reduces the dataset bytes(features) to $0 < \mathbb{X} \leq 100$.

IV. PERFORMANCE EVALUATION

This section discusses the setup and result of experiments to validate the performance of the proposed method.

A. Experimental Setup

1) Testbed Environment: Python3.8 was used to implement the proposed method with TensorFlow 2.4 as the DL library, PyShark for raw packet processing, Scikit-Learn for preprocessing, and Flower 14 for the FL framework. The source code for the proposed method is available on GitHub [24]. The experiment was conducted on three Nvidia Jetson Nano devices, each with a quad-core ARM A57 CPU, 4 GB 64-bit LPDDR4 memory, 128-core Maxwell GPU, and 64 GB MicroSD storage. All devices are connected via a 1GB unmanaged switch. In previous works [7], Jetson Nano is used to emulate resource-constrained edge devices.

2) Dataset: This work chooses the ISCX 2016 VPN-nonVPN dataset [15] representing real-world network traffic. We took 400k samples with ten traffic classes such as AIM, Email, FB Audio, FB Chat, Gmail Chat, Hangouts, ICQ, Netflix, Spotify and Youtube. The dataset $\mathbb{X}$, which has undergone preprocessing, is randomly split into training and testing datasets with a 70:30 ratio. After that, the training and testing dataset is split into Independent and identically distributed (IID) data shards [22], $\mathbb{X}_1, \mathbb{X}_2 \ldots \mathbb{X}_k$. For experiment which involve two client k, the dataset is split into 2 shards, for three client k, 3 shards for each clients.

3) Evaluation Metrics: The experiment collected metrics to evaluate model performance, including F1-score, Training Time, Peak memory utilization, and Peak GPU utilization. The F1-Score evaluated classifier performance using micro, macro, and weighted averages. The training time for FL refers to the number of communication rounds required to train the model, as each round involves transmitting and aggregating model updates between the clients and the server. The FL training time can be estimated as T = r * (T_comm + T_comp), where r is the number of rounds, T_comm is the time required to communicate between the clients and the server, and T_comp is the time required for local computation on the edge clients.

B. Experiment Result

In this experiment, we compared the performance and resource utilization of centralized and federated learning with feature reduction on two or three-edge client k. We also compared the performance between Mini-Batch (MB) and Multi-Epoch (ME) aggregation approaches in MLP and CNN. The results are shown in Figures 3 and 4, which display the performance comparison.

Figure 3 shows that the FL ME approach for MLP achieved significantly higher performance than the centralized approach, with a 2.35% increase in performance for two clients and 1.31% for three clients. The CNN result in Figure 4 shows that the FL ME approach also achieved significantly higher performance, with a 4.45% increase for two clients and 3.77% for three clients. On the other hand, the FL MB approach showed slightly lower performance for both clients. This is expected as the ME approach has greater training epochs in each FL round than the MB and centralized approach.

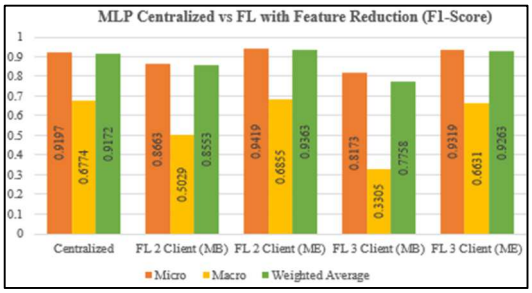

Fig. 3. Experiment 2 MLP F1 Score.

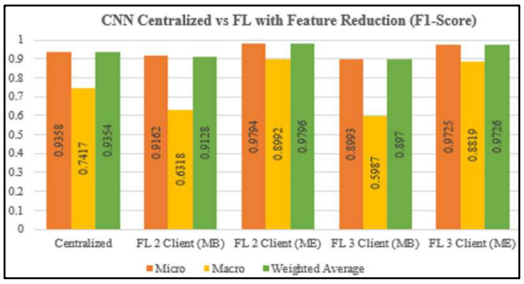

Fig. 4. Experiment 2 CNN F1 Score.

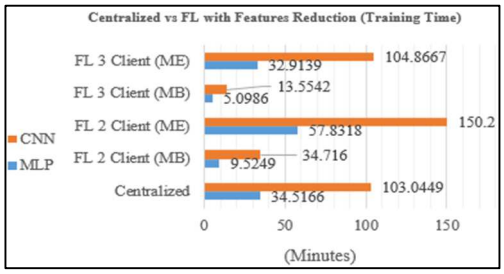

Fig. 5. Experiment 2 Training Time.

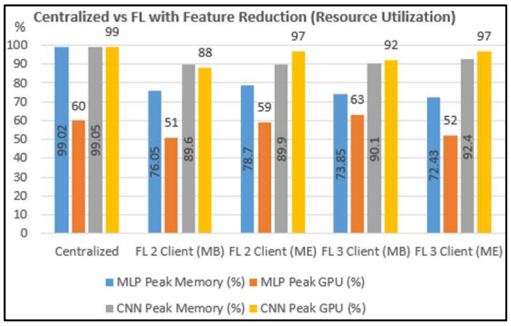

Fig. 6. Experiment 2 Resource Utilization.

Regarding training time, Figure 5 shows that both the FL MB and ME achieve lower training times for MLP, although the ME approach performs has more epochs. The FL MB recorded a 72.4% reduction for two clients and 85.22% for three clients. The FL ME recorded a 4.64% reduction for three

clients and a 40.3% increase for two clients. Meanwhile, for CNN, the FL MB achieved a 66.3% reduction for two clients and 86.84% for three clients. For FL ME, the approach recorded a 31.3% increase in training for two clients and a slight 1.7% increase in training time. The FL MB and ME approach achieve lower resource utilization for comparison regarding resource utilization, as shown in Figure 6, as processing is distributed. We can conclude that the proposed method with ME aggregation and three clients achieved the best overall performance compared to the centralized approach. The setup achieved a 1.31% increase in F1-score, a 4.64% reduction in training time, a 36.71% reduction in memory utilization and a 13.33% reduction in GPU utilization for MLP training. While a 3.77% increase in the F1-score, a 7.16% reduction in memory utilization, and a 5.32% reduction in GPU utilization but a slight 1.7% increase in training time for CNN training. Although the ME has greater epochs in both trainings. The increase in performance and reduction in training time and resource utilization are gained while having the benefits of preserving the privacy of the local data in the edge devices.

V. CONCLUSION

This paper proposes a distributed method that leverages FL to distribute the FL training process to multiple edge devices and preserve local data privacy. The method also utilizes explainable AI techniques to reduce the number of features for lightweight edge training. We benchmark the proposed method using the ISCX2016 dataset on NVIDIA Jetson to emulate real-edge devices. The experiment shows that the proposed method achieves higher performance with ME and three client setups, with shortened training time and reduced resource utilization. The outcome led to lightweight implementation of FL with improved NTC accuracy. For future work, we will improve the proposed method by fine-tuning the FL and DL algorithms and evaluating them with larger edge devices. Additionally, we will study the impact and mitigation of the malicious byzantine client on the global model performance.

REFERENCES

[1] "Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper - Cisco," 2020. Accessed: Mar. 21, 2023. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] A. Azab, M. Khasawneh, S. Alrabaee, K.-K. Raymond Choo, and M. Sarsour, "Network traffic classification: Techniques, datasets, and challenges," Digit. Commun. Networks, 2022, doi: 10.1016/j.dcan.2022.09.009.

[3] B. Mohammed et al., "Edge Computing Intelligence Using Robust Feature Selection for Network Traffic Classification in Internet-of-Things," IEEE Access, vol. 8, pp. 224059–224070, 2020, doi: 10.1109/ACCESS.2020.3037492.

[4] N. Kukreja et al., "Training on the Edge: The why and the how," Proc. - 2019 IEEE 33rd Int. Parallel Distrib. Process. Symp. Work. IPDPSW 2019, pp. 899–903, 2019, doi: 10.1109/IPDPSW.2019.00148.

[5] G. Paul and J. Irvine, "Privacy implications of wearable health devices," ACM Int. Conf. Proceeding Ser., vol. 2014-September, pp. 117–121, Sep. 2014, doi: 10.1145/2659651.2659683.

[6] W. Wei, H. Gu, W. Deng, Z. Xiao, and X. Ren, "ABL-TC: A lightweight design for network traffic classification empowered by deep learning," Neurocomputing, vol. 489, pp. 333–344, Jun. 2022, doi: 10.1016/J.NEUCOM.2022.03.007.

[7] S. K. Prashanthi, S. A. Kesanapalli, and Y. Simmhan, "Characterizing the Performance of Accelerated Jetson Edge Devices for Training Deep Learning Models," Proc. ACM Meas. Anal. Comput. Syst., vol. 6, no. 3, Dec. 2022, doi: 10.1145/3570604.

[8] M. Abbasi, A. Taherkordi, and A. Shahraki, "FLITC: A Novel Federated Learning-Based Method for IoT Traffic Classification," Proc. - 2022 IEEE Int. Conf. Smart Comput. SMARTCOMP 2022, pp. 206–212, 2022, doi: 10.1109/SMARTCOMP55677.2022.00055.

[9] J. Cheng, R. He, E. Yuepeng, Y. Wu, J. You, and T. Li, "Real-Time Encrypted Traffic Classification via Lightweight Neural Networks," 2020 IEEE Glob. Commun. Conf. GLOBECOM 2020 - Proc., Dec. 2020, doi: 10.1109/GLOBECOM42002.2020.9322309.

[10] J. Cheng et al., "MATEC: A lightweight neural network for online encrypted traffic classification," Comput. Networks, vol. 199, p. 108472, Nov. 2021, doi: 10.1016/J.COMNET.2021.108472.

[11] B. A. Bhuvaneswari and S. S., "Anomaly detection framework for Internet of things traffic using vector convolutional deep learning approach in fog environment," Futur. Gener. Comput. Syst., vol. 113, pp. 255–265, Dec. 2020, doi: 10.1016/J.FUTURE.2020.07.020.

[12] I. Hegedűs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," J. Parallel Distrib. Comput., vol. 148, pp. 109–124, Feb. 2021, doi: 10.1016/J.JPDC.2020.10.006.

[13] H. Mun and Y. Lee, "Internet traffic classification with federated learning," Electron., vol. 10, no. 1, pp. 1–18, 2021, doi: 10.3390/electronics10010027.

[14] S. M. Lundberg, P. G. Allen, and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," Adv. Neural Inf. Process. Syst., vol. 30, 2017, Accessed: Mar. 25, 2023. [Online]. Available: https://github.com/slundberg/shap

[15] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related features," ICISSP 2016 - Proc. 2nd Int. Conf. Inf. Syst. Secur. Priv., no. Icissp, pp. 407–414, 2016, doi: 10.5220/0005740704070414.

[16] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in SDN home gateway," IEEE Access, vol. 6, pp. 55380–55391, 2018, doi: 10.1109/ACCESS.2018.2872430.

[17] S. Rezaei and X. Liu, "Multitask Learning for Network Traffic Classification," Proc. - Int. Conf. Comput. Commun. Networks, ICCCN, vol. 2020-August, Jun. 2019, doi: 10.1109/ICCCN49398.2020.9209652.

[18] E. Chen and A. Perez-Pons, "Malware Network Traffic Classification on the Edge," Proc. - 2022 IEEE 19th Int. Conf. Mob. Ad Hoc Smart Syst. MASS 2022, pp. 754–758, 2022, doi: 10.1109/MASS56207.2022.00118.

[19] M. T. Ribeiro, S. Singh, and C. Guestrin, " 'Why Should I Trust You?': Explaining the Predictions of Any Classifier," NAACL-HLT 2016 - 2016 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. Proc. Demonstr. Sess., pp. 97–101, Feb. 2016, doi: 10.18653/v1/n16-3020.

[20] D. J. Beutel et al., "Flower: A Friendly Federated Learning Research Framework," Jul. 2020, Accessed: Mar. 29, 2023. [Online]. Available: https://github.com/adap/flower

[21] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," Proc. 20th Int. Conf. Artif. Intell. Stat. AISTATS 2017, Feb. 2016, Accessed: Mar. 29, 2023. [Online]. Available: https://arxiv.org/abs/1602.05629v4

[22] Kiyoshi Nakayama and G. Jeno, Federated Learning with Python: Design and implement a federated learning system and develop applications using existing frameworks, 1st Editio. Packt Publishing, 2022.

[23] V. Rey, P. M. Sánchez Sánchez, A. Huertas Celdrán, and G. Bovet, "Federated learning for malware detection in IoT devices," Comput. Networks, vol. 204, no. April 2021, p. 108693, 2022, doi: 10.1016/j.comnet.2021.108693.

[24] Azizi Ariffin and Faiz Zaki, “CoR-Cyber-Security-and-Network-UM/edge-ntc-fl,” Github, 2023. https://github.com/CoR-Cyber-Security-and-Network-UM/edge-ntc-fl (accessed Apr. 01, 2023).