Intrusion Detection Using Federated Learning

Dhileeban Kumaresan
dkumares@berkeley.edu

Jocelyn Lu
jocelyn.r.lu@berkeley.edu

Nitin Pillai
nitin.pillai@berkeley.edu

Riyaz Kasmani
rkasmani@berkeley.edu

Abstract

Intrusion detection is a huge domain that needs to balance a large number of sometimes conflicting requirements such as accuracy, speed, and privacy. While centralized models built on the aggregation of all available network data can be robust at detecting attacks, privacy concerns can make the sharing and distribution of this data from individual entities impossible. Classical decentralized modeling can address privacy concerns, but may be inadequate for the network traffic use case, where each client will likely produce data that is not identically distributed with other clients (i.e.: non-IID data) (7) (5).

In this paper, we propose using a federated learning framework, implementing the federated averaging algorithm by aggregating locally-computed weights. We show that this approach both preserves the privacy of individual clients and is also robust to unbalanced data distributions in this domain, maintaining high model fidelity.

1 Introduction

Cybersecurity has always been a field of cat-and-mouse games: defenders and attackers are constantly trying to stay one step ahead of the other in an effort to further their objectives, malicious or not. In the 21st century, there is no more a concept of a network perimeter with organizations allowing Bring Your Own Devices (BYOD), or hosting sensitive data on the cloud, which can be accessed from an authenticated computer or mobile device anywhere in the world. This immensely increases the attack surface, and to catch hackers in this domain, host and network intrusion detection using machine learning to identify anomalous behavior has proven to be very effective. In a tight race like in this domain, practitioners must use all the tools at their disposal, and at the core of network intrusion detection, that tool is data.

In general, data volume is a necessary but not sufficient aspect in training well-performing models. In the case of intrusion detection, we require another facet of data: breadth. In particular, we cannot rely on data from a single machine or server to provide the best predictions as the type of attacks can vary quite drastically between different organizations, attackers, and time periods. Only by combining many of these examples together can we provide useful insights into attacks. However, individuals can be rightfully wary of providing this data to a central server carte-blanche for analysis due to privacy and data sensitivity concerns, even when data is anonymized.

In this project, we embrace rather than try to fight this type of data segregation. Instead of relying on a centralized data and server architecture, we implement a federated learning model that allows users to collectively benefit from a shared model by providing local model weights, rather than data, to a centralized aggregator, which can calculate global model information to pass back to each server.

The foundation of this process is the federated averaging algorithm incepted by McMahan et al. (4). We use four NVIDIA Jetson Xavier NX devices as our servers, which will hold data private to each machine, and a cloud AWS EC2 instance as a centralized aggregator, henceforth called the

coordinator, for model aggregation. Model updates are communicated between the coordinator and the training servers using Paho MQTT as the message broker. A detailed architecture of this process is provided in Section 3.2. In this fashion, we can continue to iterate through model improvements using populational data across multiple machines instead of being limited by local data, even when the local data is not evenly distributed. Furthermore, because no data is being sent across the network, we are still able to maintain data privacy for all users. These two points are the cornerstones of the federated learning approach.

2 Background

2.1 Data and Use Case

An intrusion detection system is a device or software application that monitors a network or systems for malicious activity or policy violations. This space is constantly evolving due to the increasing nature of novel attacks and is naturally suited for incremental machine learning. Building a machine learning model for intrusion detection requires access to internal data with real attacks, but such data cannot be shared due to privacy concerns, and if shared, generally is anonymized and may not reflect the current trends. To overcome these shortcomings, we explore the federated learning technique on the CSE-CIS-IDS 2018 dataset.

This dataset is generated by the Communications Security Establishment (CSE) and the Canadian Institute for Cybersecurity (CIC) (6) over the course of ten days. They apply a systematic approach to generate a comprehensive benchmark dataset for network intrusion detection by creating different user profiles, each with different abstract representations of events and behaviours. These profiles spread out over 50 machines are then used to carry out sample attacks to a victim network of 30 servers and 420 machines.

Fourteen different types of malicious network traffic make up 17% of the total network traffic, with the remaining being benign events. The malicious attacks include attacks such as denial of service (DoS), distributed denial of service (DDoS), bot attacks, brute force, web attacks, and internal infiltration, with a total of 16.2 million samples including benign events. Relevant features include options like packet length, flow duration, active and idle session time, and protocol.

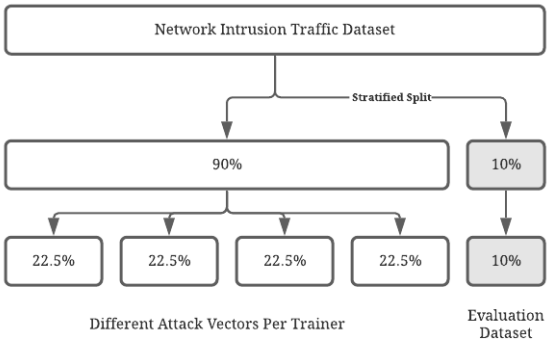

Figure 1: Visualization of the breakdown of the training and tests sets from the CSE-CIS dataset. As an example, `Training Dataset 1` would hold all events in the `DDoS attacks-LOIC-HTTP`, `Infilteration`, `Brute Force -XSS` classes, `Training Dataset 2` would hold all events in the `DoS attacks-Hulk`, `FTP-BruteForce`, `DDOS attack-LOIC-UDP` classes, etc.

For our approach, we want to lean into the skewed distribution of the data across our learners to showcase the generalizability of our federated learning algorithm and provide the model a scenario that is more likely to reflect the real world. After preprocessing steps to remove unnecessary columns and dropping events with empty or infinity values, we apply a specific data sampling strategy. First, we take a 10% stratified sample to serve as our test or evaluation set. However, instead of stratifying across each label to generate our training data, we split up the fourteen different attack vectors into four datasets, one for each server trainer, such that all malicious attacks of a single type live only in a single dataset. We then split the benign class events across all four training datasets. The training data is finally bootstrap sampled to boost the number of events for some of the minority classes, as many of the attack classes make up less than 1% of all data. Refer to Figure 1 for an example.

Algorithm 1 Federated Averaging Algorithm. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Ensure: Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  end for
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
end for

Ensure: ClientUpdate(k, w)**:** *// Run on client* k
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
  end for
end for
return w to server

The real world analogy for this data segmentation strategy is as follows. Foo Inc. may have recently suffered a DDoS attack, which we want to use to learn more about this family of malicious traffic to provide insights to other companies such as Bar Corp. However, Bar Corp. by itself, without access to Foo Inc.'s data, will not have the capability to build a model to detect this style of attack. We split the data in this fashion to show that our approach is robust enough to handle this style of decentralized information without the need to share data directly across our trainers.

2.2 Federated Averaging Algorithm

With our data segmentation strategy, we can define each client K each with their own dataset. The federated averaging algorithm then is as follows in Algorithm 1.

Intuitively, we combine the local stochastic gradient descent ∇ on each client C with a server S_t that performs model averaging. Each client locally takes one step (or, alternatively to add more computation, multiple iterations) of the gradient descent on the current local model trained on its privately held data. Afterwards, the server then takes a weighted average of the resulting models.

3 Methodology

3.1 DNN Model Architecture

To highlight the functionality of the federated network, we start with a simple base model, using a multi-layer perceptron with one hidden layer of size 256 (Figure 2).

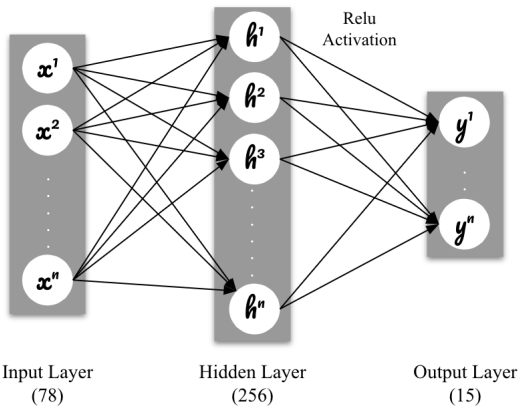

Figure 2: Architecture of the simple MLP network.

The 78 input layers correspond to the 78 features we use from the IDS 2018 dataset, and the 15 output layers align with the 14 malicious and single benign output classes.

We make a strong note here that while previous work on this dataset often reduces the number of labels to just two, a benign class and a malicious class (for instance, Ahmad et al. (1) and Kanimozhi et al. (3)), we keep the large number of classes to showcase the non-conformity of data across our four training samples. This is a conscious trade-off to proving that the network is capable of handling more complicated data at the expense of higher accuracy from an easier classification task.

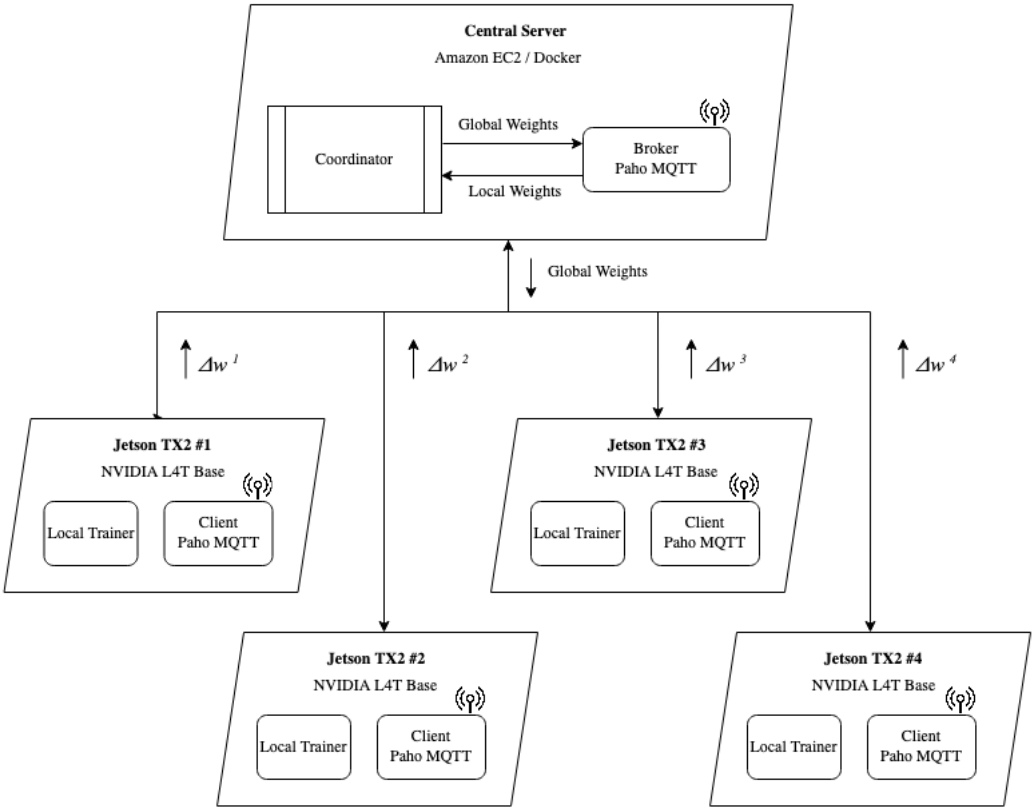

Figure 3: Figure of federated learning architecture, with one coordinator running on Amazon EC2 and four NVIDIA Jetson NX machines as client trainers.

Another note we make is that within our 78 input features, the timestamp is not one of them. Previous work has also used the timestamp changed to epoch time (2). However, we found that the timestamp correlates heavily to the attack type solely due to the way the IDS data is generated: in particular, attack vectors in a single category tend to be created during the same day, likely for ease of execution for the creators. It is our opinion that while this is a highly predictive feature for this dataset, it is not actually reflective of production situations, and we choose to leave the feature out of our model.

3.2 Federated Learning Architecture

Our federated learning architecture is composed of a central server, hosted in AWS on an EC2 instance, along with four client servers hosted locally to train local versions of our network intrusion detection model. Our client trainers are four NVIDIA Jetson Xavier NX's, each of them holding one private dataset as explained in the previous section. On the AWS EC2 instance, we have implemented the coordinator, which aggregates the model weights from each of the trainers. A Paho MQTT broker is created on the cloud to which the coordinator and all the trainers are subscribed to. The weights are then passed between the coordinator and the trainers (and vice versa) for each iteration of the training run. A detailed depiction of the architecture is provided in Figure 3.

McMahan et al. (4) mentions the advantages of starting multiple models from common initialization and training each independently on different subset of data. As found in the paper, common initialization is much more well behaved compared to relying on each model having different random initializations.

For the scope of this experiment, we implement federated learning in a synchronized setting. All four trainers are started and wait for the initial

weights from the coordinator. The coordinator initializes a common model, which could be a pre-trained model, and broadcasts the initial weights. The trainers subscribed to the MQTT broker receive the initial model and use it to train further for a specified number of local epochs on their private datasets. At the end of the specified number of local epochs, the updated weights are published to the MQTT broker to which the coordinator is subscribed to. The coordinator upon receiving the updated weights from each of the trainers, averages these weights. This round trip is considered one global epoch. The aggregated weights are then sent back to the trainers for further training and this process continues for the specified number of global epochs.

While there are existing libraries to facilitate federated learning approaches, such as Tensor Flow Federated or PySyft, we opt instead to create our network from scratch. The reason for this is two-fold. First, we want to apply the principles of federated learning manually, as this gives us more control over our implementation and fosters better understanding of the problem. Secondly, if we were to use some of these packages, we would not be able to leverage the NVIDIA Jetson machines to their full potential as GPU support is not available for many of these solutions.

4 Empirical Results

4.1 Single Node Model

We start our baseline with a model run on a single node in Amazon EC2. As the structure of a single node model does not need the same data split defined in Section 2.1, we simply keep the structure of the data the same but merge all events into a single training dataset.

The accuracy and loss graphs for this single node model are provided in Figure 4. We quickly note here that we see a fairly standard loss curve for a single model. Here, we can set our baseline, showing the average accuracy of the model at around 85%.

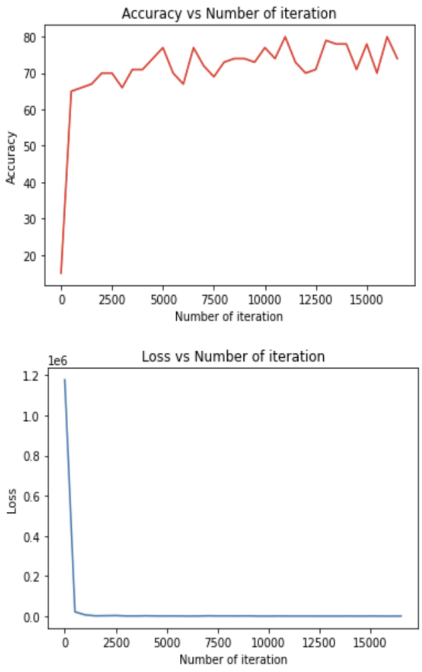

Figure 4: Accuracy and loss graphs for a single node MLP model.

4.2 Federated Averaging Model With Four Trainers

We use the same MLP model architecture from the single node model described in Figure 2 on each of our four Jetson trainers. We initiate the MQTT broker on AWS and wait for all four trainers to connect to receive messages. Once all clients are connected, the coordinator passes the initialized common weights to each trainer and begins the training process. We run the training for five global epochs coordinated by the aggregator, with each trainer running two local epochs at 1800 iterations each within each global epoch. At the end of each global epoch, the trainer sends its final local weights to the coordinator, where the global weights are calculated and passed back to the trainers, in which the cycle continues. A short video demo of this process using a stub dataset can be found here.

As each trainer houses its own unique set of privately held data, we expect different training behaviors as we iterate through the model. In fact, we do see this materialize in Table 1, where each trainer starts with a different local accuracy and

Client Server	Initial Local Accuracy	Final Local Accuracy
Trainer 1	18%	97.5%
Trainer 2	3%	80.5%
Trainer 3	5%	97.5%
Trainer 4	1%	96.0%

Table 1: Final local trainer accuracies after five global epochs, with two local epochs at 1800 iterations each.

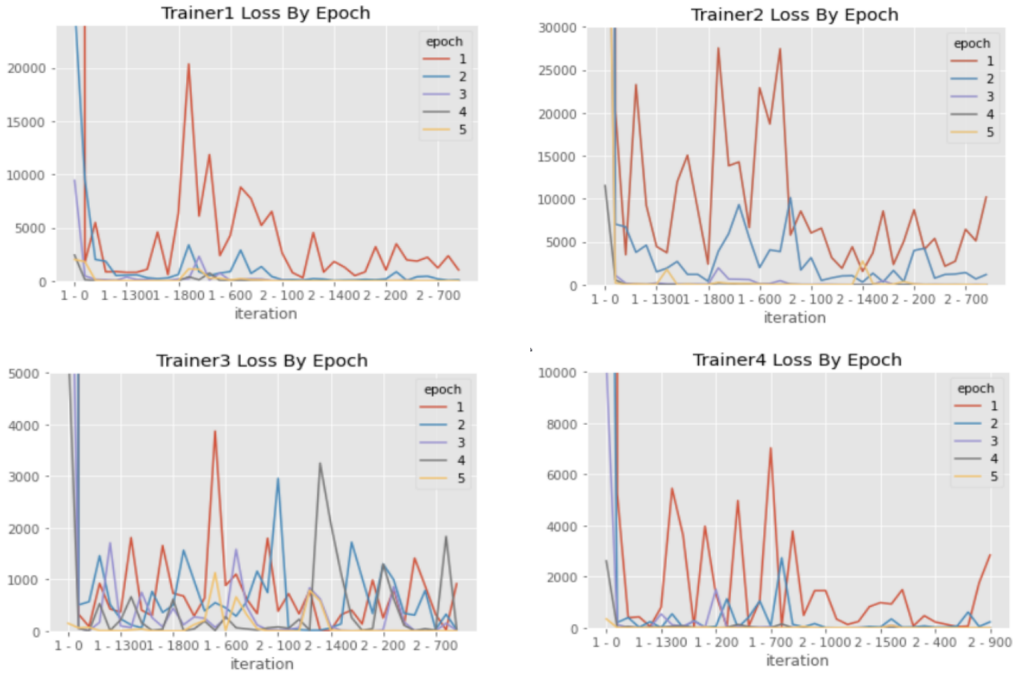

Figure 5: Local trainer loss across five global epochs. The entirety of the x-axis shows the metric at each local epoch and iteration combination.

converges to a different final local accuracy. We also note here that not all trainers carry the same performance, where Trainer 2 completes training with a local accuracy of more than 15 percentage points lower than the other three trainers. This can be an indication that our intuition regarding the data segmentation process was correct.

We also plot the loss and accuracy graphs for each trainer in Figures 5 and 6. The final accuracy in Table 1 corresponds to the last accuracy point for each trainer. We notice a few things here. First, there can be trainers that perform more poorly than others or struggle more to converge. Second, in situations where later epochs in a local trainer perform worse than previous epochs, there may be a case to implement early stopping, which could improve the global model. These topics are discussed in future work in Section 5.

Lastly, in Figures 7 and 8, we present the global model accuracy and loss from the aggregated model in the coordinator. With five global epochs, we see that the overall model accuracy closes at 83.6%, only 1.4 percentage points lower than the accuracy of the one-node model.

5 Conclusion and Future Work

AI and machine learning continue to grow in many domains, and network security and anomaly detection is a prime use-case. However, as with any AI application with vigorous and well-performing models, the main caveat in this area is the need for data, both in volume as well as quality.

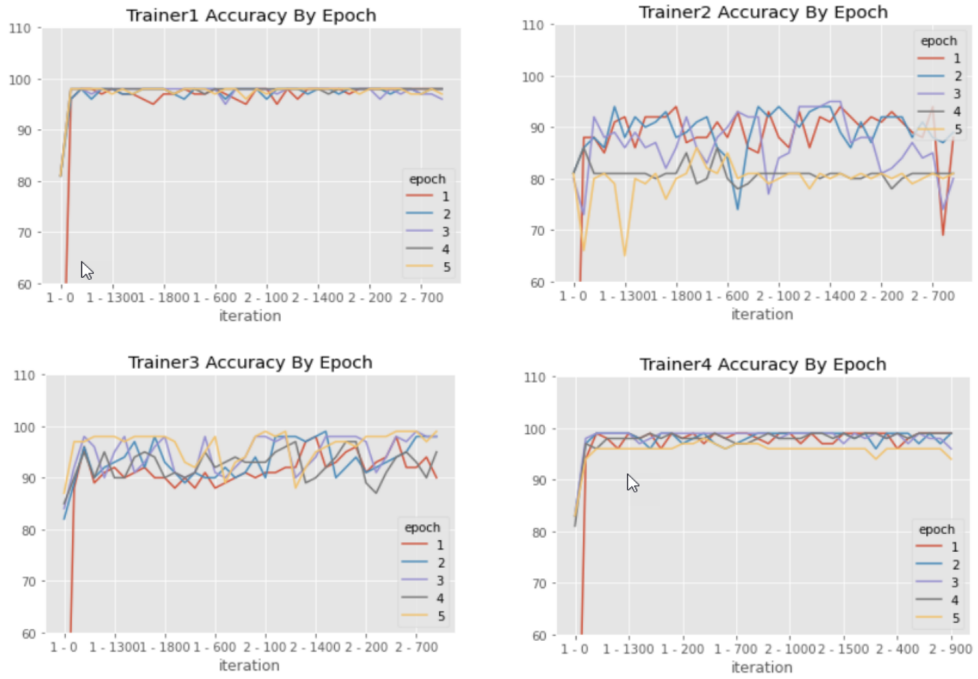

Figure 6: Local trainer accuracy across five global epochs. The entirety of the x-axis shows the metric at each local epoch and iteration combination.

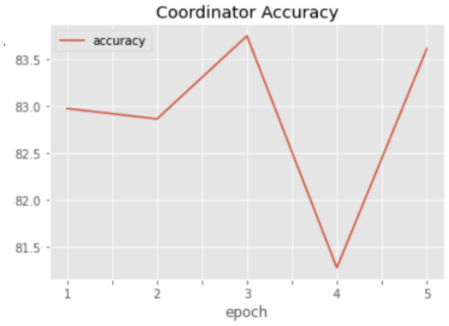

Figure 7: Coordinator accuracy across five global epochs.

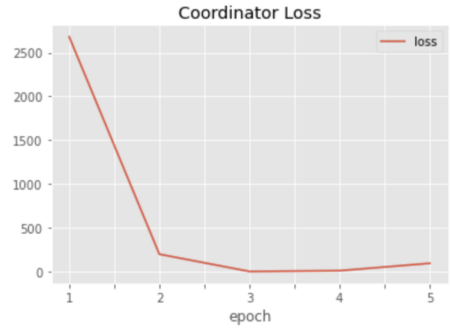

Figure 8: Coordinator loss across five global epochs.

Federated learning is an ideal solution for this domain. Training on real world data from actual devices provides a distinct advantage over training on proxy data that is generated from a data center. However, this data may be both extremely large in volume, which would make constant data updates to a central server prohibitive in cost and time, or can be highly sensitive. Local training of models is also not an option as the types of events detected may only be a small subset of the global population. With the distinct advantage of decoupling the model training from the need for direct access to the data, while also not making any generalizing assumptions about the distribution of the data, the federated averaging algorithm proves to address these issues while still providing accurate predictions for network attacks.

Our results for this approach are able to show the robustness of the federated learning model for the network intrusion use case. In particular, while the overall model accuracy for the federated process is slightly lower that of our single node baseline, we feel that the privacy and data latency trade-off is worth the marginal decrease in performance. In ad-

dition, we see from empirical results from McMahan et al. (4) that for situations where data is extremely non-IID, like in our data segmentation approach, more rounds of weight updates are needed to reach a stable equilibrium. Unfortunately, given our hardware and availability limitations, we were not able to achieve this.

Future work also includes enhancements that would likely further mitigate or close this gap entirely. One insight we saw from our federated learning training was that not all trainers showed the same loss decrease and accuracy increase for each epoch. For instance, in Figure 6, we see that Trainer 2 distinctly has a a lower accuracy in Epoch 5 than it does in Epoch 1. We note that this may be an additional hyperparameter to tune. One option is to tune when to pass the weights from the client to the server, and another is to apply a more sophisticated weighting algorithm to take into account which trainers are performing better. We could also save history of past iterations of accuracies and losses to use as checkpoints for the model updates. We believe that all of these solutions could drastically improve our model performance compared to the baseline.

Lastly, we note that our current architecture is synchronous: in particular, we require that all four client trainers are available on the network to send and receive model information before beginning training, and all four must stay online to complete the process. Future work could expand this to create an asynchronous setup that is more fault tolerant, where not all servers must be online at once. This is the architecture of a classic federated learning approach in production environments, but was outside of the scope of our work.

References

[1] Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., & Ahmad, F. (2021). Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1), e4150.

[2] Basnet, R. B., Shash, R., Johnson, C., Walgren, L., & Doleck, T. (2019). Towards Detecting and Classifying Network Intrusion Traffic Using Deep Learning Frameworks. *J. Internet Serv. Inf. Secur.*, 9(4), 1-17.

[3] Kanimozhi, V., & Jacob, T. P. (2019, April). Artificial Intelligence based Network Intrusion Detection with hyper-parameter optimization tuning on the realistic cyber dataset CSE-CIC-IDS2018 using cloud computing. In *2019 International Conference on Communication and Signal Processing (ICCSP)* (pp. 0033-0036). IEEE.

[4] McMahan, B., Moore, E., Ramage, D., Hampson, S., & Arcas, B. A. (2017, April). Communication-efficient learning of deep networks from decentralized data. *In Artificial Intelligence and Statistics* (pp. 1273-1282). PMLR.

[5] Povey, D., Zhang, X., & Khudanpur, S. (2014). Parallel training of deep neural networks with natural gradient and parameter averaging. *arXiv preprint arXiv:1410.7455*.

[6] Sharafaldin, I., Lashkari, A.H., & Ghorbani, A. (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. *ICISSP*.

[7] Zhang, S., Choromanska, A., & LeCun, Y. (2014). Deep learning with elastic averaging SGD. *arXiv preprint arXiv:1412.6651*.