Enhancing RetinaMNIST Federated Learning with PID Client Filtering

Anonymous Author(s)

1 Overview

Diabetic retinopathy (DR) is one of the leading causes of blindness around the world, and its early detection can make a difference in preventing one's loss of vision. Recently automated models have been introduced to help with the classification of the severity of the disease. However, these models are typically trained using centralized machine learning, so that the data comes from multiple clients (hospitals or clinics), and are collected all on one server. While these models are very efficient and makes it easier to train models with higher accuracy, having all of these images and other data in one large centralized 'database' not only poses multiple security concerns, but also raises a lot of privacy concerns. Medical data like this is protected by many regulations, and many clients are unwilling or not allowed to share these forms of patient records due to their confidentiality agreements. In addition to all of this, having one form of centralized storage for all of this imagery creates one large single point of failure, as if the server is compromised, all of its sensitive data becomes exposed. These limitations make centralized ML very difficult to use in health care settings.

To help with these issues, federated learning (FL) has become a much better option. Instead of just sending all of the retinal imagery to one large server, each client has the ability to keep their data on local machines and their own models for patients. What typically gets shared with the central medical server are only the model updates, and not the images or any other confidential information itself. This configuration keeps peoples' data private while allowing the model to grow and learn from a more diverse pool of data. Without federated learning, and because centralized ML is not typically an option in the health care scene, the lack of information that clients would have could lead to many concerns, such as false reporting of the severity of diseases (in this case DR). A clinic working with its own dataset might miss key patterns that appear in other pools of data, such as one's of larger hospitals, causing their model to underestimate or overestimate the severity of the disease which could lead to a misdiagnosis, delayed treatment, and many other unnecessary issues. All of this shows how important federated learning is for this field.

2 Related Work

A lot of different aggregation strategies have been proposed in recent years to make federated learning much more reliable in the medical field, each aiming to solve slightly different issues. Some methods have a larger focus on patient privacy, others having more of a primary focus on improving the performance on using non-independent and identically distributed (non-IID) data, and some aiming to defend against data poisoning attacks. The following subsections outline the different approaches taken in recent studies, which is then compared to highlight the methods and their strengths/weaknesses.

2.1 Federated Learning Methods

2.1.1 Secure Aggregation with Weight Hiding [1]. Xiao and Wang introduce a secure aggregation approach made to address privacy concerns when it comes to the phase of sending data out to clients. The main idea is to make sure that the central server, or anyone who may try to attack it, cannot figure out what data each client has by analyzing the updates they send. To make this possible they incorporate three different techniques:

- Weight Hiding Function Encryption
- Bayesian Differential Privacy
- Sparse Differential Gradients

All together, these three techniques allow for a secure aggregation process. First, the **weight hiding** function encryption makes sure that each client's updates are encrypted before they are sent out for updates. Each parameter is encrypted separately which are then passed to the centralized server. Next, **Bayesian Differential privacy** introduces noise to these updates, preventing possible attackers from being able to work backwards to find sensitive information. Finally, **Sparse Differential Gradients** reduce the amount of information that needs to be shared by only sending key updates, which not only helps with cutting down on communication costs, but also limits the amount of information that could be exposed.

2.1.2 Improved FedAvg for Medical Image Classification [2]. Li et al. propose a new version of the FedAvg algorithm to help with both privacy and performance when it comes to being tasked with medical image classification. The first modification to FedAvg comes from using an enhanced model called *SE-ResNet18-E* which follows a CNN architecture, in this case with the weights of their model being initialized randomly at the start of training. For FedAvg, this means that the aggregated global model starts from a common baseline from its clients, and reduces the risk of biases.

The second change comes from how updates are shared and distributed to clients. Instead of sending raw model weights and gradients to the server, a threshold Paillier encryption algorithm is put into place which allows each client to have the "same public-private key pair, and this public-private key pair is not allowed to be known to non-federation members and the central server. The model parameters obtained from the local training of each medical institution are encrypted with the public key and uploaded to the central server." [2] The central server can then go on to send the updated global model parameter ciphertext to each client , in which the client can then decrypt to plaintext with their private key.

Overall, the improved approach to FedAvg stated here helps strengthen the model's ability to learn from all sorts of medical imagery, while also using Paillier encryption to enhance privacy and reliability.

2.1.3 Mixed Data Calibration (MIDAC) for non-IID Data [3]. Zhang and Uludag focus on the fact that data in the real world are typically non-IID, meaning that in most cases, clients have very different

distributions of data. Some clients may have data on all different levels of a disease, while others may only have only very severe cases (again which can lead to misdiagnoses). To aid with this, the proposal of an aggregation method called **Mixed Data Calibration (MIDAC)** is introduced where after training on local machines, each client can create synthetic images by mixing together samples that they already have. These mixed samples and their labels are encrypted and sent to the centralized server. The server then aggregates the models per standard FedAvg algorithms, but then uses calibration steps using the new combined images from all of the clients. However, tnis step also introduces new weaknesses, since if data is poisoned, corrupted, or mislabeled, the calibration process can spread those errors to the global model.

This calibration step itself consists of multiple phases. The first step involves the local training completed by each of the clients. Normally here is when the updates are sent to the central server but instead of sending raw images, each client takes a small portion of its dataset and mixes multiple images together. This *synthetic* sample now represents a pattern for a given class/label, without having to share the original images. The server then performs regular FedAvg, encrypting the local model updates to get a global model. Finally once this model is formed, the server takes the new synthetic images and uses it to calibrate and train the local model based on the new information. This allows for the trained model to display the data used by all clients with less risk for security and privacy concerns than if the original un-combined had been aggregated.

2.2 Comparing Strengths and Weaknesses

Each of the proposed aggregation methods build on FedAvg to each help solve a specific problem, they do all come with their own strengths and weaknesses as mentioned previously, and compared to each other as seen here. When it comes to Xiao and Wang, data and patient privacy is the center of their concerns. Their techniques to address these issues (weight hiding encryption, Bayesian differential privacy, and sparse differential gradients), together helped with client updates to remain protected while still allowing for aggregation. Li et al. on the other hand, had more of a focus on improving the classification accuracy itself first, and then on maintaining the security. Their report used the SE-ResNet18-E architecture to help boost their accuracy, while Paillier encryption help to protect updates, showing that it is plausible to combine both stronger models with stronger privacy for clients and patients. Finally, Zhang and Uludag had more of a focus on non-IID data among clients, creating synthetic images and use them in their calibration steps, all allowing for their model to represent all of the data among clients while mitigating any privacy concerns.

The results of all of these methods more or less matched what the authors all expected out incorporating their methods in practice, though even their expectations still left some security concerns up in the air. Xiao and Wang came to the conclusion that their aggregation strategy demonstrated how privacy could be kept without a large loss in accuracy, but that having these scale hand-in-hand would be very computationally expensive which would greatly limit scalability. Li et al.'s results were very similar in the way that they showed that accuracy could be gained even with non-IID datasets, but the encryption also comes at a big cost of computation, and distributing the updates to clients. Zhang and Uludag may have introduced the biggest weakness of the three reports. Although their calibration step and combining images was great for the accuracy of models under non-IID conditions, that extra step allows for the poisoning of the calibration samples.

Table 1: Summary of Aggregation Methods: Strengths and Weaknesses

Method & Study	Strengths	Weaknesses
Secure Aggregation (Xiao & Wang, 2023)	Protects privacy with encryption and differential privacy, hides client weights, reduces communication of params	Extra computation required,, slight accuracy drop due to added noise, does not address non-IID distributions
Improved FedAvg (Li et al., 2025)	Better accuracy with SE-ResNet18-E, encryption protects updates	High computation and communication cost, requires secure key management
Mixed Data Calibration (Zhang & Uludag, 2024)	For non-IID data, calibration step, avoids sharing raw images	Vulnerable to data poisoning, weaker privacy at class level

2.3 Summary of Comparisons

Federated learning makes it possible to train models in the medical field without the risk of having large amounts of sensitive centralized data and there are many ways this data can be aggregated as seen in the examples from these three reports. Each method analyzed focused on a different priority being privacy, accuracy, or the handling of non-IID data. While each would be viable in practice, they do all pose trade-offs whether it be in regards to computational complexity, communication, or vulnerability to attacks from malicious actors. Looking ahead some possible improvements that could be made include slightly lighter encryption which would allow for more overhead in when it comes to upping accuracy, or the other way around being sacrificing accuracy for more secure systems. It really comes down to finding the right balance especially having to do with the datasets selected, their size, and even the dataset's ability to support effective model training and output reliable predictions.

3 Data Preparation Strategy

For this report, MedMNIST was selected as the dataset, specifically the RetinaMNIST subset. The data of this subset contains retinal fundus images which are used to classify the severity of diabetic retinopathy in the eye. This presents a real healthcare application, with a data set that can be implemented into a FL framework.

3.1 Data Cleaning

The first step when it comes to data cleaning was making sure all the files were readable images, in which no corrupted files were

found (simulation of corruption will take place later). A check for duplicate images was also implemented, in which no duplicates were found. When sorting through the dataset, a very few amount of retinal images with white backgrounds were found, and a method was written to take out those inputs to help avoid possible bias in the training, picking up on background artifacts rather than the medically relevant features.

3.2 Data Normalization

For normalization, pixel values were scaled from 0-255 down to a 0-1 scale. The original three RGB color channels were kept to maintain the information. Also, since PyTorch requires inputs in channel-first format, the arrays were transposed during this preprocessing.

3.3 Data Augmentation

When it comes to data augmentation, transformations were applied only to the training set to help improve the models strength and prevent overfitting. These include random horizontal and vertical flips, as well as rotating images randomly in 90 degree increments.

3.4 Data Partitioning

(Implemented but not yet in use) The dataset was split into 20 clients using a random shuffle and equal partitioning, resulting in a non-IID distribution. Each client then performed an internal 80/20 split into local training and evaluation sets.

4 Initial Model Selection

The ResNet18 architecture was chosen as a pretrained option primarily because of how well it has been shown to work with medical imaging tasks. In particular, Li et al. [2] showed that their enhanced SE-ResNet18-E produced very strong results on medical image classification while still being efficient in federated settings. This made the pretrained model feel adequate for the dataset with it being very good at pulling out visual features. By making slight changes to the final layer to output to the five classes needed for RetinaMNIST, the model was immediately able to be applied to the dataset.

5 Description of Experiments

The goal of this experiment was to evaluate how poisoning attacks by label-flipping affects the performance of a federated learning model. The chosen dataset, RetinaMNIST, consists of retinal fundus images labeled depending on the severity of diabetic retinopathy in the eye. The label 0 represents a healthy retina and labels 1-4 represent the escalating severity of the retina's condition. For simplicity of this experiment, and due to the smaller dataset size of RetinaMNIST, the dataset was converted to a binary classification task, with class 0 representing healthy eyes and class 1 representing any stage of diabetic retinopathy. This overall allowed for a better evaluation with a more consistent baseline to then be compared to the model under various severities of attack.

The experiment was implemented as a local FL simulation through Flower, where all clients were trained sequentially without any outside networking. The total number of clients in the simulation was set to 20, where during the poisoning attacks 8 of which were randomly selected as malicious in each run.

Each simulation consisted of 12 federated training rounds, in which the number of rounds was determined by initial testing showing that performance began to plateau after roughly 10 rounds and also to avoid completely random accuracy fluctuations and overfitting in the client's local data distributions. At the beginning of every round, the global model parameters are shared to all clients. Then each client performs a single local epoch of training on the partitioned dataset it was provided with. At the end of the round the local parameters were then returned to the 'central server'. The server then aggregated the updates using FedAvg to produce a new global model. During each step of every round, both the per-client and global metrics, including accuracy and loss, were stored in respective CSV files for later analysis.

After this implementation was complete, a baseline experiment was conducted where no clients were poisoned, followed by 5 more experiments in which 8 of the 20 clients were attacked with flipped-labels based on the respective flip-rate of the given attack scenario.

The final results were analyzed based on five main metrics:

- Loss per client over rounds
- Average loss among clients
- Accuracy per client
- Average accuracy among clients
- Final global loss and accuracy vs. attack severity

These allowed for comparisons between clean and poisoned runs, showing how increasing the severity of flipping labels affected overall model reliability and accuracy. Higher flip rates consistently resulted in greater loss, reduced accuracy, and less stable training behavior across clients.

The goal of the next experiment was to extend the previous analysis by integrating a PID anomaly detection layer into the FL process. The same dataset of RetinaMNIST and model architecture of ResNet18 were used in order to stay comparable to the original results without the anomaly detection. The main difference was that after every round, each client's weights were compared to the centroid of all client models and a PID controller was used to calculate a 'PID Score'. This score shows how far a client's updates are from the global model.

Clients with a PID score that was larger than the given threshold were excluded from FedAvg for that round. This step is there to help simulate a real-world FL environment where malicious activity is bound to exist. Experiments followed the same attack severity and flip rates as before.

6 Description of Code

The following section provides an overview of how the FL system was implemented and explains how to run the provided notebook to reproduce the experiments. It includes setup instructions for running the notebook through Google Colab and an explanation of the code as a whole and its implementation.

6.1 Running the Code

The notebook was designed and tested within Google Colab for accessibility and GPU acceleration purposes. In order to run the code:

(1) Open the notebook in Google Colab (File > Upload Notebook)

(2) In the top menu-bar, Runtime > Change runtime type, and set Hardware accelerator = T4 GPU
(3) Run the first pip install block and wait for completion, it will prompt you to restart the runtime
(4) Restart the runtime, then run by selecting 'Run All'

This ensures that all dependencies are installed correctly and the GPU is active before beginning with training. The main federated training loop takes around 13 minutes, but the results of training are visible in the outputs of the notebook provided.

If you plan to run the FL model with anomaly detection, run all cells until you reach the markdown that states **Stop here if running PID scenarios** and do not run any cells until you reach the markdown that states **Continue here for PID Scenarios**. This is in place to prevent variables from overwriting each other as some names have been reused in the notebook environment. This training loop takes over an hour to complete.

6.2 Description of the Code

The implementation starts by importing all required dependencies and loading the MedMNIST retina dataset. The code installs and imports Flower, PyTorch, MedMNIST, and plotting libraries such as pandas and matplotlib. The RetinaMNIST dataset is then loaded, and a brief visualization displays sample images to verify the data and its format.

Next, the dataset is cleaned and goes through preprocessing where a helper function removes samples with white backgrounds by checking the average brightness of corner pixels, this is for consistency as the dataset itself is not too large. The images with darker backgrounds form the training dataset. Both the accepted and discarded samples are displayed to confirm the images were filtered properly, ensuring the model is being trained on clean data, removed of artifacts that could bias the classification.

After cleaning, the original five class labels are converted into a binary classification, where 0 represents a healthy retina and 1 represents any of the severity levels of diabetic retinopathy. A dataset expansion function is also used to increase the effective training size by generating transformed copiers of each image. To do this, each image is flipped horizontally and vertically, as well as rotated by increments of 90 degrees, to produce multiple variants of each original image for the dataset.

The model itself is implemented inside of a custom RetinaCNN class, wrapping a pretrained ResNet-18 model. The class includes methods for normalization, optional augmentation, training, and evaluation. It also contains functions to manage parameters like get_parameters() and set_parameters() which allow model weights to be extracted and reassigned for the federated aggregation.

To simulate the distributed setting, more helper functions handle dataset partitioning and label manipulation. The partition_data() method evenly divides the dataset among all clients, to_loader() converts the subsets into PyTorch DataLoaders, and the flip_labels() function which brings in label flipping attacks by randomly changing the given percentage of labels within poisoned clients.

The main training for the Federated Training loop is within the federated_training() function, implementing a sequential version of FedAvg. This begins by initializing a global model and distributing parameters to each of the clients. Each client trains locally on its partition of data for an epoch at a time, then returns the updated weights to the server. The server averages these updates to produce a new global model. This repeats for the twelve rounds mentioned before, recording metrics at each step.

The baseline is then ran along with the different attack scenarios. First executing a clean training run without any poisoned clients, saving the results in a baseline metrics file. Then training is repeated for attack scenarios where 8 of 20 clients are poisoned with label flip rates of 10%, 25%, 50%, 75%, and 100%. Each of these outputs is put in their own respective metrics file for later analysis.

Finally, multiple plotting functions were created in order to visualize the metrics that were recorded.

6.2.1 Inclusion of Anomaly Detection. This section was expanded upon with changes made to the original FL implementation to integrate the PID anomaly detection layer. The main body of the code remained the same, with the necessary additions to make the decision to exclude 'unstable' clients at the beginning of every round per each attack scenario. The *federated_training_with_pid* method handles all of the main operations. After each local training step, the server collected the client weights and calculated their euclidean distances from the global centroid. Using the proportional, integral, and derivative gains the client's PID score was found individually, to decide whether or not to include it in the aggregation step for the given round. Clients with a score that fell above the threshold were excluded for that round with the goal to minimize the amount of poisoned updates. Finally, the code first runs a baseline session without any poisoned clients, and then iterates through the five different attack setups. Each run logs all of the client metrics in a .csv for later plotting and analysis.

7 Interpretation of Results

This section focuses on how the model performed during both the baseline run and the various attack scenarios, allowing for comparisons of loss and accuracy to be drawn. Each experiment tracked how both per-client and averaged metrics allow to see patterns in how the model trained over time.

7.1 Baseline (No Attack)

In the baseline experiment, the per-client loss plots showed shows relatively stable behavior. Figure 1 shows this loss per client over the 12 rounds. However, when looking at figure 2 there is a bit more instability seen in the average loss across all of the clients over the rounds before it tapered off at around 0.66.

When taking a look at the models accuracy, the evaluation accuracy across each individual client shows an upwards trend as was expected. The accuracy begins to improve early and then fluctuates near a stable range between roughly 65%–85% across most clients as seen in figure 3. Finally, figure 4 displays the average evaluation accuracy across all of the rounds, showing a similar spike in accuracy in the first 4 rounds to figure 3 before tapering off and reaching a plateau near 80% - 82%. Here it remains relatively consistent providing a baseline benchmark for comparison with the attack scenarios.

These results confirm that the baseline model was learning as expected, demonstrating stable communication between the clients and the global server through the FedAvg aggregation process.

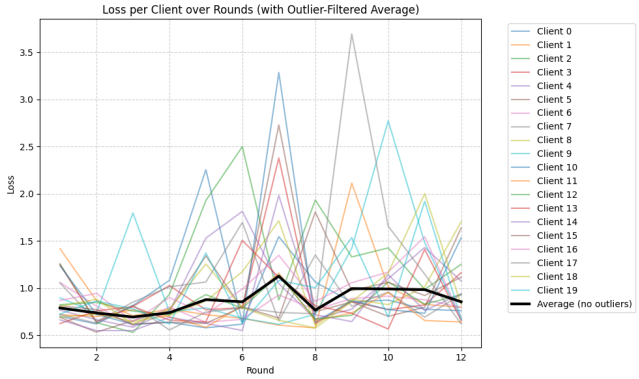

Figure 1: Loss per client over rounds (Baseline)

Figure 2: Average client loss over rounds (Baseline)

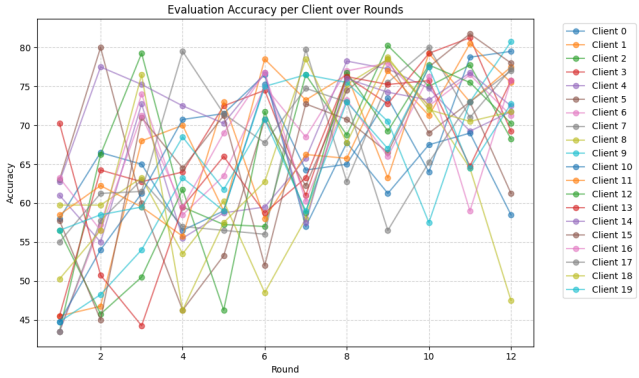

Figure 3: Accuracy per client over rounds (Baseline)

7.2 Attack Scenarios

When label-flipping is introduced, eight out of the twenty clients were randomly selected as poisoned in each case. Across the different attack severities, the results show that the global convergence was slightly effected meaning that the aggregation remained relatively stable. At round twelve the loss values began to converge, showing that the system stayed resistant to poisoning even at the higher flip rates. For all attack severities, the average loss started at around 0.66 and ended at around 0.61 as seem in figure 6.

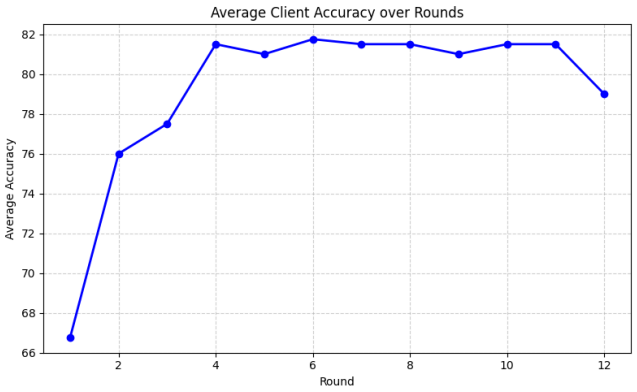

Figure 4: Average client accuracy over rounds (Baseline)

The accuracies of the clients had more variance in the higher attack settings, where it can easily be made out just from the visualization in figure 7. Also in figure 9 it can be seen that the average accuracy across the clients of each attack scenario stabilizes at around 76-82%. Another great visualization of how poisoned clients were affected can be seen in figure 8, where it can easily be made out which of the eight clients had been poisoned by taking a look at their resulting accuracy, in this case of the Flip-50 attack (something that can't be seen when just looking at the averages of all clients). The average accuracy still remained close to that of the original baseline experiment suggesting that though the malicious updates affected some local performance, the overall performance of the global model remained strong not causing total degradation.

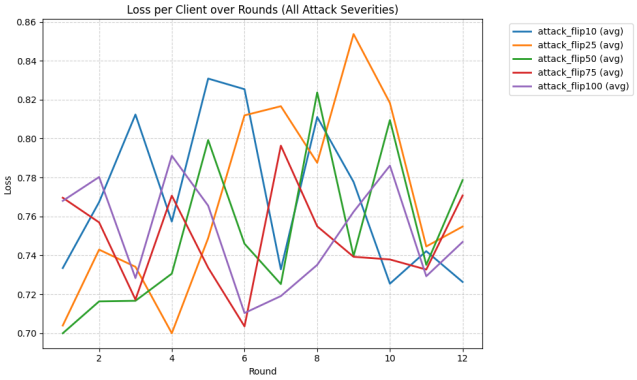

Figure 5: Loss per client over rounds (Attack)

7.3 PID Attack Scenarios

The addition of PID anomaly detection greatly improved the overall accuracy of the model compared to the original FL runs without any of the anomaly detection. When looking at the loss and accuracy plots across all of the clients, the new changes filtered out clients who were poisoned, and an example of that confidence can be seen in figure 10. In this given figure, it can be seen that clients 8, 9, 12, and 14 were all trusted sources throughout the training process, while clients 0, 2, 6, 10, 13, 15, and 17 were deemed to be volatile and were not included in the FedAvg aggregation for three of the twelve rounds of training.

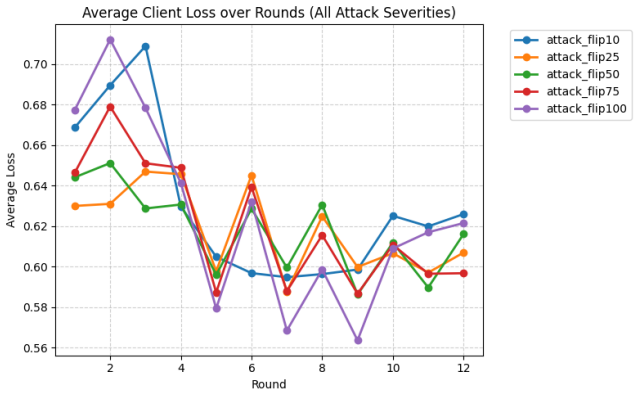

Figure 6: Average Loss over rounds (Attack)

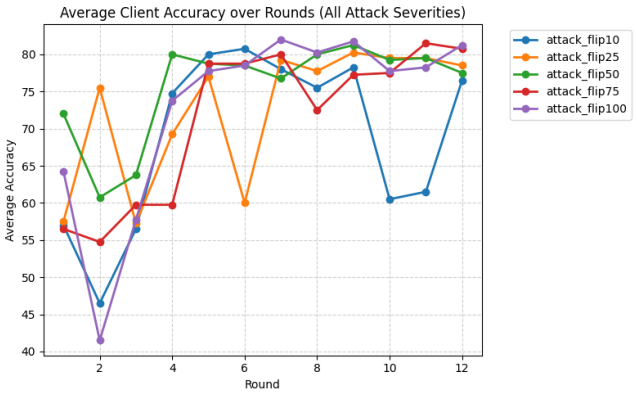

Figure 9: Average Accuracy over rounds (Attack)

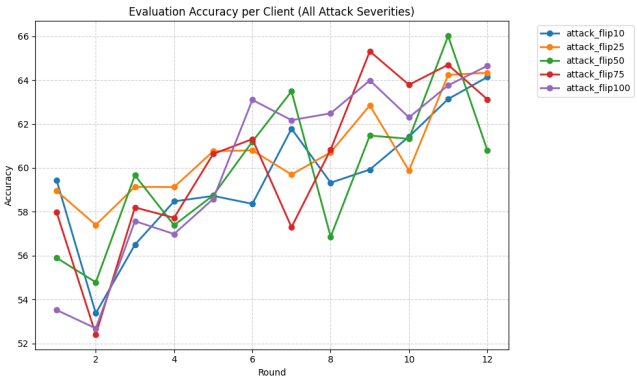

Figure 7: Accuracy per client over rounds (Attack)

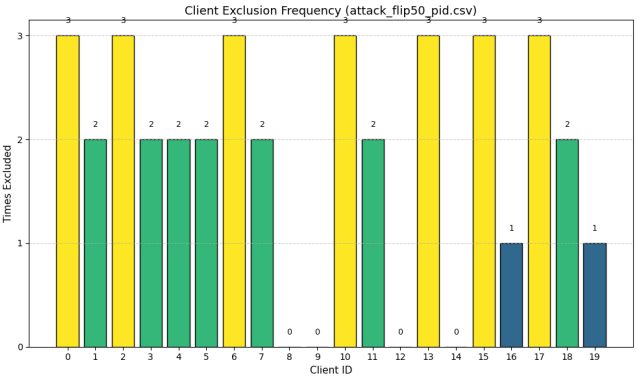

Figure 10: Client Exclusion Frequency for Flip50 Attack)

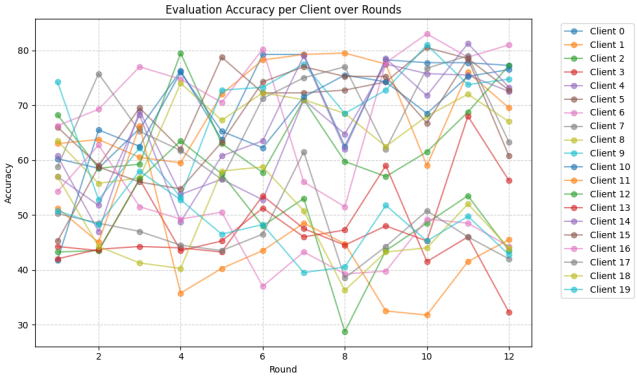

Figure 8: Client Accuracy over rounds (Attack Flip50)

Figure 11: Training Loss Per Client for Flip50 Attack)

The training loss per client as seen in figure 11 and an example of the average loss across all clients in the Flip75 attack scenario in figure 12 show much smoother curves and more of a downward trend as to how they appeared before in the regular FL experiments.

As for the accuracy of the resulting models, inclusion filtering helped with keeping higher and more stable accuracy across the rounds, even in the higher flip rate attack scenarios. The training accuracy per client in the Flip50 attack can be seen in figure 13, while an average accuracy across all clients over rounds in a given scenario can be seen in figure 14 which displays the average accuracy per round during the Flip25 attack.

Finally, figure 15 shows the relationship between the final loss and accuracy as the flip rate increases. What was unexpected is that the loss and accuracy seem to peak at the 50Flip attack scenario, which raised some skepticism about the result. However, this could be explained by the PID controller being the most effective at that level of corruption, doing a great job at filtering out the poisoned

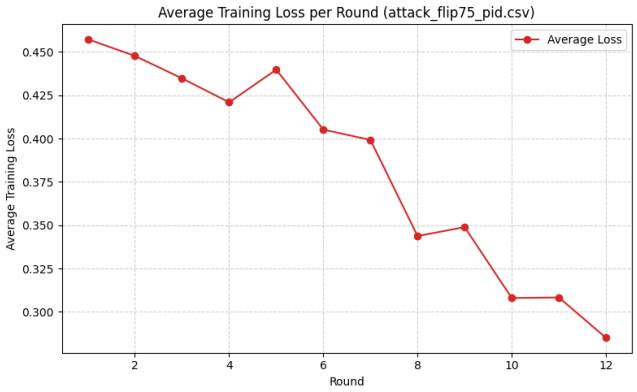

Figure 12: Average Loss for Flip50 Attack)

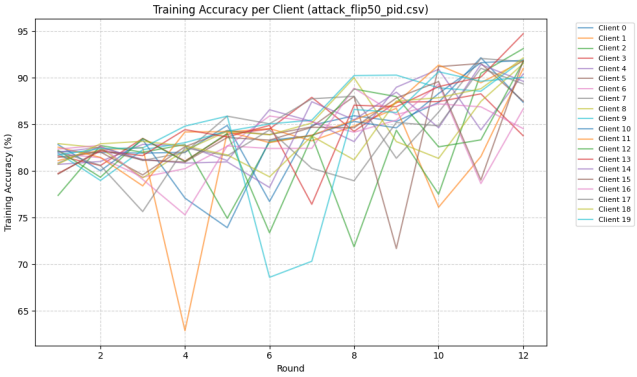

Figure 13: Training Accuracy Per Client for Flip50 Attack)

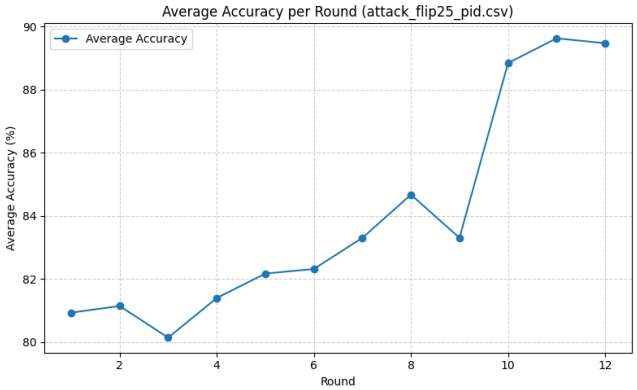

Figure 14: Average Accuracy Across Clients for Flip25 Attack)

clients and at the same time, keeping the healthy ones to maintain stable training.

8 Conclusion

Overall, this experiment displayed how a federated learning system reacts when label flipping attacks are introduced, resulting in clients sending data altered to be intentionally incorrect for training. Using RetinaMNIST, along with the preprocessing steps such as the removal of images with white backgrounds and augmenting

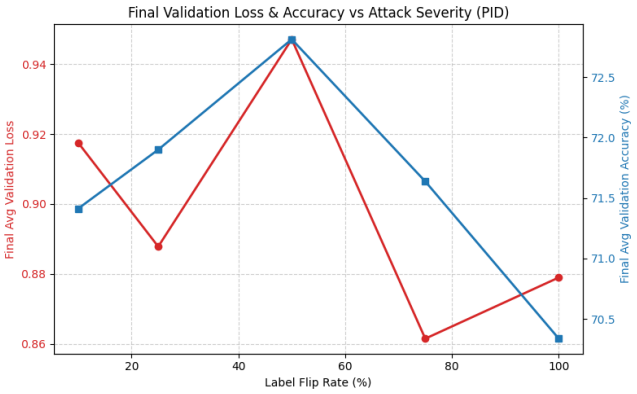

Figure 15: Loss and Accuracy vs. Attack Scenario)

copies of images to create a more robust dataset, a realistic setting for federated learning was introduced whose setup also allowed for the introduction of PID filtering for anomaly detection, similar to how these environments perform in the real world.

In the standard FL trials, the model still learned, but its behavior was noticeably less consistent. The poisoned clients were east to determine through the accuracy plots across all the attack severity levels, and the loss curves showed more bend and noise over the rounds. Even though the poisoned clients were noticeable through the plots, nothing was being done about them in this FL environment without the anomaly detection. Once PID was implemented, the system showed a much steadier behavior with loss dropping in a much smoother, nearly linear, pattern. Not only were improvement to loss found, but accuracy also seemed to have a large improvement over the rounds performed without PID. PID consistently removed clients whose updates were not similar enough to the patterns of others', in turn stopping their bad information from being averaged and aggregated into the global model. Though, one improvement that could be made in the future would involve using identical partitions of data between the clients of the standard FL and the PID trials. As in the current state, all clients end up starting with different accuracies and loss for each of the attack severities, leading to the average accuracy and loss of each scenario to not allow for complete direct comparison, though the improvements in these metrics from scenarios lacking anomaly detection to the scenarios with PID is still very clear.

All together, these results show that simple filtering methods like PID can make federated learning a lot more stable and reliable, even when almost half the clients within a FL environment are sending in malicious data. While the system was still able to learn without any protection in place, adding a lightweight anomaly detection step made the training process a lot smoother, giving better outputs, and making the global model less effected by bad data.

9 Elaboration on Team Distribution

This project and report has been completed **individually**, with part 2 taking roughly **30+ hours** and part 3 taking about half of that time to complete between implementing code, writing the report, and time spent training models.

References

[1] J. Xiao and S. Wang, "Data aggregation method for privacy protection in federated learning environment," in *2023 3rd International Conference on Electronic Information Engineering and Computer (EIECT)*, pp. 620–623, 2023.

[2] R. Li, H. Wang, Q. Lu, J. Yan, S. Ji, and Y. Ma, "Research on medical image classification based on improved fedavg algorithm," *Tsinghua Science and Technology*, vol. 30, no. 5, pp. 2243–2258, 2025.

[3] X. Zhang and S. Uludag, "Poisoning attacks against non-iid federated learning with mixed-data calibration," in *2024 7th International Conference on Machine Learning and Natural Language Processing (MLNLP)*, pp. 1–5, 2024.