CS350 - MINI PROJECT II

# Federated Learning for Pneumonia Detection and Segmentation

Leveraging ResNet for Classification and nnU-Net for Segmentation in a Privacy-Preserving Framework

---


**PROFESSOR** : Dr. ANNAPPA B

**MENTOR** : Naveen Kumar M R

**TEAM MEMBERS** :

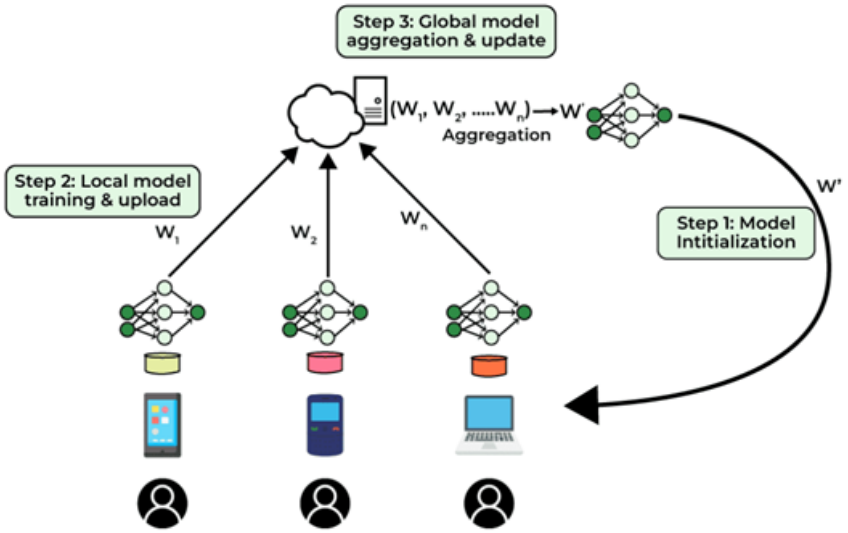
| | |
|---|---|
| NAMBURI YASWANTH | 221CS232 |
| N NAGABHUSHANAM | 221CS231 |

# Federated Learning

Federated Learning (FL) is a machine learning approach where the training process occurs directly on distributed devices or servers (clients) without moving data to a central server. This method ensures privacy and reduces data transfer overhead while leveraging decentralized data for model training.

## Key Features of Federated Learning:

1. Data Privacy: Keeps sensitive data on the client devices, adhering to regulations like GDPR, CCPA, and others.
2. Scalability: Utilizes distributed data from multiple devices or organizations, enabling training across vast and diverse datasets.
3. Efficiency: Reduces data transmission costs by sharing only model updates, not raw data.



## How Federated Learning Works:

---

1. Initialize Global Model:
    - A global model is initialized on a central server with random or pre-trained parameters.
2. Distribute Model:
    - The server sends the global model to client devices/organizations.
3. Local Training:
    - Clients train the model using their local data for a limited duration, like one epoch or a few mini-batches.
4. Update Aggregation:
    - Clients send model updates (parameters/gradients) back to the server.
    - The server aggregates updates (e.g., using Federated Averaging) into a new global model.
5. Iteration:
    - Steps 2-4 are repeated until the model converges to the desired performance

## Applications:

---

- Healthcare: Combine patient data from different hospitals to train diagnostic models.
- Finance: Detect fraud by utilizing data from multiple financial institutions.
- Smart Devices: Improve personalized AI models, e.g., predictive text on mobile keyboards.

## Benefits Over Centralized ML:

---

- Privacy Compliance: Eliminates the need for raw data transfer, adhering to user privacy preferences and regulations.
- Reduced Costs: Avoids transmitting large volumes of data to a central server.
- Access to Diverse Data: Leverages data from multiple, geographically distributed sources.

## Supporting Technologies:

---

- Federated Evaluation: Evaluate models on decentralized data to derive metrics.
- **Federated Analytics**: Perform statistical analysis on decentralized data for insights.
- **Differential Privacy**: Protects individual privacy by adding noise to model updates.

**Flower** is a framework that simplifies federated learning by providing infrastructure for training, evaluating, and analyzing federated models. It supports the development of advanced federated learning systems while emphasizing privacy and scalability.

# Flower Framework

---

Flower is an open-source framework designed to facilitate **Federated Learning (FL)** by providing the necessary infrastructure for training, evaluating, and analyzing federated models. It enables researchers and developers to build scalable and privacy-preserving FL systems with minimal effort.

### Key Features of Flower:

- Cross-Platform Compatibility **:** Works with popular ML frameworks like PyTorch, TensorFlow, and JAX.
- Flexible Client-Server Architecture **:** Supports heterogeneous clients (e.g., mobile devices, edge devices, or cloud servers).
- Simulation & Real-World Deployment **:** Allows running FL simulations on a single machine before deploying to real-world distributed environments.
- Privacy & Security : Integrates with privacy-preserving techniques like Differential Privacy and Secure Aggregation.
- Scalability : Handles FL setups ranging from a few clients to thousands of devices.

## How Flower Works in Federated Learning:

---

1. Server Initialization: The central server starts and defines the FL strategy (e.g., FedAvg).
2. Client Selection & Training: Selected clients train the global model on their local data.
3. Model Updates & Aggregation**:** Clients send updated model parameters to the server, which aggregates them.
4. Iteration: The process repeats until the model converges.

## Use Cases of Flower:

---

- Healthcare: Training models across multiple hospitals without sharing patient data.
- Finance: Fraud detection across different banks while preserving customer privacy.
- Edge AI: Improving AI models on mobile devices (e.g., personalized recommendations).

# Hybrid Partition

---

Hybrid partition in Federated Learning (FL) combines:

- **Dirichlet Distribution** to introduce non-IID (non-Independent and Identically Distributed) data by assigning different class proportions to clients.
- **Stratified Sampling** to ensure each client has some samples from all classes, preventing missing classes.

## Why Use Hybrid Partition?

---

In real-world FL, data is:

- **Imbalanced**: Some clients have more data for certain classes.
- **Non-IID**: Data is not uniformly distributed across clients.

Hybrid Partition solves this by:

- Creating class imbalance through **Dirichlet**.
- Ensuring class representation through **Stratified Sampling**.

## How it Works?

---

1. **Dirichlet Distribution (α)**: Controls data skew.
    - Lower α → High skew → Few dominant classes per client.
    - Higher α → Low skew → Balanced data.
2. **Stratified Sampling**: Ensures no class is entirely missing in any client.

Example

- Client 1: Mostly Class A, few B, C, D, E.
- Client 2: Mostly Class C, few A, B, D, E.

## Benefits

- Maintains data diversity while simulating real-world non-IID scenarios.
- Improves model generalization by balancing data heterogeneity and class coverage.

## Conclusion

Hybrid Partition optimizes FL by combining **Dirichlet** for realistic imbalance and **Stratified Sampling** for class coverage, ensuring better model performance across diverse clients.

# nnU-Net

---

nnU-Net is a self-configuring deep learning framework for biomedical image segmentation. It automatically analyzes a dataset and customizes a U-Net-based model pipeline—no manual tuning required.

## Key Features:

---

- Supports 2D & 3D image segmentation.
- Handles diverse modalities (e.g., X-Ray, CT, MRI, microscopy).
- Requires no deep learning expertise to use.
- Automatically sets architecture, training, preprocessing, and postprocessing steps based on dataset analysis.

## How it works:

---

1. Dataset fingerprinting → learns dataset characteristics.
2. Chooses between 2D U-Net, 3D full resolution U-Net, or 3D cascade U-Net based on image size and modality.
3. Applies a three-step configuration: fixed, rule-based, and empirical parameters.

## Scope & Limitations:

---

- Best for custom biomedical datasets or research use-cases from scratch.
- Not ideal for standard RGB datasets like Cityscapes; pretrained foundation models perform better there.

## Why use nnU-Net V2?

---

- Clean architecture, better usability.
- Designed as a development and benchmarking framework.
- Widely adopted (e.g., MICCAI challenge winners).

# Results

---

### Centralized:

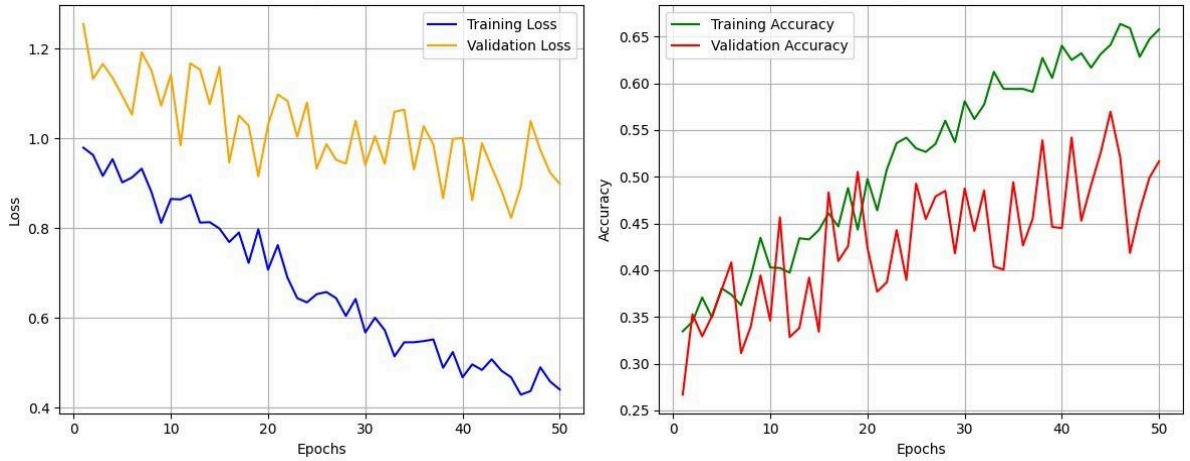



Figure: Training results with ResNet34

This approach led to an under-fit model due to:

1. Class imbalance in the dataset.
2. Use of Adam optimizer, which has less generalization capability compared to SGD with momentum.

---

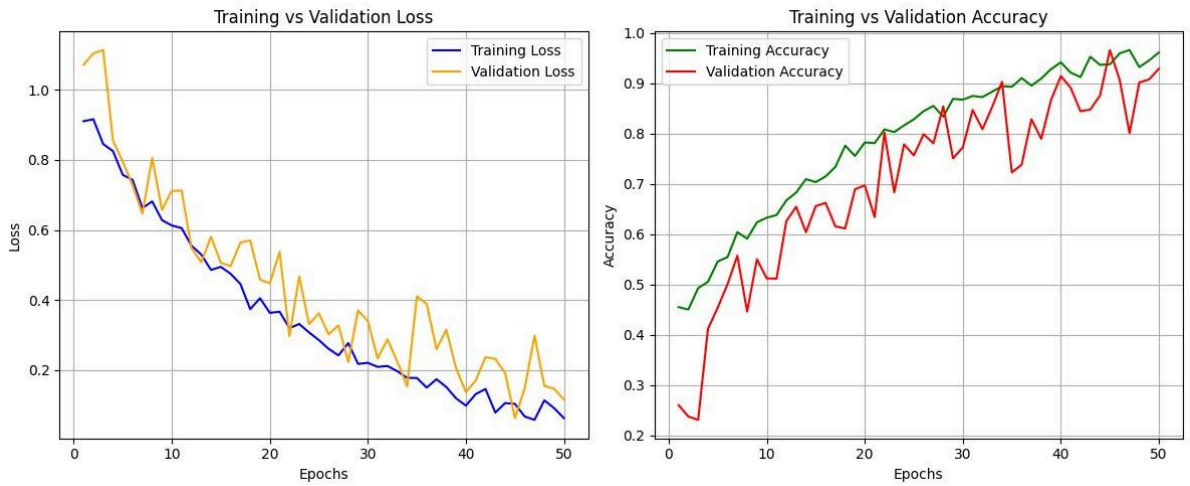



Figure: Training results with ResNet50

This showed to be a better model after resolving the issues with the ResNet34 model, and as this is a deeper CNN this lead to the extraction of much larger features than the 34 layer model.

---

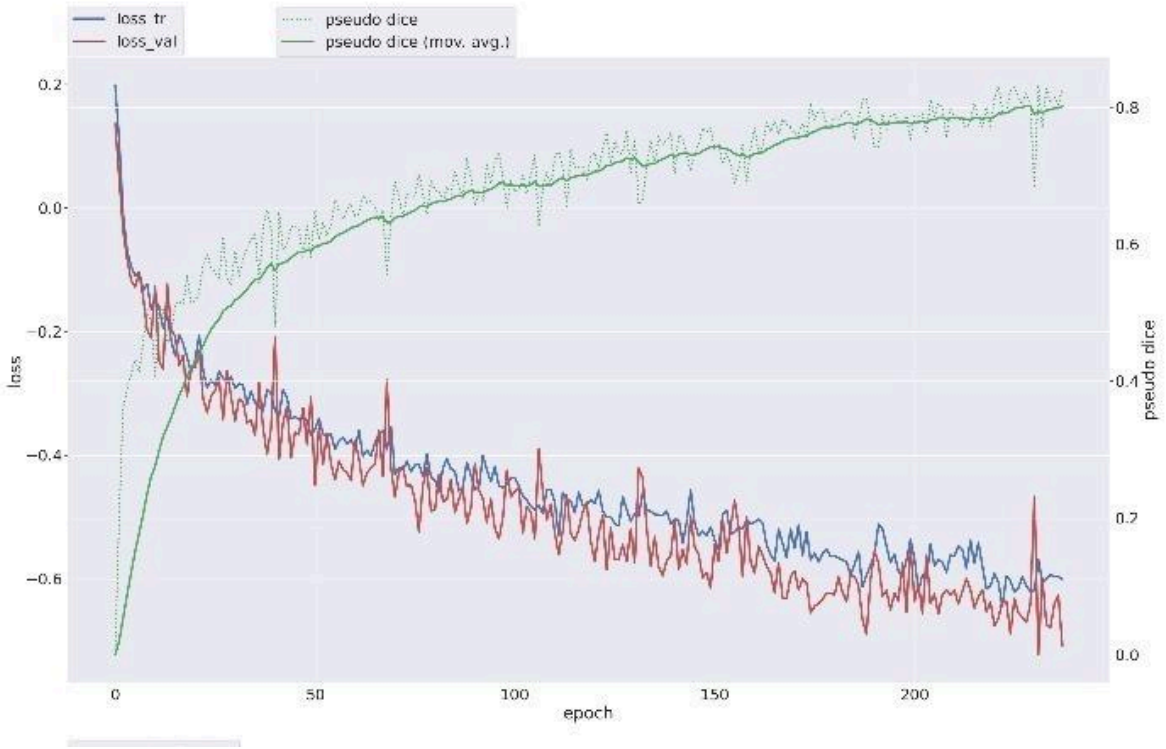



Figure: Training results with nnU-Net

This shows us that the model converges over 240 epochs and reaches near -0.6 loss value with the best being -1.

1. The validation loss curve doesn't seem to show any overfitting.
2. The dice score achieved was around 0.82
3. The hyperparams used were:
    a. MODEL: PlainConvUNet
    b. EPOCHS = 240
    c. BATCH SIZE = 12
    d. LEARNING RATE = 0.01
    e. OPTIMIZER = SGD with MOMENTUM = 0.9
    f. LR Scheduler = Polynomial decay: with exp=0.9

The loss function use was:

• Combined Dice Loss and Cross Entropy Loss

• Designed to handle class imbalance and segmentation accuracy, ranges from +inf to -1

- **Soft Dice Loss:**

$$\mathcal{L}_{\text{Dice}} = -\frac{2\sum_i p_i g_i + \epsilon}{\sum_i p_i + \sum_i g_i + \epsilon}$$

- **Cross Entropy Loss:**

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N}\sum_i g_i \log(p_i)$$

- **Combined:**

$$\mathcal{L}_{\text{Total}} = \lambda_{\text{CE}} \cdot \mathcal{L}_{\text{CE}} + \lambda_{\text{Dice}} \cdot \mathcal{L}_{\text{Dice}}$$

$$\lambda_{\text{CE}} = 1, \lambda_{\text{Dice}} = -1$$

---

**Federated:**

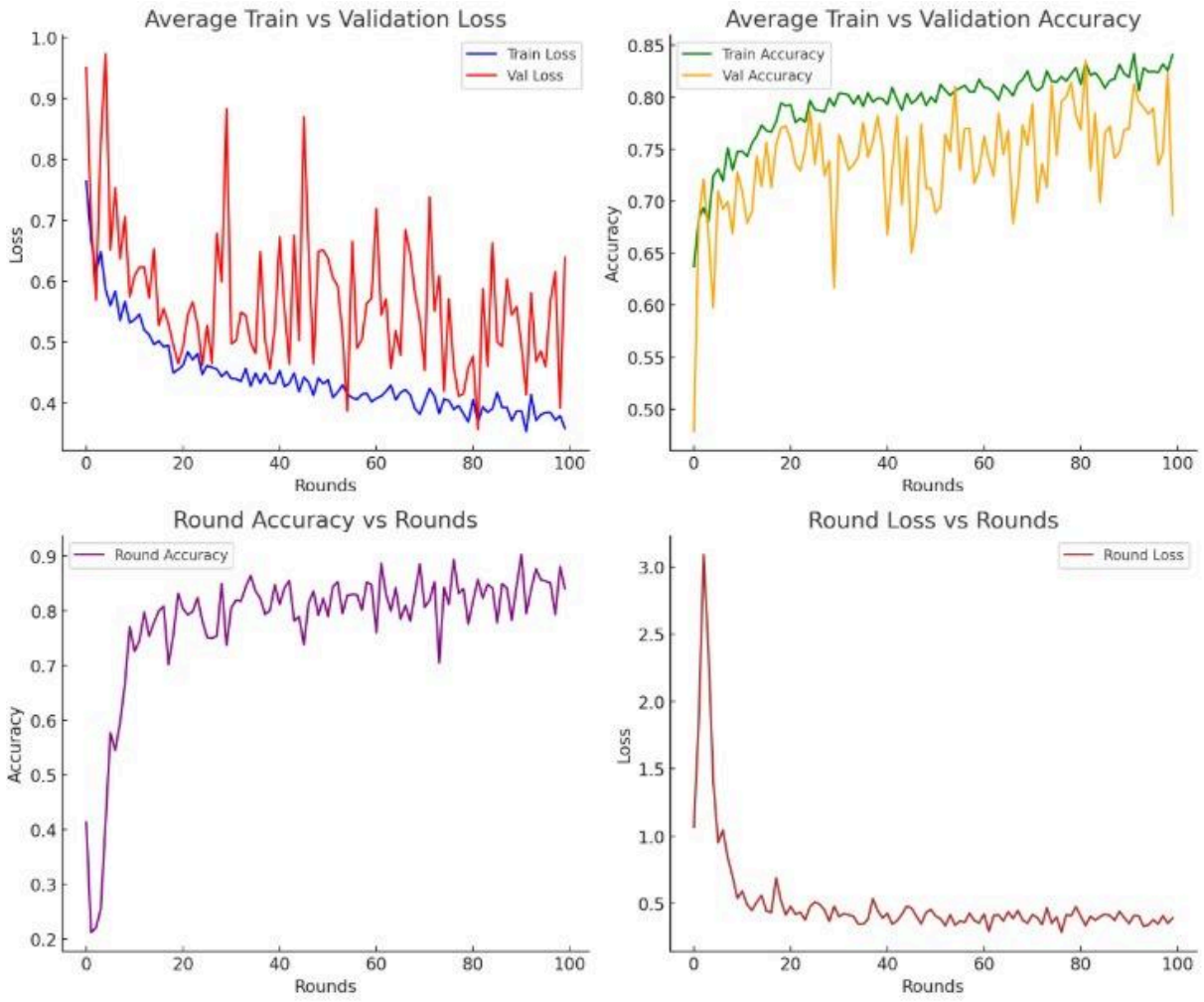



Figure: ResNet50 (Federated) Results

The federated approach with resnet50 involved these params:

a. NUM PARTITIONS = 20
b. MIN FIT CLIENTS = 6
c. EPOCHS = 1
d. NUM ROUNDS = 100
e. BATCH SIZE = 32
f. LEARNING RATE = 0.003
g. OPTIMIZER = SGD with MOMENTUM = 0.7

---

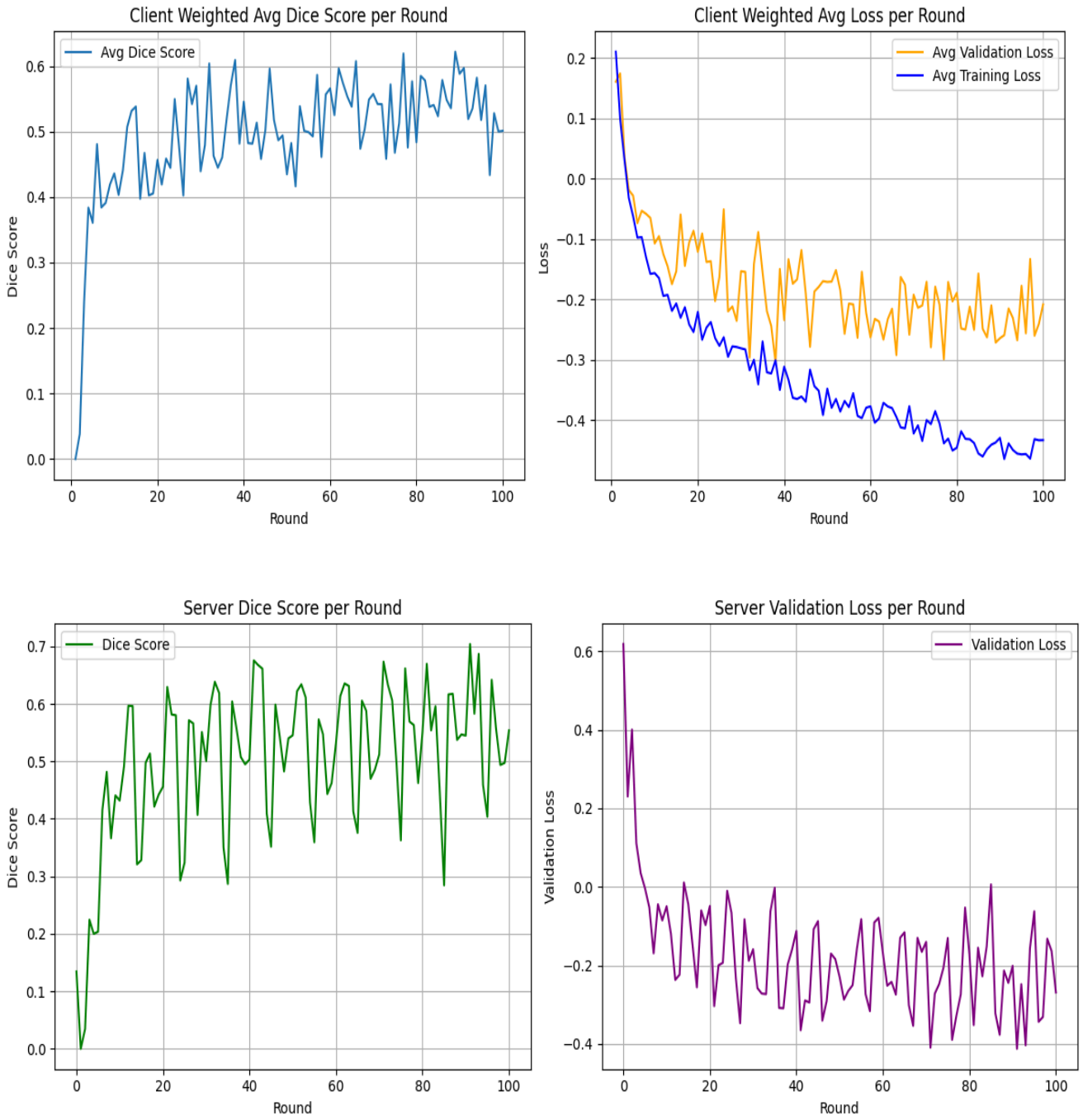



Figure: nnU-Net (Federated) Trail-1 Results

The Federated approach for nnU-Net seems to converge but the validation curve seems to flatten out due to class imbalance and several bottom valleys are found in server dice score as some of the clients had very high class imbalance in their partition leading to overfitting with no high level features being learnt.

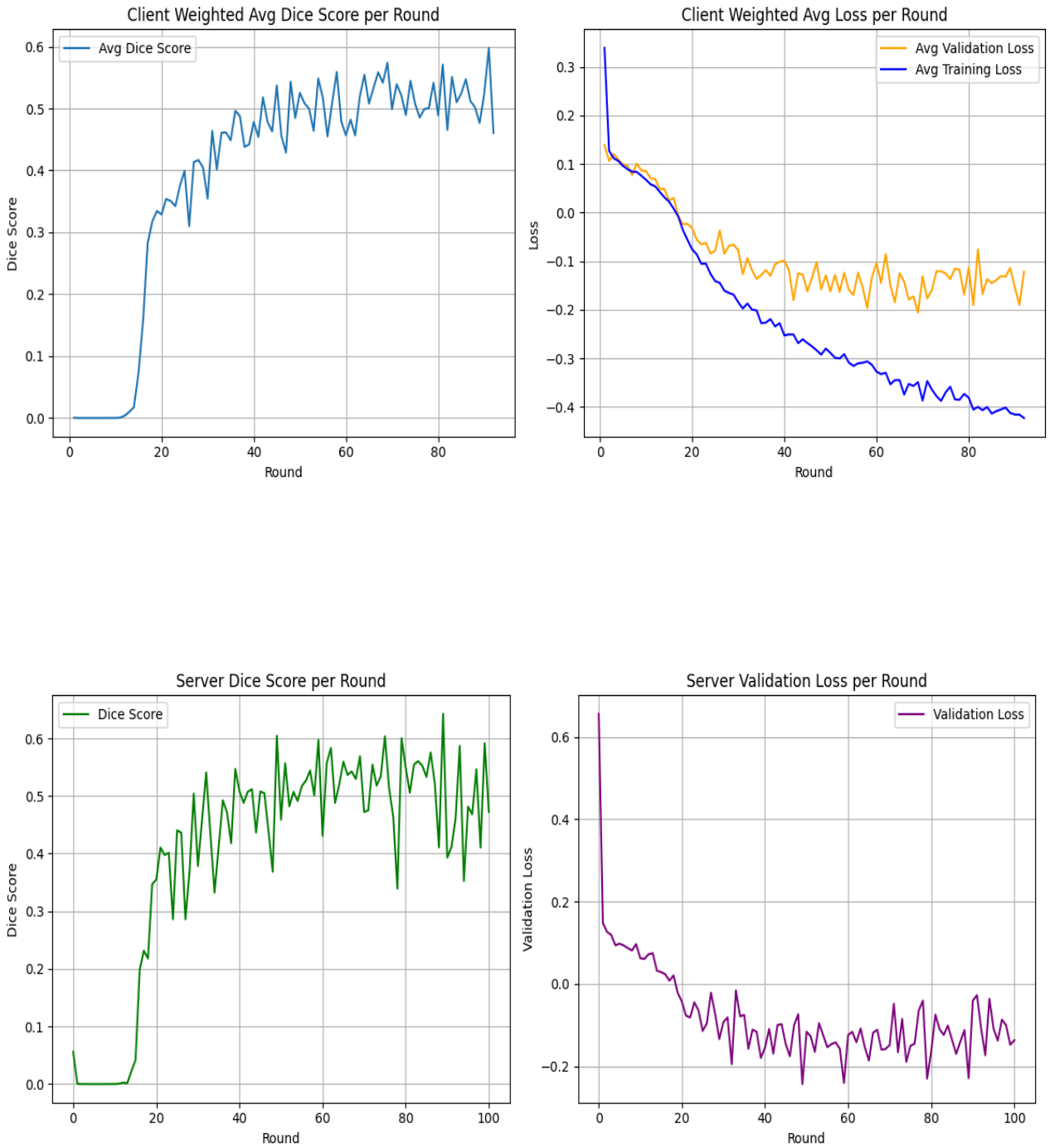



Figure: nnU-Net (Federated) Trail-2 Results

This trial involved the hybrid partitioning method that we mentioned earlier, and reduced the class imbalance between the pneumonia cases and normal cases, with the border of around 500 images on average, with the following partitions:

```
# Partition 0: 634 images

# Partition 1: 529 images

# Partition 2: 887 images
```

```
# Partition 3: 732 images

# Partition 4: 670 images

# Partition 5: 584 images

# Partition 6: 717 images

# Partition 7: 667 images

# Partition 8: 588 images

# Partition 9: 592 images
```

And the hyperparams used were:

a. MODEL: PlainConvUNet
b. EPOCHS = 240
c. BATCH SIZE = 12
d. LEARNING RATE = 0.01
e. OPTIMIZER = SGD with MOMENTUM = 0.9
f. LR Scheduler = Polynomial decay: with exp=0.9 (Doesn't work)

The validation curve seems to overfit as the learning rates aren't optimized and every round the models state dict is being loaded and saved after every round, excluding the hyperparams, hence every round acts as the Epoch 0 for each client.

---

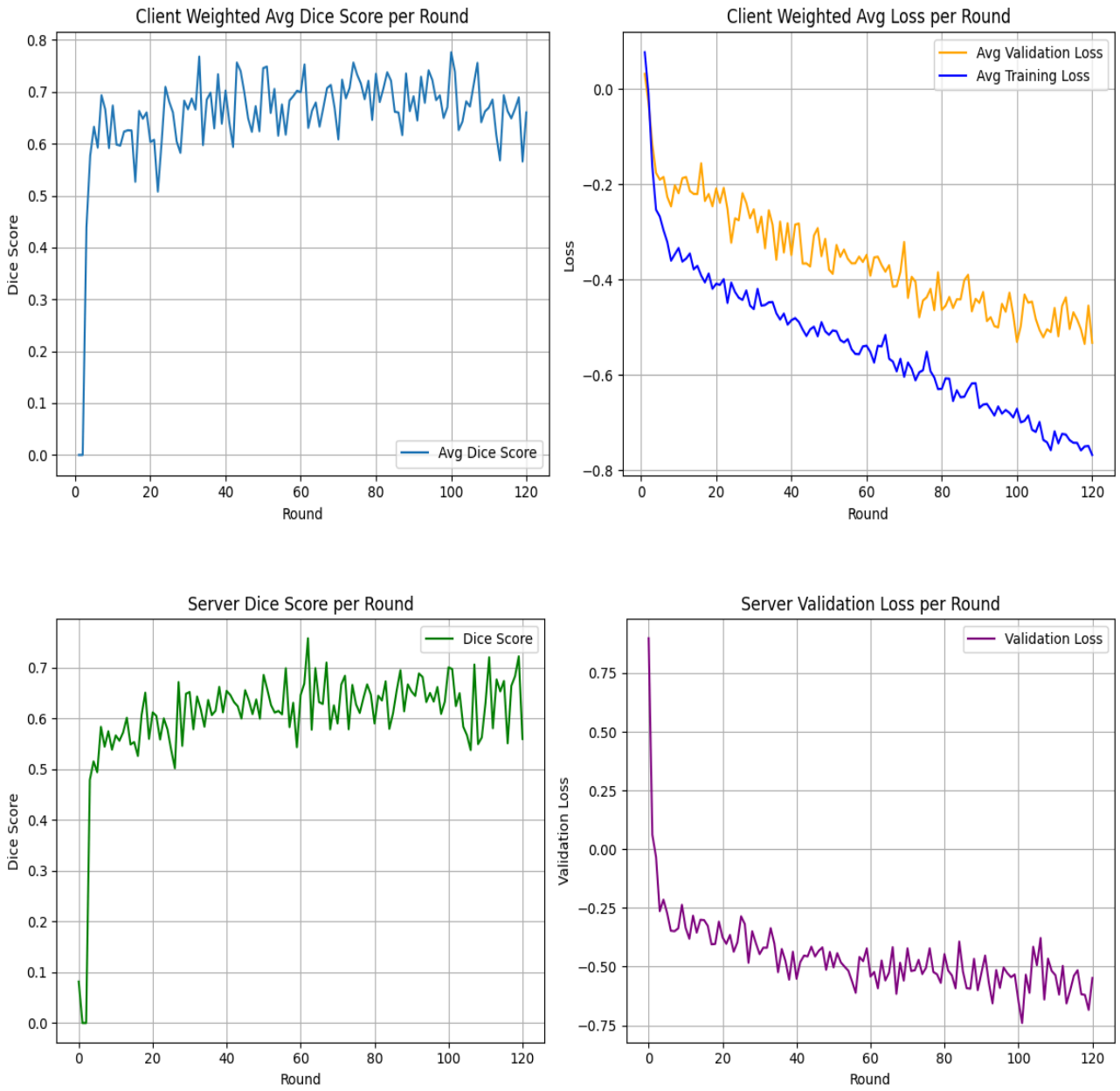



Figure: nnU-Net (Federated) Trail-3 Results

This trial involved the hybrid partitioning method that we mentioned earlier, and reduced the class imbalance between the pneumonia cases and normal cases, with the border of around 400 images on average, with the following partitions:

```
Partition 0: 360 images

Partition 1: 401 images

Partition 2: 493 images

Partition 3: 303 images

Partition 4: 308 images

Partition 5: 302 images
```

```
Partition 6: 294 images

Partition 7: 301 images

Partition 8: 294 images

Partition 9: 500 images

Partition 10: 577 images (Test set)
```

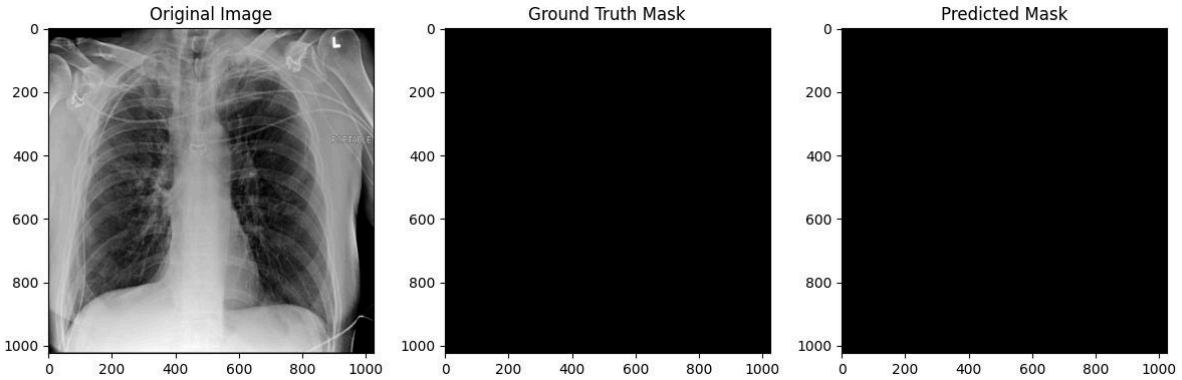
And the hyperparams used were:

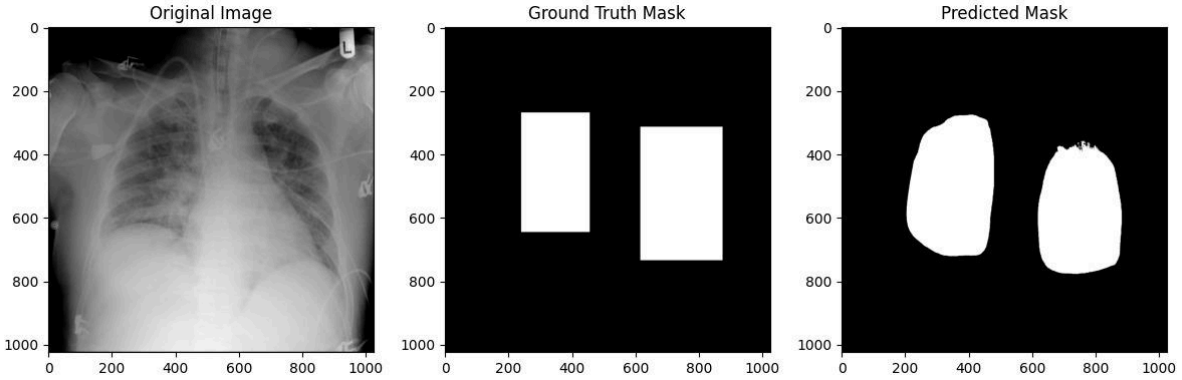
g. MODEL: PlainConvUNet
h. EPOCHS = 240
i. BATCH SIZE = 12
j. LEARNING RATE = 0.01
k. OPTIMIZER = SGD with MOMENTUM = 0.9
l. LR Scheduler = Polynomial decay: with exp=0.9 (Custom)

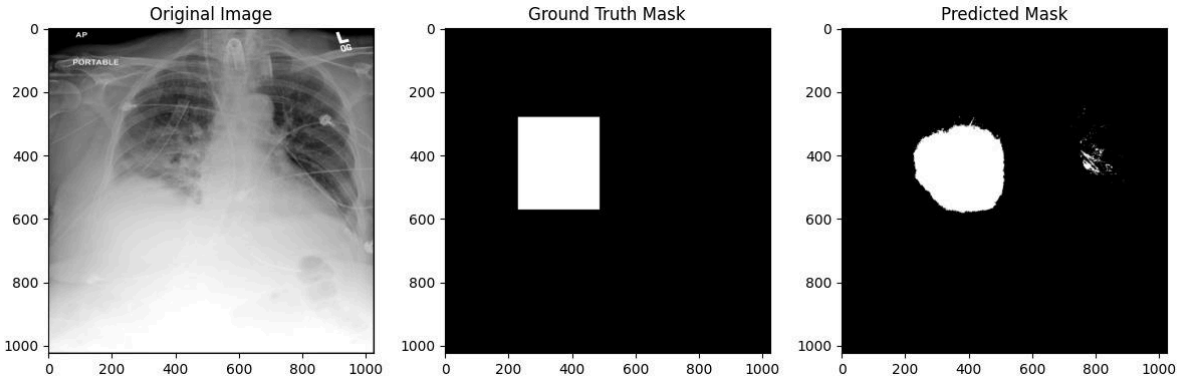
The validation curve seems to be best fit as the learning rates are optimized and every round the models state dict is being loaded and saved after every round along with the optimizers and loss functions state dict, including the hyperparams, and the LR of each round is scheduled by the global LR Scheduler which is at the server (Custom with the same formula). Hence every round acts as the Epoch of the round number for each client. This resulted in a model having a dice score of around 0.74 on the server side.

---

# nnU-Net (Federated) Results



| Model | Centralized Accuracy/Dice | Federated Accuracy/Dice |
|---|---|---|
| ResNet50 | 96% / - | 91% / - |
| nnU-Net | - / 0.824 | - / 0.74 |

---

# Key Observations

---

• **Model Performance**:  ResNet50 outperformed ResNet34 in both centralized and federated settings.

• Federated ResNet50 showed promising accuracy, though slightly lower than centralized due to data heterogeneity.

• **Segmentation Quality**:  nnU-Net demonstrated better Dice scores and generalization capability in both training modes.

• Federated training showed minor performance drops but maintained consistent trends across clients.

• **Partition Strategy:** Hybrid partitioning balanced non-IID distribution and class diversity effectively.

# Conclusion

---

• Federated Learning enables privacy-preserving model training for pneumonia detection and segmentation.

• Flower framework effectively simulated FL on multiple clients with scalable and flexible configurations.

• ResNet and nnU-Net adapted well to federated settings, proving that high-performing Medical models can be trained without centralized data.

• Hybrid partitioning strategy helped bridge the gap between real-world data distribution and training stability.

# Future Work

---

• Integrate Differential Privacy for stronger privacy guarantees.

• Expand the number of clients and evaluate real-world FL deployment across hospitals.

• Automate hyperparameter tuning using tools like Optuna in a federated context

# Weekly Progress

---

### Week 1 (Jan 1 - Jan 7, 2025)

- Contacted Professor and mentor for guidance on the project.
- Discussed initial ideas and project goals in detail.

### Week 2 (Jan 8 - Jan 14, 2025)

- Studied the fundamentals of federated learning theory.
- Explored the Flower framework to understand its features and use cases.

### Week 3 (Jan 15 - Jan 21, 2025)

- Completed the implementation of a basic centralized model for pneumonia detection.
- Implemented ResNet34 and ResNet50 classifier code with the centralised dataset.
- Verified the functionality of the initial model.

### Week 4 (Jan 22 - Jan 28, 2025)

- Explored other federated learning methods via public repositories.
- Updated the ResNet50 classifier code with the Flower framework, implementing one server and one client.
- Faced issues with `flwr.simulation.start_simulation` on Kaggle; resolved it by using `run_simulation`.

### Week 5 (Jan 29 - Feb 4, 2025)

- Worked on a multiple-client approach, which is almost functional.
- Encountered issues with the `get_parameters` and `set_parameters` functions in the client class as per Flower documentation.
- Fixed parameter loading issues by using `torch.from_numpy` to maintain consistent data types.

### Week 6 (Feb 5 - Feb 11, 2025)

- Resolved batch normalization layer issues caused by mismatched data types (torch.float64 vs. torch.float32).
- Identified data limitations: currently using a single-source dataset, leading to only ~160 images per client for 10 clients.

- Planned to train with disjoint datasets for better distribution.

### Week 7 (Feb 12 - Feb 18, 2025)

- No progress due to quizzes.

### Week 8 (Feb 19 - Feb 25, 2025)

- No progress due to mid-semester exams.

### Week 9 (Feb 25 - Mar 2, 2025)

- Implemented Hybrid Partition to balance data heterogeneity and class coverage across clients.
- Integrated FedAvgM (Federated Averaging with Momentum) to improve model convergence.
- Removed dynamic dataset splitting from the `fit` and `evaluate` methods. Instead, used predefined `self.trainloader`, `self.valloader`, and `self.testloader` in the client class to ensure consistent data distribution.

### Week 10 (Mar 3 - Mar 10, 2025)

- During training, we implemented weighted average losses in both the `train` function and the `client class`.
- Removed the validation loader and created a validation set inside the `fit` function in the `client class`.Applied the same weighted average loss calculation for validation.
- Plotted the weighted average training and validation losses along with the server loss for better performance analysis.
- Additionally, added momentum to the SGD optimizer to enhance convergence speed.

### Week 11 (Mar 11 - Mar 18, 2025)

- Reviewed nnU-Net documentation to understand its preprocessing and training pipeline.
- Preprocessed and formatted 20% of the dataset to ensure compatibility with nnU-Net for training.
- Reviewed reference papers and created a comparison table based on various metrics.

### Week 12 (Mar 19 - Mar 26, 2025)

- Performed segmentation using nnU-Net for a centralized approach with a 2K image subset of the original dataset.

### Week 13 (Mar 27 - April 2, 2025)

- Reviewed segmentation outputs of nnU-Net of centralized approach. On review, dice score of 0.88 was established before overfitting for 223 epochs via early stopping
- Started work on pre data processing on the dataset for federated approach.

### Week 14 (April 3 - April 9, 2025)

- Preprocessing and validation of partitioned dataset (uniform) in terms of nnU-Net v1 was completed.
- Ran a simulation with 4 clients as a test in Kaggle with batch size 6 (15 GB capacity GPUs) uptill 56 rounds (Kaggle time limit).
- Moved this dataset (10 uniform partitions) to the VGX GPU server.
- Made sure the new dataset is validated by nnU-Net v2 (which accepts .png format also)

### Week 15 (April 10 - April 16, 2025)

- The moved script was tested and made sure it ran on multiple GPUs.
- Allocated specific gpus to be used by nnU-Net v2.
- Made a new dataset with hybrid partitioning and ran a 100 round FL training sessions with 9 clients and 1 server test dataset and produced inferences from the test dataset. And got a dice score of 0.642.

### Week 16 (April 17 - April 23, 2025)

- No progress due to end-semester preparation.

### Week 17 (April 24 - April 30, 2025)

- No progress due to end-semester examinations.

### Week 18 (May 1 - May 7, 2025)

- Created custom LR Scheduler and set nnU-Net's LR Scheduler as None
- Ran a 100 round FL training sessions with 9 clients and 1 server test dataset and produced inferences from the test dataset. And got a dice score of 0.674.
- Modified the datasets to include more positive cases
- Ran a 120 round FL training sessions with 10 clients and 1 server test dataset and produced inferences from the test dataset. And got a dice score of 0.704.

# Review Meets

---

### Meet 1 (Jan 17, 2025)

- Discussed ideas and fixed the project objectives.

### Meet 2 (Jan 23, 2025)

- Model based on ResNet50 arch. was shown, with accuracy of 96%
- Dataset wasn't used completely, due to which the model stagnated with 52% accuracy.
- Solution for the problem was to be researched.

### Meet 3 (Feb 11, 2025)

- Suggested using the Dirichlet distribution for better data partitioning among clients.
- Recommended reducing the number of epochs per client while increasing the number of clients to improve generalization.
- Advised expanding the dataset by merging multiple datasets to address data scarcity issues.

### Meet 4 (Mar 05, 2025)

Discussed the implementation of the following changes as suggested in the previous meeting:

- During training, calculate weighted average losses in the `train` function and `client class`.
- Remove the validation loader and create a validation set inside the `fit` function in the `client class`.
- Apply the same weighted average loss calculation for validation.
- Plot the weighted average training and validation losses along with the server loss for better performance analysis.

### Meet 5 (Mar 20, 2025)

- Suggested starting with a subset of 2K to 6K images instead of using the full 28K image dataset for nnU-Net segmentation.

### Meet 6

- Reviewed the nnU-Net centralized approach and cleared on the Loss function misconceptions used by the nnU-Net library.

### Meet 7

- Reviewed the FL approach with nnU-Net and insisted to do hyperparam tuning for LR with Optuna, met another PhD Scholar having experience in Optuna framework and insisted us to use a global LR Scheduler for the 1st trial, and it suffices tuning of LR.

# Difficulties Faced

---

## Model accuracy stagnation

- Achieved 96% accuracy in the centralized ResNet50 model, but it dropped to 52% in the federated setup due to incomplete dataset utilization.
- Started researching solutions to improve the model's performance.

## Multiple-client parameter synchronization

- Faced issues with get_parameters and set_parameters functions due to data type mismatch (torch.float64 vs. torch.float32).
- Fixed it using torch.from_numpy() for consistent data types across clients.

## Batch normalization layer issues

- The batch normalization layer caused inconsistent learning across clients due to varying batch sizes.
- Resolved it by using predefined datasets and adjusting batch normalization settings.

## Limited dataset

- Only 160 images per client led to poor generalization of the model.
- Planned to expand the dataset by merging multiple open-source datasets or with new large datasets.
- Finally shifted to one dataset with 26 thousand images (RSNA dataset).

## Data distribution imbalance

- Some clients received near zero samples for certain classes due to non-IID data distribution.
- Solved it by implementing a hybrid partition ensuring class coverage.

## Slow model convergence

- Model convergence was slow while increasing the number of clients.
- Improved it by using FedAvgM with momentum in the SGD optimizer.

## Dynamic dataset splitting issues

- Dynamic splitting of training and validation sets caused inconsistent results.

- Fixed it by using predefined self.trainloader, self.valloader, and self.testloader in the client class.

### Weighted average loss calculation

- Faced challenges in aggregating losses from clients, to generate inference on the performance of the clients.
- Implemented weighted average loss for both training and validation.

### Validation loader removal

- Validation loader produced inconsistent results across rounds.
- Replaced it with a validation set inside the fit function for consistent evaluation.

### Loss plotting issue

- Difficulty in visualizing consistent loss patterns from different clients.
- Solved it by plotting weighted average train, validation, and server losses.

---

## nnU-Net Segmentation

### Dataset Preprocessing Challenges

- Faced difficulties in formatting the dataset to comply with nnU-Net requirements.
- Resolved by adhering to the nnU-Net preprocessing pipeline and ensuring the correct folder structure and naming conventions.

### Large Dataset Handling

- The full dataset of 26,000 images was too large for initial training.
- To manage training and evaluation efficiently, started with a subset of 2,000 to 6,000 images.
- Initial trials used a partition of 200 images.
- After validation, it adopted the same **hybrid partitioning** strategy used in the classification task, resulting in ~500 images per client.
- Detected class imbalance among clients—some had significantly more or fewer pneumonia cases.
- Addressed this by removing unmasked images from certain partitions, leading to a more balanced distribution.

## nnU-Net Framework and Integration Issues

- The nnU-Net framework requires specific environment variables, but these weren't accessible due to Flower (FLWR) framework's multiprocessing and forking behavior.
- Resolved by creating a wrapper script that exports the necessary environment variables before launching FLWR processes, ensuring child processes inherit them.
- Models were getting re-initialized in every round.
- Resolved by modifying the `Client.fit()` and `Client.evaluate()` methods to load checkpoints at the start of each round and save them afterward.
- Saved client models separately to maintain state across federated rounds.

## Server-Side Evaluation and Trainer Issues

- Server-side evaluation created a new independent process, which again faced environment variable issues with the `nnUNetTrainer` class.
- These were fixed using the same script-based environment setup as the clients.
- The `LR scheduler` in the `nnUNetTrainer` was stepping at epoch 0 every round, which disrupted learning rate adjustments.
- Local tuning of hyperparameters at each client did not yield effective results.
- Migrated learning rate control to the **server side**, implementing a global learning rate scheduler that adjusted the LR values centrally.

## Dataset Partition Adjustments

- Observed that certain client partitions had very few pneumonia cases, while others had many.
- Modified the partitioning strategy by reassigning data to reduce class imbalance across clients.

# Comparison table

| Aspect | Paper 1: Blockchain-Based Medical Systems | Paper 2: Federated Learning for Pneumonia Detection | Paper 3: Federated Learning in IoT | Paper 4: Secure Federated Learning for Healthcare | Paper 5: Pneumonia Detection Using Unsupervised Learning |
|---|---|---|---|---|---|
| **Primary Objective** | Enhance security in medical systems through blockchain | Detect pneumonia using federated learning | Optimize IoT device learning | Improve privacy and performance in healthcare FL | Use unsupervised learning for pneumonia detection |
| **Learning Approach** | Blockchain with FL | Federated Learning (FL) with CNN | Federated Learning (FL) | Federated Learning with Secure Aggregation | Unsupervised Learning with Autoencoder |
| **Data Privacy Focus** | Blockchain secures patient data | Local training without sharing data | Minimize data transfer | Enhanced data security | Local training with Autoencoder |
| **Dataset Used** | Medical Records | Chest X-ray Images | IoT Sensor Data | Various Healthcare Datasets | Chest X-ray Images |
| **Accuracy Achieved** | Not explicitly mentioned | 94.82% | Around 92% | 95.32% on X-rays, 96.65% on Symptoms | Not explicitly mentioned |

| | | | | | |
|---|---|---|---|---|---|
| **Key Contributions** | Blockchain-based security | Pneumonia detection through FL | Optimize IoT device learning | Secure data sharing and model performance | Improved pneumonia detection without labeling |
| **Limitations** | High computational cost | Requires high-quality X-ray images | Dependency on IoT devices | High computational resources required | Limited training data availability |
| **Year of Publication** | 2024 | 2024 | 2023 | 2023 | 2024 |
| **Learning Rate** | Not mentioned | 0.001 | 0.01 | 0.001 | 0.0001 |
| **Batch Size** | Not mentioned | 32 | 64 | 32 | 16 |
| **Optimizer Used** | Not mentioned | Adam | SGD | Adam | Adam |
| **Number of Epochs** | Not mentioned | 50 | 100 | 50 | 30 |
| **Activation Function** | Not mentioned | ReLU | Sigmoid | ReLU | ReLU |
| **Number of Clients** | Not mentioned | 75 | 100 | 150 | 50 |
| **F1-Score** | Not mentioned | 0.91 | 0.89 | 0.93 | 0.88 |
| **Precision** | Not mentioned | 0.90 | 0.88 | 0.92 | 0.86 |

| | | | | | |
|---|---|---|---|---|---|
| **Recall** | Not mentioned | 0.92 | 0.90 | 0.94 | 0.89 |