Developing a Transferable Federated Network Intrusion Detection System

Abu Shafin Mohammad Mahdee Jameel*, Shreya Ghosh*, Aly El Gamal*
* School of Electrical and Computer Engineering, Purdue University, USA
Email: {amahdeej, ghosh64, elgamala}@purdue.edu

***Abstract*—Intrusion Detection Systems (IDS) are a vital part of a network-connected device. In this paper, we develop a deep learning based intrusion detection system that is deployed in a distributed setup across devices connected to a network. Our aim is to better equip deep learning models against unknown attacks using knowledge from known attacks. To this end, we develop algorithms to maximize the number of transferability relationships. We propose a Convolutional Neural Network (CNN) model, along with two algorithms that maximize the number of relationships observed. One is a two step data pre-processing stage, and the other is a Block-Based Smart Aggregation (BBSA) algorithm. The proposed system succeeds in achieving superior transferability performance while maintaining impressive local detection rates. We also show that our method is generalizable, exhibiting transferability potential across datasets and even with different backbones. The code for this work can be found at https://github.com/ghosh64/tabfidsv2.**

***Index Terms*—Network Intrusion Detection, Transferability, Federated Learning, Block-Based Smart Aggregation, Temporal Averaging.**

I. Introduction

Intrusion Detection Systems (IDS) are essential for protecting network infrastructures from unauthorized access and potential threats by identifying and mitigating the efforts of malicious actors. Historically, network intrusion detection systems have relied on traditional algorithms like Naive-Bayes classifiers, Random Forest classifiers, and Support Vector Machines (SVM) for threat detection [1]. Methods like stochastic gradient descent based models, hypergraph based machine learning systems, random forest models with class probability distributions have been shown to produce good intrusion detection performances [2], [3]. Statistical models such as Logistic Regression, SVMs, Decision Trees, Random Forests, Multi layer Perceptrons have also been shown to perform well with Feature selection methods [4].

While these methods have been foundational in effective IDS development, recent research highlights the enhanced performance of deep learning approaches and show that they surpass traditional algorithms in detecting network intrusions [5]. The popularity of these deep learning methods stems from their exceptional performance, and their ability to effectively manage imbalanced datasets, a prevalent issue in network security [6]. As a result, deep learning techniques are now considered benchmark standards in the development and evaluation of IDS datasets, playing a pivotal role in driving the evolution of future network security measures [7].

The surge in research within this area has resulted in the creation of numerous network traffic datasets, such as KDD 99 [8], NSLKDD [8], CIC-IDS 2017 [9], and ToN-IoT [10]. These datasets, often compiled using traffic monitoring tools like Wireshark, provide an extensive overview of network activities. However, their complex, high-dimensional nature presents a significant challenge for machine learning models in extracting meaningful information.

Contemporary deep learning models in intrusion detection are often trained to identify specific, previously known attacks or to learn from historical attack patterns. These models excel in detecting familiar threats with high accuracy. However, their capability significantly decreases when confronting novel or zero-day attacks—unseen and undocumented threats appearing in real-time. The unpredictable nature of these attacks represents a substantial challenge in network security, given the difficulty in predicting the spectrum of possible intrusions. Moreover, anomaly detection approaches tend to categorize all unknown network intrusions under a single class, thereby failing to offer a comprehensive analysis of model transferability.

The concept of model transferability has gained traction, positing that models could recognize a broader array of attacks beyond their initial training scope. Research into transferability, such as highlighted in [11] and [12], explores this potential. In [11], models are trained on a specific dataset and evaluated on another, focusing on a single type of attack. The work presented in [12] extends this approach by investigating the model's ability to generalize across different attack types, training on one and testing on another. However, it is worth noting that both studies are performed on pre-selected training and testing attack classes.

Recent advances have highlighted a critical challenge traditionally linked to deep learning: the substantial computational power and memory it requires [13]. Deep learning models are now embedded directly into hardware. The concept of model transferability plays a pivotal role here, allowing for the use of smaller datasets while maintaining the model's ability to generalize across various attack vectors. This strategy significantly cuts down on training times and reduces the computational load and memory requirements, which is especially beneficial for deployment on resource-constrained edge devices.

This integration of edge based federated learning into the realm of Intrusion Detection Systems (IDS) signifies a pivotal shift towards distributed, privacy-preserving, and computationally efficient cybersecurity measures [14]. Research, including findings from [14] and [15], [16] highlight federated learning's capability to surpass self-learning models and achieve comparable accuracy to centralized systems in IDS applications. The exploration of model transferability within federated learning

The first two authors contributed equally to this paper.

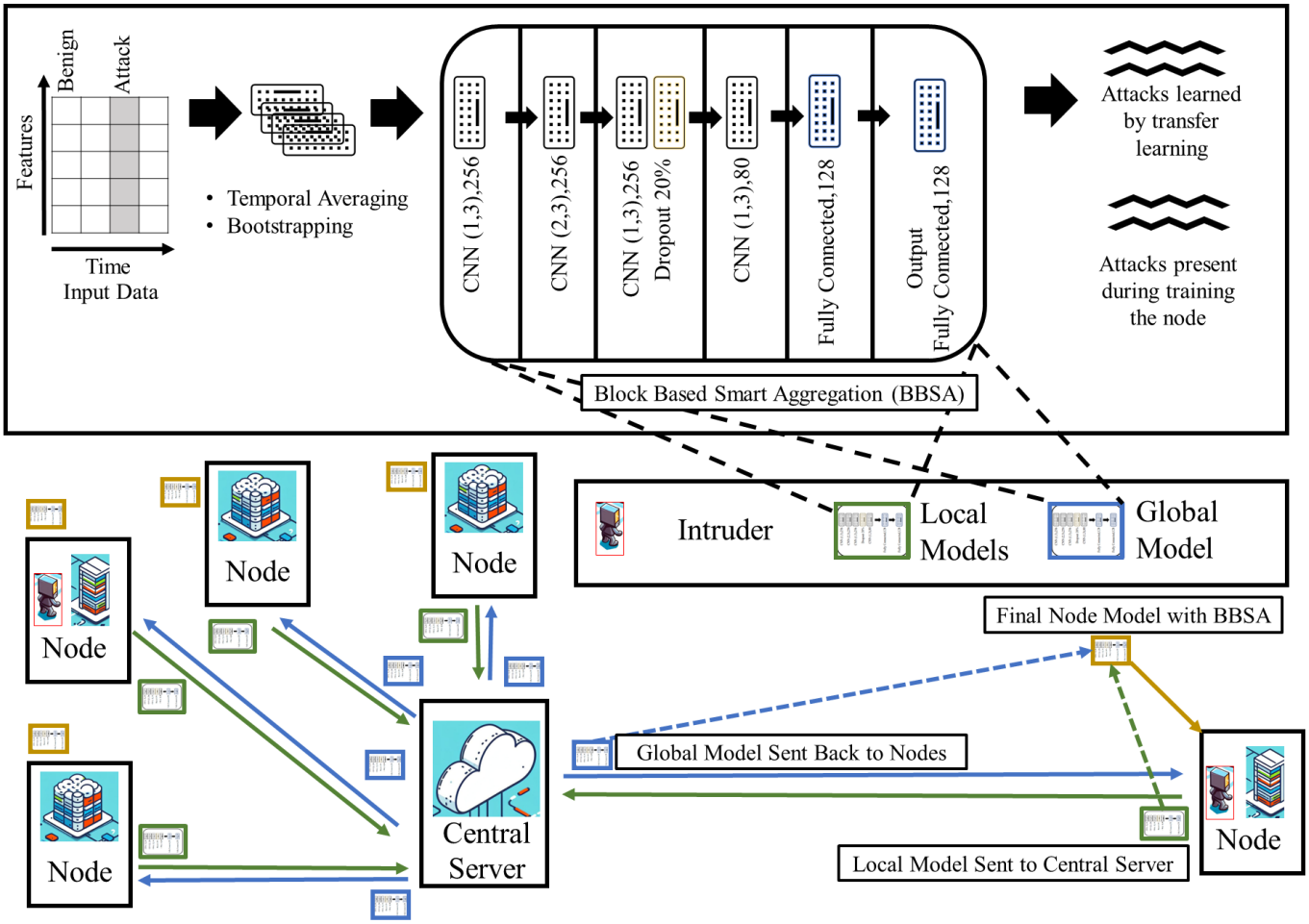

Fig. 1. Architecture of proposed intrusion detection model.

is emerging as a key area of interest [17].

In this paper, we propose a federated deep learning setup aimed at achieving maximum transferability, while preserving in-class IDS performance. The contributions of this work can be outlined as follows:

1) We develop a deep learning model that performs well for both traditional and transferable IDS applications.
2) We introduce two data preprocessing steps before model training-temporal averaging and bootstrapping that boost the transferability performance, and name the resulting algorithm **T**emporally **A**veraged **B**ootstrapped **F**ederated **I**ntrusion **D**etection **S**ystem (TabFIDS) version 1.
3) We further propose a novel Block-Based Smart Aggregation(BBSA) algorithm that works with FedAvg [18] aggregation to make performance driven decisions about layer-wise weight aggregation in every round of training. This setup is called TabFIDSv2. In our experiments, we find that TabFIDSv2 outperforms other methods in both localized detection and transferability, while being generalizable across different neural network backbones, and datasets.
4) We develop a Data-Driven Feature Elimination(DDFE) algorithm that helps us determine which features are most important for model learning.

We note that part of the work included in this paper has been presented in [19]. This paper is organized as follows: In Section II, we explain our system architecture, and the federated learning setup. Next, we present the deep learning model for intrusion detection. We then explain bootstrapping and temporal averaging, the two pre-processing steps to improve the transferability along with our BBSA algorithm. Finally, in Section III, we explain our experimental setup and show that our proposed algorithm significantly improves the transferability in a federated setup, performs well even with different backbones, and shows inter dataset transferability.

II. System Architecture

In this section, we discuss the overall architecture of our federated system environment, as shown in Fig. 1.

A. Nodes

Consider a scenario where several devices with networking capabilities are connected on a network. Each of these devices are referred to as a node. There are attackers within the network injecting harmful data packets with the objective of compromising the security of the nodes. Each node however, is equipped with a deep learning based intrusion detection system

that should be able to identify the malicious data packets. Our objective is to create a framework in which the intrusion detection softwares leverage traffic data at each node to detect existing as well as rare attacks.

B. Intrusion Detection System

The IDS on each node is trained before deployment on labeled benign and attack data. There is a central server that participates in the training of the IDS, and it may have one of two different roles:

1) Centralized Learning: In centralized learning configuration, the data on each node is sent to a central server where a central model is trained using all the aggregated data. The trained model is sent back to each of the local nodes for deployment and detection. Nodes do not perform any kind of localized training based on the data samples of that node.

2) Federated Learning: In a federated learning configuration, each node trains their local models on local data. Each node then sends their locally trained model to the central server for aggregation. The central server uses an aggregation algorithm to create a single model from the localised models called global model. The global model is then sent back to each of the nodes for another round of continued training. In this regime, there is no data being exchanged over the network, rather only the model weights are sent.

In our setup, the central server uses a FedAvg [18] aggregation algorithm to aggregate the model parameters sent by the nodes. The global model is sent back to the nodes for further rounds of localised training and this continues over multiple communication rounds. The local and global models have the same architecture as shown in 1 but differ in their parameters. The global model is aggregated from the local models using the following equation:

$$\forall k, w_{t+1}^k \leftarrow w_t - \eta g_k; w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k \tag{1}$$

where w_t are the model weights after communication round t, n_k is the number of local samples in the training data, n is the total number of data samples across all nodes, assuming we have K nodes in total.

3) Block-Based Smart Aggregation: A notable limitation within the federated averaging methodology is the potential reduction in detection accuracy at the level of individual nodes over successive communication rounds. This challenge arises from the practice of averaging the weights across nodes for updating a model, where a locally optimized set of weights might be more effective for a particular node than the globally aggregated set.

To mitigate this, a strategy is employed wherein nodes, upon receipt of the global model, undertake a phase of re-training using their local data prior to the deployment of the model, incorporating local specificity with broader, aggregated insights. However, some nodes might still encounter a drop in intrusion detection performance. The diversity in data distribution and attack patterns across nodes can result in varying degrees of effectiveness for this approach, with some nodes potentially not achieving the desired level of accuracy improvement in identifying network intrusions.

Algorithm 1 Block-Based Smart Aggregation (BBSA) Technique

Require: N nodes each with local dataset D_i, $i \in \{1, 2, \ldots, N\}$
Ensure: Optimized global model G

```
Initialize global model G with random weights
for each communication round r do
    for each node i in parallel do
        Train local model L_i^r on D_i
        Save local model weights W_{L_i}^r before aggregation
        Send local model weights W_{L_i}^r to server
    end for
    Aggregate weights on server to update global model G^r
    Distribute updated global model G^r to all nodes
    for each node i in parallel do
        Re-train local model L_i^{r+1} on D_i using G^r
        Save re-trained model weights W_{L_i}^{r+1}
        BBSA Step: Compare W_{L_i}^r and W_{L_i}^{r+1}
        for each block b in model do
            Select weights for block b from either W_{L_i}^r or W_{L_i}^{r+1} based on performance
        end for
        Update L_i^{r+1} with selected block weights
        Send updated L_i^{r+1} weights to server for next round
    end for
    Aggregate updated weights on server to update G^{r+1}
end for
```

The proposed Block-Based Smart Aggregation (BBSA) technique addresses the challenge of maintaining or enhancing accuracy across individual nodes during iterative communication rounds in a federated learning setup. This method introduces a strategic layer of decision-making in the model updating process that is tailored to optimize performance on a per-node basis.

The algorithm for BBSA is presented in Algorithm 1. Here, nodes save their local model weights prior to dispatching them to the central server for aggregation. Upon receiving the aggregated global model, nodes engage in a re-training session utilizing their specific local datasets.

Unlike the conventional federated learning process, where this updated model would directly advance to the central server for further aggregation, the BBSA methodology introduces a critical comparative evaluation step. In the proposed selective weight integration process, the decision to utilize weights from either the original local model, as it existed before being sent for aggregation, or the model retrained after receiving the global updates, is made. The primary objective is to harness the strengths of both models—leveraging the specific insights captured by the local model before aggregation and the broader, globally informed updates of the re-trained model.

The methodology for constructing the combined model involves carefully analyzing the deep learning architecture to identify distinct blocks within the model, delineated based on the function of layers that operate collectively within the architecture. The optimization process entails selecting all the weights within a given block to be exclusively sourced either from the original local model prior to aggregation or from the model that has been re-trained post-aggregation. This block-based selection process is critical for several reasons:

1) *Preservation of Functional Integrity:* By treating blocks as the units for weight selection, this approach ensures that the functional integrity of closely interacting layers is preserved. This is crucial because layers within a block often share a contextual relationship, designed to capture specific features or patterns from the input data.
2) *Computational Efficiency:* Working with blocks significantly reduces the computational overhead —instead of making decisions for potentially thousands of individual weights or layers, the process is simplified to a choice between block configurations. This streamlined approach facilitates quicker iterations and optimizations, especially when computational resources and time are at a premium.
3) *Enhanced Generalizability:* By evaluating the effectiveness of entire blocks of layers, the block-based strategy can leverage the collective performance enhancement that these layers provide when their weights are aligned with either the local context (pre-aggregation) or the global learning objective (post-aggregation). This approach allows for a strategic enhancement of the model's ability to generalize across diverse datasets and attack scenarios, potentially leading to better detection capabilities.

In summary, by identifying and optimizing blocks of layers as cohesive units, this strategy ensures the computational tractability of the optimization process, and aims to enhance the model's overall performance by maintaining the functional cohesion of layers that are designed to work together.

C. Deep Learning Model

The proposed model is developed empirically with the aim to not only maximize the classification accuracy, but also to uncover transferability—when the model is trained with one attack and tested on another attack. This model has been tuned to perform deep feature extraction from high dimensional dataset both for well-represented as well as under-represented classes of data. Fig. 1 shows the architecture of this model.

D. Bootstrapping

A characteristic of network traffic data is that the number of benign data packets encountered significantly surpasses the number of attack data packets, which poses challenges for deep learning models. A popular solution to this problem is bootstrapping, where the minority class is re-sampled and appended to the majority class until the dataset is balanced.

E. Temporal Averaging

Temporal averaging, when employed as a pre-processing step, has been shown to boost the transferability of an IDS system [20]. Mathematically this can be given as:

Algorithm 2 Data-Driven Feature Elimination (DDFE)

Require: TabFIDS model, Full 1D input feature vector
Ensure: Reduced feature set model with maintained accuracy
Initial Training:
Train the TabFIDS model on the full 1D input feature vector $\{F : lenght(F) = N\}$
for each feature f_i in the input feature vector **do**
    Create a modified version of the feature vector with f_i set to zero
    Train/evaluate the model with the modified feature vector
    Record the change in classification accuracy
end for
Performance Assessment:
for each feature f_i **do**
    if removal of f_i does not significantly reduce accuracy **then**
        Mark f_i as non-contributory
    end if
end for
Simplification:
Create a reduced feature vector $\{\hat{F}$: *nonzero feature length* $= (N - n)\}$ by setting all n non-contributory features to zero
Deploy TabFIDS model with the reduced feature set

$$y_t = \frac{\sum_{i=0}^{r-1} x(t-i)}{r}, r = \text{window size} \tag{2}$$

where x_t represents the input data sample at time t and y_t represents the corresponding temporally averaged data sample. Thus each input sample to the model is the temporal average of r other samples. Temporal averaging further improves privacy of the data, as the deep learning model is trained with the pre-processed data and it never sees the original data.

F. Data-Driven Feature Elimination (DDFE)

To reduce the computational cost of deep learning systems, optimizing the size of the feature vector is crucial. The Data-Driven Feature Elimination (DDFE) approach is designed to iteratively evaluate the impact of each feature on the model's performance and then eliminate those that contribute minimally. The algorithm for DDFE is presented in Algorithm 2. The DDFE process involves the following steps:

1) *Initial Training:* Train the TabFIDS model on a specific node using the entire 1D input feature vector to establish baseline performance metrics.
2) *Feature Elimination Simulation:*
   - For each feature in the input vector, create a modified version of the input where the current feature's value is set to zero, simulating its removal.
   - Assess the model's performance with the modified input vector to determine the impact of the feature's removal on classification accuracy.

3) *Performance Assessment:* Evaluate the model's performance with the removal of each individual feature, focusing on classification accuracy. Identify features whose elimination does not impose a large performance penalty.
4) *Simplification:*
   - Remove non-contributory features by setting their values to zero in the input vector, resulting in a reduced feature set.
   - Train or fine-tune the model on this optimized feature set to achieve a balance between computational efficiency and detection accuracy.

By employing the DDFE method, TabFIDS can maintain high detection accuracy while operating within computational resource constraints, making it particularly suitable for environments where processing power and memory are limited.

III. Experimental Results

In this section, we first present details about the dataset, experimental setup and the evaluation metric. Then we provide a step by step analysis of the development of the proposed TabFIDSv2, with each step highlighting the impact of individual components of the algorithm. Then we analyze the generalization performance of TabFIDSv2 across different deep learning backends, and across datasets.

A. Dataset

We employ the CIC-IDS 2017 dataset from the Canadian Institute of Cybersecurity [9]. This dataset features 78 features and 14 distinct attack categories. However, the data distribution is very unbalanced: 80% of it is benign, while the remaining 20% consists of various forms of attack data. We ignore three attack classes that suffer from extremely low data availability, together constituting only 0.00232% of total data. Even then, the smallest of the remaining attack categories represents just 0.023% of the total dataset, making it extremely challenging for a data-driven IDS. We use 80% of this data for training, 10% for validation and 10% for testing.

B. Evaluation Metric

Intrusion detection systems typically encounter imbalanced datasets, with significantly more benign data packets than attack data packets. This imbalance poses a unique evaluation challenge, as high overall accuracy may mask poor intrusion detection performance. For instance, in a dataset with 90% benign data, a classifier achieving 90% accuracy might fail to detect any attack packets.

To address this problem, precision and recall metrics are often reported in addition to accuracy, providing a more nuanced understanding of the system's performance. Let tp, tn, fp and fn represent accurately detected attack data points, accurately detected benign data points, wrongly detected attack data points, and wrongly detected benign data points, respectively. The overall accuracy of the system would then be calculated as $(tp+tn)/(tp+tn+fp+fn)$. Precision is $(tp)/(tp+fp)$ and recall would be $(tp)/(tp+fn)$. The recall value provides the ratio of attack data samples that are accurately classified. However, it does not contain information about whether the system can accurately classify benign data samples.

To overcome this issue, we propose to use the attack accuracy, defined as:

$$Accr = \frac{\frac{tn}{tn+fp} + \frac{tp}{tp+fn}}{2} \tag{3}$$

which gives equal weight to correct detection of benign and attack data packets. In this paper, we use the attack accuracy metric to quantify the performance of IDS systems in our severely imbalanced dataset.

C. Transferability in a centralized setup

In the next few subsections, we will gradually introduce individual components of the proposed TabFIDSv2, with accompanying analysis of the design choices.

D. Experimental Setup

In a centralized environment, the model is trained on all available benign data in the training set and one class of attack data. The transferability is then evaluated by testing the model against test data packets from attack classes it has not seen during the training phase. We perform this experiment for all 11 attack classes.

On the other hand, in a federated setup, we divide the training dataset to 11 separate nodes. During training, we expose each node to a single attack type only. There is no overlap between the benign data present in different nodes during training. Each node only sees a $\frac{1}{11}$ fraction of the benign data and one attack class, effectively making local training data set availability much lower compared to a centralized system.

When employing bootstrapping, we sample attack data packets from the same class with replacement until the dataset contains 50% benign data and 50% attack data. We employ the Adam [21] optimizer with a learning rate of 0.001 and the loss function is CrossEntropy Loss. For the federated model, we run 20 rounds of federated model aggregation.

For the purpose of this study, we define transferability as an IDS achieving an attack accuracy $\geq$70% on a test attack class after training on a different attack class. Transferability pairs are denoted by (training attack, test attack) and categorized by performance: very high ($\geq$90%), high (80-90%), or moderate (70-80%) transferability.

E. Centralized vs Federated Transferability

We start our analysis with an investigation of the transferability relationships for the centralized and federated approach. First, in Fig. 2(a), we observe that the centralized model performs reasonably well, identifying 13 transferable pairs. In contrast, the federated approaches yield disappointing results, with only 5 transferable pairs detected, as presented in Fig. 2(b). This is surprising, given that federated learning facilitates information exchange between nodes with diverse training attacks during aggregation, which should enhance transferability. However, the smaller individual training sets at each node,

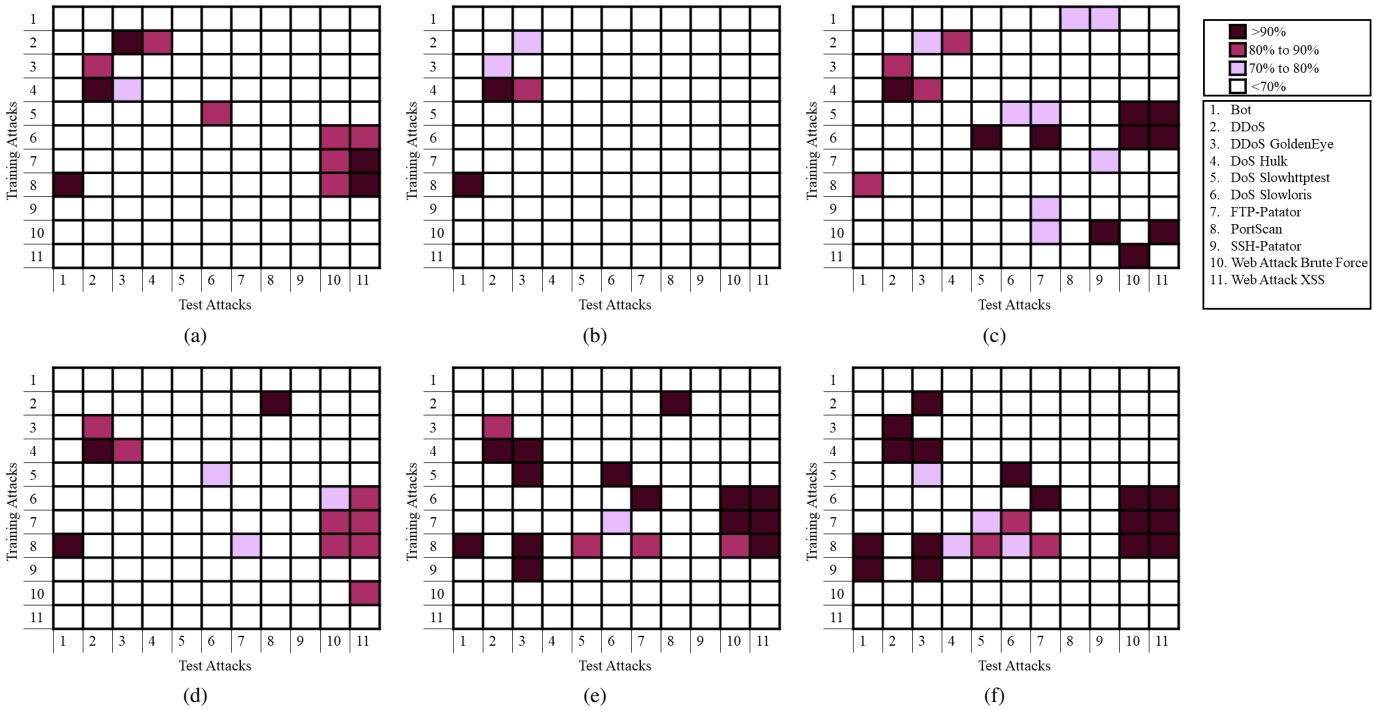

Fig. 2. Transferability Relationships of (Train,Test) Attack Pairs for (a) A Centralized Setup, (b) A Federated Setup, (c) Federated Learning with Bootstrapping, and (d,e,f) Federated Learning with Temporal Averaging (d) Window Length of 3, (e) Window Length of 5, (f) Window Length of 7.

combined with the imbalanced dataset, may contribute to the significant decline in transferability performance.

We then employ bootstrapping as a pre-processing step to mitigate the impact of the imbalanced dataset in the federated setup. We present the results in Fig. 2(c). This augmentation leads to a substantial improvement in performance, with the detection of 22 transferable pairs, a notable increase from the 5 pairs detected without bootstrapping.

F. Temporal Averaging

We investigate the effect of temporal averaging on transferability, with the results in Fig. 2(d) demonstrating a remarkable improvement over the federated setup, with 14 transferable pairs detected compared to 5. Here we average three data packets and feed them to the input layer. This enhancement is noteworthy, as it demonstrates that temporal averaging can boost deep learning model performance even in the presence of severe dataset imbalance, without relying on data augmentation. This can be attributed to the ability of temporal averaging to aggregate multiple input data packets, providing the IDS with a more comprehensive understanding of the network environment. This approach is particularly valuable in resource-constrained training scenarios, where data augmentation can substantially increase training time.

1) Window Length for Temporal Averaging: One question arising from the previous discovery is the impact of the number of samples that are averaged before the input layer. We define this as the window length of temporal averaging. In Fig. 2(e) and Fig. 2(f), we present the results for window lengths of 5 and 7, respectively. The results illustrate that there is a correlation between the window length and the number of transferable pairs. A window length of five results in an increase to 19 transferable pairs, and a window length of seven gives us 23 transferable pairs, which is higher than the bootstrapping approach. Thus with a simple pre-processing operation we can achieve comparable trasferability to bootstrapping while avoiding the massive increase in training dataset size. The tradeoff here is the introduction of a latency even during deployment in a live system. A bigger window requires more input data packets, resulting in a higher latency penalty.

G. TabFIDS

Next, we combine bootstrapping and temporal averaging to create the Temporally Averaged Bootstrapped Federated Intrusion Detection System (TabFIDS). This approach yields the most impressive transferability results yet, achieving transferability for 31 pairs, as seen in Fig. 3(a). To minimize the latency penalty we use a temporal averaging window size of three. However if better performance is desired and a small increase in processing time is not an issue, we can deploy the same system with a larger temporal averaging window. As shown in Fig. 3(a) and later summarized in Table II, TabFIDS not only excels in transferability but also achieves the highest number of pairs so far with attack accuracy above 90% (17 pairs), outperforming the next best approach (bootstrapping only) (10 pairs). Furthermore, TabFIDS demonstrates exceptional performance in localized intrusion detection, attaining an average accuracy of 97.18% across nodes when tested with the same attack class used during training, showcasing its robustness and effectiveness.

Notably, when combining temporal averaging and bootstrapping in TabFIDS, we observe a trade-off, where 5 transferabil-

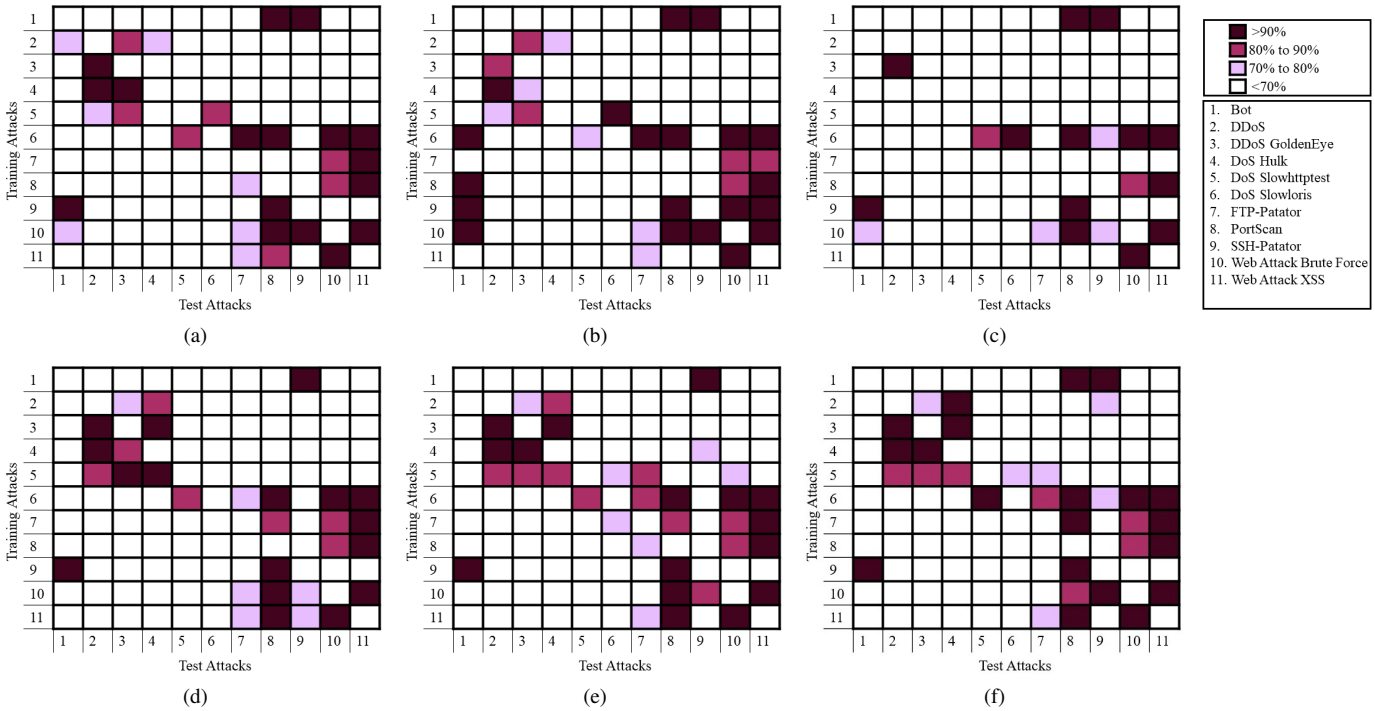

Fig. 3. Transferability Relationships of (Train,Test) Attack Pairs for (a) TabFIDSv1 - Federated Setup with Bootstrapping and Temporal Averaging, (b) Centralized Setup with Bootstrapping and Temporal Averaging, (c) TabFIDSv1 with Data-Driven Feature Elimination, and (d,e,f) TabFIDSv2 - Block-Based Smart Aggregation combined with TabFIDSv1, with a Train-Validation-Test Split of (d) 80-10-10, (e) 50-10-10, and (f) 25-10-10.

ity pairs detectable by standalone bootstrapping and 2 pairs from standalone temporal averaging approach are no longer detectable. This suggests that refining the integration method could further enhance TabFIDS.

Another finding that is worth investigating is the impact of the TabFIDS pipeline on a centralized setup, illustrated in Fig. 3(b). The integration of bootstrapping and temporal averaging in a centralized setup also results in improvements in performance, resulting in 32 transferable pairs. This highlights the versatility and broad applicability of the proposed approach.

Impact of DDFE

Next, we implement the DDFE algorithm mentioned in Algorithm 2, along with TabFIDS. The results are presented in Fig. 3(c). DDFE's ability to judiciously reduce the feature set by 39.34% on average—ranging from a minimal 8.97% reduction for specific attack types to as much as 74.36% for others—underscores its effectiveness. DDFE retains 19 transferable pairs (38.7% reduction in transferable pairs), with 13 pairs achieving high transferability rates (>90%). This demonstrates DDFE's effective feature selection, preserving the model's ability to generalize across unseen attacks. We observe that the introduction of DDFE decreases transferability performance the most for pairs which previously had moderate transferability. DDFE emerges as a valuable tool in resource-constrained setups, where balancing speed and performance is crucial, and a reduction in transferability can be tolerated with a comparable reduction in computational complexity.

H. TabFIDS v2: Incorporating TabFIDS and BBSA

In Fig. 3(d), we present the transferable pairs achieved by TabFIDSv2, an enhanced version of TabFIDS incorporating block-based smart aggregation. Interestingly, the introduction of the BBSA algorithm results in a reduction in the number of transferable pairs, from 31 to 30. However for localized intrusion detection (training and testings sets from the same attack class), performance improves from 97.18% to 99.64%.

I. Fine Tuning TabFIDSv2: Size of Training Dataset

We cannot directly train for transferability, as the unknown attack classes are not available during training. Moreover, the BBSA algorithm focuses on optimizing localized intrusion detection through recursive fine-tuning of neural architecture blocks, and this may compromise transferability. We suspect overfitting to training classes causes this issue.

To test this, we reduce the size of the available training dataset. Up to the previous section, the train-validation-test split was 80-10-10. In this experiment, we only use 50% of the available data for training, instead of 80%. For the purpose of consistency, the validation and test sets remain unchanged, with 10% of the data in each. The remaining 30% of data is not used. The results are presented in Fig. 3(e).

Here, we observe an interesting trend. Reducing the size of the training set improves transferability (we get 34 transferable pairs compared to 30 with a 80-10-10 split) while having a very small decrease in localized intrusion detection performance (from 99.64% to 99.45%). This shows that a reduction in the training set has an advantageous impact by making the training more transferable.

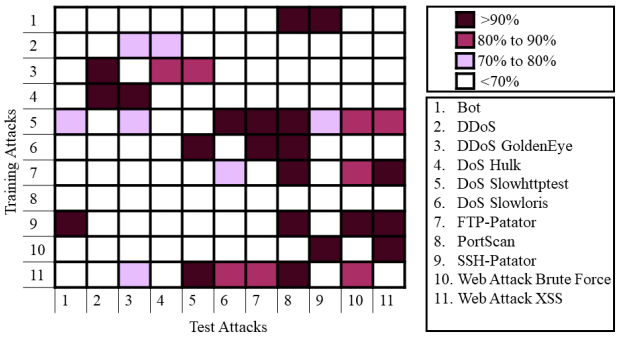

Fig. 4. Transferability Relationships for TabFIDSv2 with a Train-Validation-Test split of 05-10-10.

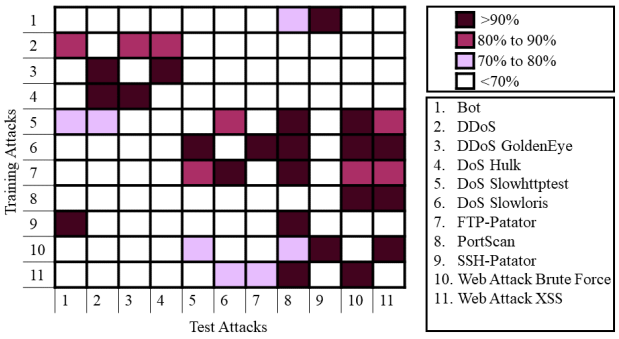

Fig. 5. Transferability Relationships for TabFIDSv2 with Temporal Averaging Window of Length 5 (25-10-10 Split).

Next, we test a 25-10-10 train-validation-test split, with the results presented in Fig. 3(f). We get 33 transferable pairs. While the total number of transferable pairs is one less than the 50-10-10 case, on a closer look we notice that here 20 pairs showcase a transferablity performance higher than 90% (dark purple), compared to 16 pairs in the previous case. In the 25-10-10 case, localized intrusion detection performance decreases slightly, to 98.09%. In our opinion, This presents a good trade off between strong transferability and good localized performance.

Finally, we also perform tests using a 5-10-10 split, where only 5% of data is used for training, and the results in Fig. 4 show 36 transferable pairs, the highest so far. However, due to the significant reduction in training data size, localized intrusion detection performance suffers and falls to 95.90%.

The notable insight from this analysis is the finding that the block-based aggregation algorithm in TabFIDSv2 achieves better transferability when aided with a reduction in the training set size. We also note that in addition to facilitating robust and resilient intrusion detection, this reduction in the training set size results in significantly faster training times.

1) Even Higher Transferability: Window Size: Next, we focus on extending the analysis on the impact of temporal window size on TabFIDSv2. In Figs. 5 and 6, we present the impact of temporal windows of sizes 5 and 7 on a 25-10-10 training case. We note that with a temporal window size of 3 in Fig. 3(f), we got 33 transferable pairs with a localized intrusion detection of 98.09%. For a temporal

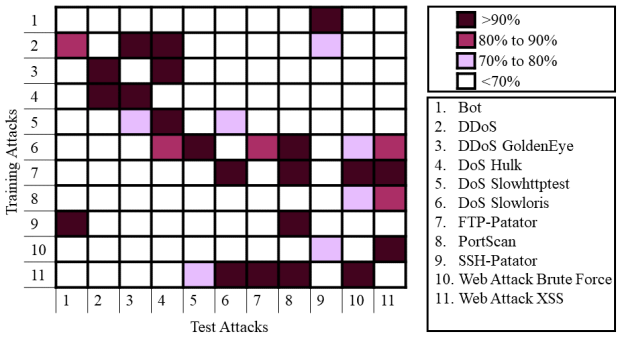

Fig. 6. Transferability Relationships for TabFIDSv2 with Temporal Averaging Window of Length 7 (25-10-10 Split).

TABLE I
NUMBER OF OCCURRENCES FOR AN ATTACK CLASS IN TRANSFERABLE (TRAIN ATTACK, TEST ATTACK) PAIRS (TABFIDSV2, 25-10-10 SPLIT, TEMPORAL AVERAGING WINDOWS OF LENGTH 5)

Attack Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Present as Train Attack	2	3	2	2	6	5	5	2	2	4	4
Present as Test Attack	3	3	2	2	3	3	2	7	2	5	5

window size of 5, we get 37 transferable pairs (22 with $> 90\%$), which is the highest we have achieved. The localized intrusion detection rate also improves to 98.36%. Interestingly, when we move to a temporal window of 7, the localized intrusion detection rate further improves to 98.82%, but the number of transferable pairs falls to 33 (20 with $> 90\%$). This indicates that there is also an optimal temporal window size for maximum transferablity.

In Table I, we present an analysis of the (train,test) pairs present in the best performing experiment (TabFIDSv2, 25-10-10 train-validation-test split, temporal windows of size 5). We note that every attack is present in at least two (train,test) pairs, with the DoS Slowhttptest attack showing the highest transferability in detecting other attacks, being present in 6 pairs as train attack. On the other hand, the PortScan attack is the easiest attack to be detected by transferable learning, being present as test attack in 7 pairs.

J. Generalizability of TabFIDSv2: Using a Different Back-end

TabFIDSv2, as presented above, consists of several components - the federated system architecture, the neural network architecture, temporal averaging, bootstrapping, block-based smart aggregation, and optimal training set selection. So far we have analyzed the individual impacts of all of these components except the neural network architecture. In this section, we analyze the impact of the neural network architecture by replacing the back-end with two different architectures - Resnet and Autoencoder.

1) Resnet: First, we use a resnet architecture presented in Fig. 8, similar to the one in [22]. This model consists of 3 blocks. The first block, consisting of linear layers takes the input and resizes it to fit the inputs of the resnet backbone. The resnet backbone is a pretrained 18-layer model, followed by the last block which takes the output from the resnet backbone and classifies it.

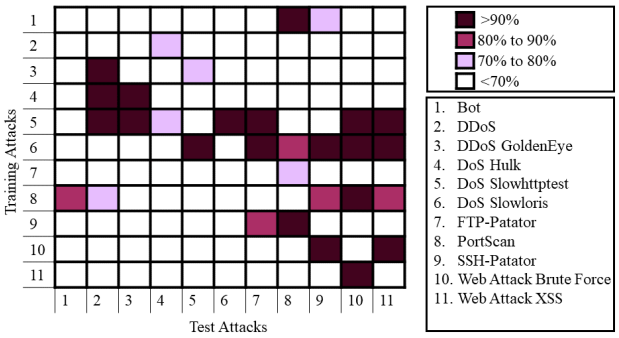

Fig. 7. TabFIDSv2 Transferability with a ResNet Backend (25-10-10 Split).

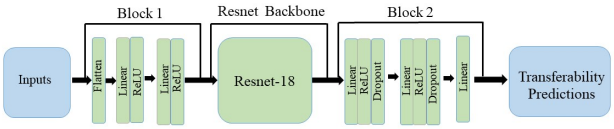

Fig. 8. Architecture of the ResNet Backend.

The results from this analysis are presented in Fig. 7. Here we use a 25-10-10 train-validation-test split. As we can see from the analysis, even with the resnet backend, we find 31 transferable pairs, which shows the generalizability of the proposed approach. This setup achieves a localized intrusion detection rate of 98.91%.

2) Autoencoder: We also utilize an autoencoder based approach, with the architecture—similar to the approach in [23]—presented in Fig. 10. The autoencoder network consists of an encoder and a decoder. The encoder, using convolutional layers, captures the important features of the network and learns the lower dimensional representation of the input. The decoder consists of convolutional, upsample and transpose convolution layers to reconstruct the original input from the encoded lower dimensional representation.

The results from this approach are presented in Fig. 9, with a 25-10-10 split. We get 24 transferable pairs. One interesting point to note here is that 5 training attacks and 2 test attacks are absent from any transferable pairs. In the original TabFIDSv2 or the Resnet backend, every train or test attack

TABLE II
NUMBER OF TRANSFERABLE PAIRS FOR DIFFERENT APPROACHES AND THEIR ATTACK ACCURACY

	Total	>90%	80% -90%	70% -80%	Localized Accr %
Central	13	5	7	1	80.09
Federated	5	2	1	2	74.27
Fed+BStrap	22	10	4	8	98.18
Fed+TempAv(3)	14	3	8	3	80.91
TabFIDSv1(80/10/10)	31	17	7	7	97.18
TabFIDSv2(25/10/10)	33	20	7	6	98.09
TabFIDSv2(25-TempAv 5)	**37**	22	8	7	98.36
TabFIDSv2(ResNet)	31	20	5	6	98.91
TabFIDSv2(AutoEnc)	24	11	8	5	75.09

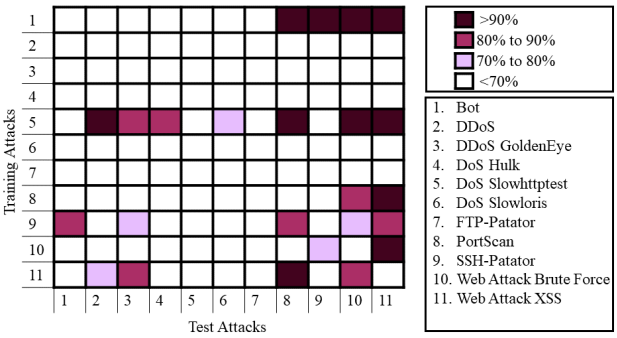

Fig. 9. Transferability Pairs Detected by TabFIDSv2 with a Autoencoder Backend (25-10-10 Train-Validation-Test Split).

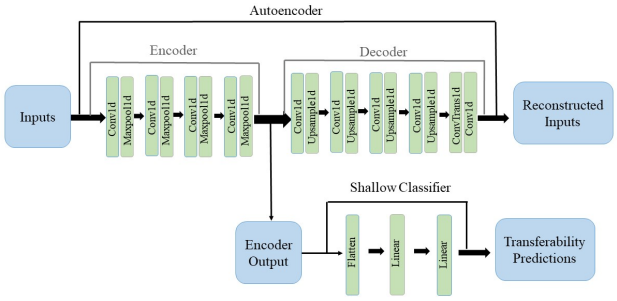

Fig. 10. Architecture of the Autoencoder Backend.

was represented in multiple (train,test) pairs. This indicates the importance of choosing a backend carefully.

Finally, in Table I, we summarize the notable experimental results. Here, we note the total number of transferable pairs achieved by each approach, as well as a breakdown of very high (≥90%), high (80-90%), and moderate (70-80%) transferability. We also include the localized intrusion detection rates for each of these approaches. We note that both TabFIDS and TabFIDSv2 achieve excellent transferability while maintaining very high localized intrusion detection rates. The combination of TabFIDSv2, with a 25-10-10 train-validation-test split, and a temporal windows of size 5 achieves the best transferability. While the Resnet backend performs pretty well, the autoencoder backend underperforms in both transferability and localized intrusion detection.

K. TabFIDSv2 with a different dataset

To further investigate the generalizability of TabFIDSv2, we replace the CIC-IDS 2017 dataset with the CIC-ToN_IoT dataset [10]. This analysis aims to show that the versatility of the proposed approach is not dependent on a specific dataset. The CIC-ToN_IoT dataset consists of benign data along with 8 different types of attacks-XSS, Injection, Password, Scanning, MITM, DDoS, DoS, and Backdoor.

For this experiment, the non-numeric features are removed from the dataset. Similar to the approach in [15], we restrict our analysis to the data samples corresponding to the top 10 destination IP addresses. This data is then distributed among 8 nodes. The benign data packets are divided equally to each of the 8 nodes. Each node gets only one class of attack data. This distribution is similar to the the one followed earlier for

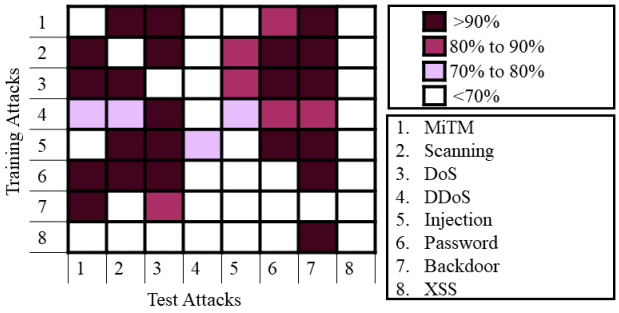

Fig. 11. Transferability Relationships with Tabfidsv2 on the Ton-IoT Dataset.

CIC-IDS 2017. The results for TabFIDSv2 on this dataset are presented in Fig. 11. The train, validation and test split for this experiment is 80-10-10. Here, we notice transferability in 32 pairs, out of a possible 56 pairs. Out of these 32 pairs, 22 pairs show transferability with >90%. This performance validates the generalizability of our approach in uncovering transferability across a wide variety of attacks.

IV. Conclusion

In this paper, we develop TabFIDSv2, a novel intrusion detection system designed to optimize transferability in federated learning environments. By integrating two pre-processing techniques—bootstrapping and temporal averaging—with our empirically developed deep neural network, our proposed TabFIDSv1 algorithm achieves exceptional localized detection performance and high transferability across significantly imbalanced datasets. We also evaluate a Data-Driven Feature Elimination algorithm that presents a reasonable trade-off between transferability and processing cost. The integration of a Block-Based Smart Aggregation algorithm, resulting in TabFIDSv2, further enhances transferability performance and localized intrusion detection accuracy. Comprehensive evaluations demonstrate the system's generalizability across diverse neural network backbones, and different datasets. Although TabFIDSv2 is developed with a focus on intrusion detection systems, this development can be of value in other deep learning domains where transferability is a desired attribute.

References

[1] Lansky, Jan and Ali, Saqib and Mohammadi, Mokhtar and Majeed, Mohammed Kamal and Karim, Sarkhel H Taher and Rashidi, Shima and Hosseinzadeh, Mehdi and Rahmani, Amir Masoud, "Deep learning-based intrusion detection systems: a systematic review," *IEEE Access*, vol. 9, pp. 101 574–101 599, 2021.

[2] Saleh, Hadeel M. and Marouane, Hend and Fakhfakh, Ahmed, "Stochastic Gradient Descent Intrusions Detection for Wireless Sensor Network Attack Detection System Using Machine Learning," *IEEE Access*, vol. 12, no. , pp. 3825–3836, 2024.

[3] Lin, Zong-Zhi and Pike, Thomas D. and Bailey, Mark M. and Bastian, Nathaniel D., "A Hypergraph-Based Machine Learning Ensemble Network Intrusion Detection System," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–13, 2024.

[4] Bakro, Mhamad and Kumar, Rakesh Ranjan and Alabrah, Amerah and Ashraf, Zubair and Ahmed, Md Nadeem and Shameem, Mohammad and Abdelsalam, Ahmed, "An Improved Design for a Cloud Intrusion Detection System Using Hybrid Features Selection Approach With ML Classifier," *IEEE Access*, vol. 11, no. , pp. 64 228–64 247, 2023.

[5] Barnard, Pieter and Marchetti, Nicola and DaSilva, Luiz A, "Robust Network Intrusion Detection through Explainable Artificial Intelligence (XAI)," *IEEE Networking Letters*, vol. 4, no. 3, pp. 167–171, 2022.

[6] Fu, Yanfang and Du, Yishuai and Cao, Zijian and Li, Qiang and Xiang, Wei, "A Deep Learning Model for Network Intrusion Detection with Imbalanced Data," *Electronics*, vol. 11, no. 6, p. 898, 2022.

[7] Koroniotis, Nickolaos and Moustafa, Nour and Sitnikova, Elena and Turnbull, Benjamin, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.

[8] Tavallaee, Mahbod and Bagheri, Ebrahim and Lu, Wei and Ghorbani, Ali A., "A detailed analysis of the KDD CUP 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[9] Sharafaldin, Iman and Lashkari, Arash Habibi and Ghorbani, Ali A, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." in *ICISSp*, 2018, pp. 108–116.

[10] Moustafa, Nour, "A new distributed architecture for evaluating AI-based security systems at the edge: Network TON-IoT datasets," *Sustainable Cities and Society*, vol. 72, p. 102994, 2021.

[11] Verkerken, Miel and D'hooge, Laurens and Wauters, Tim and Volckaert, Bruno and De Turck, Filip, "Towards model generalization for intrusion detection: Unsupervised machine learning techniques," *Journal of Network and Systems Management*, vol. 30, no. 1, pp. 1–25, 2022.

[12] Catillo, Marta and Del Vecchio, Andrea and Pecchia, Antonio and Villano, Umberto, "Transferability of machine learning models learned from public intrusion detection datasets: the CICIDS2017 case study," *Software Quality Journal*, pp. 1–27, 2022.

[13] Rizvi, Syed and Scanlon, Mark and McGibney, Jimmy and Sheppard, John, "Deep learning based network intrusion detection system for resource-constrained environments," in *Springer*, 2023, pp. 1–7.

[14] Rahman, Sawsan Abdul and Tout, Hanine and Talhi, Chamseddine and Mourad, Azzam, "Internet of things intrusion detection: Centralized, on-device, or federated learning?" *IEEE Network*, vol. 34, no. 6, pp. 310–317, 2020.

[15] Ruzafa-Alcazar, Pedro and Fernandez-Saura, Pablo and Marmol-Campos, Enrique and Gonzalez-Vidal, Aurora and Hernandez-Ramos, Jose L and Bernal-Bernabe, Jorge and Skarmeta, Antonio F, "Intrusion detection based on privacy-preserving federated learning for the industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1145–1154, 2021.

[16] Cui, Jie and Sun, Hu and Zhong, Hong and Zhang, Jing and Wei, Lu and Bolodurina, Irina and He, Debiao, "Collaborative Intrusion Detection System for SDVN: A Fairness Federated Deep Learning Approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 9, pp. 2512–2528, 2023.

[17] Zhang, Jiazhen and Luo, Chunbo and Carpenter, Marcus and Min, Geyong, "Federated Learning for Distributed IIoT Intrusion Detection using Transfer Approaches," *IEEE Transactions on Industrial Informatics*, 2022.

[18] H. Brendan McMahan and Eider Moore and Daniel Ramage and Seth Hampson and Blaise Agüera y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," 2023. [Online]. Available: {https://arxiv.org/abs/1602.05629}

[19] Ghosh, Shreya and Jameel, Abu Shafin Mohammad Mahdee and Gamal, Aly El, "Improving Transferability of Network Intrusion Detection in a Federated Learning Setup," in *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*. IEEE, 2024, pp. 171–176.

[20] Ghosh, Shreya and Mahdee Jameel, Abu Shafin Mohammad and El Gamal, Aly, "An Analysis of Transferability in Network Intrusion Detection using Distributed Deep Learning," in *The First Tiny Papers Track at ICLR 2023*. Tiny Papers @ ICLR 2023, 2023.

[21] Diederik P. Kingma and Jimmy Ba, "Adam: A Method for Stochastic Optimization," *CoRR*, vol. abs/1412.6980, 2014.

[22] Lim, Hyun-Kyo and Kim, Ju-Bong and Heo, Joo-Seong and Kim, Kwihoon and Hong, Yong-Geun and Han, Youn-Hee, "Packet-based Network Traffic Classification Using Deep Learning," in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2019, pp. 046–051.

[23] Kamalov, Firuz and Zgheib, Rita and Leung, Ho Hon and Al-Gindy, Ahmed and Moussa, Sherif, "Autoencoder-based Intrusion Detection System," in *2021 International Conference on Engineering and Emerging Technologies (ICEET)*, 2021, pp. 1–5.