Towards Decentralized Training of Machine Learning Models for Bone Cancer Detection

Members:

- Quan Nguyen (Capstone Mentor)
- Hina Bandukwala
- Karan Khubdikar
- Prabhjit Thind
- Salva Umar

Table of Content

1. Executive Summary

Centralized data storage presents significant challenges to model training, primarily due to concerns over data privacy, distribution across multiple institutions, and high communication overheads. Recognizing these challenges, our capstone partner, ALS GoldSpot Discoveries Ltd., was interested in exploring Federated Learning (FL) for data-driven mineral exploration. FL is a novel approach that enables machine learning models to be trained on a multitude of local datasets without requiring the data to be uploaded to a central server. This methodology not only enhances privacy and security but also reduces the dependency on extensive data communication networks [10].

This project utilizes the decentralized framework `Flower` and transfer learning with the `ResNet-34` model to develop a multi-class image classification model. We used an open-source Osteosarcoma dataset comprising 1,144 images across three classes, *Non-tumour*, *Non-Viable-Tumour* and *Viable-Tumour*. This dataset was chosen due to its resemblance to the class imbalance and data heterogeneity commonly found in the mining industry.

The primary objective of the project was to overcome the limitations of centralized data storage by evaluating the efficacy of Federated Learning compared to traditional centralized training methods. To this end, we developed a FL framework that can be used to simulate a multi-client scenario by dividing the training dataset across clients, where each client trains its model independently. The parameters are then aggregated to update the global model. In our developed framework, we achieved comparable balanced accuracy and F1-scores when compared to our baseline centralized model.

Overall, this project seeks to evaluate the efficacy of Federated Learning compared to traditional centralized training methods. The ultimate goal was to demonstrate that Federated Learning can achieve comparable model performance while addressing the inherent limitations of centralized data storage, and provide ALS GoldSpot Discoveries ltd. with a Federated Learning framework that can increase the efficiency of training ML models in their operations.

2. Introduction

Federated Learning (FL) allows machine learning models to be trained across multiple decentralized devices or servers, keeping data localized. This approach mitigates challenges associated with centralized data storage, ensuring enhanced privacy and reducing communication overhead. In FL, models are trained collaboratively across multiple devices or nodes without requiring data to be centralized. Each node computes

updates to the model using its local data, and only the model updates are shared with a central server, which aggregates these updates to improve the global model. This decentralized method offers several advantages [1].

Firstly, FL enhances privacy since data remains on local devices, reducing the risk of data breaches and ensuring compliance with privacy regulations. Secondly, only model updates, not raw data, are transmitted, significantly lowering the amount of data exchanged and reducing communication overhead. Additionally, FL allows for personalized models by training on local data, making it adaptable to diverse environments. Finally, FL helps in adhering to data protection laws like GDPR by avoiding centralized storage of sensitive data.

2.1 Data Description

The project utilizes the Osteosarcoma dataset [7][11], containing 1,144 histology images categorized into *Non-tumour*, *Non-Viable-Tumour* and *Viable-Tumour* classes (Figure 1). This dataset mirrors the class imbalance and data heterogeneity prevalent in mining exploration, making it ideal for evaluating the efficacy of decentralized learning frameworks.

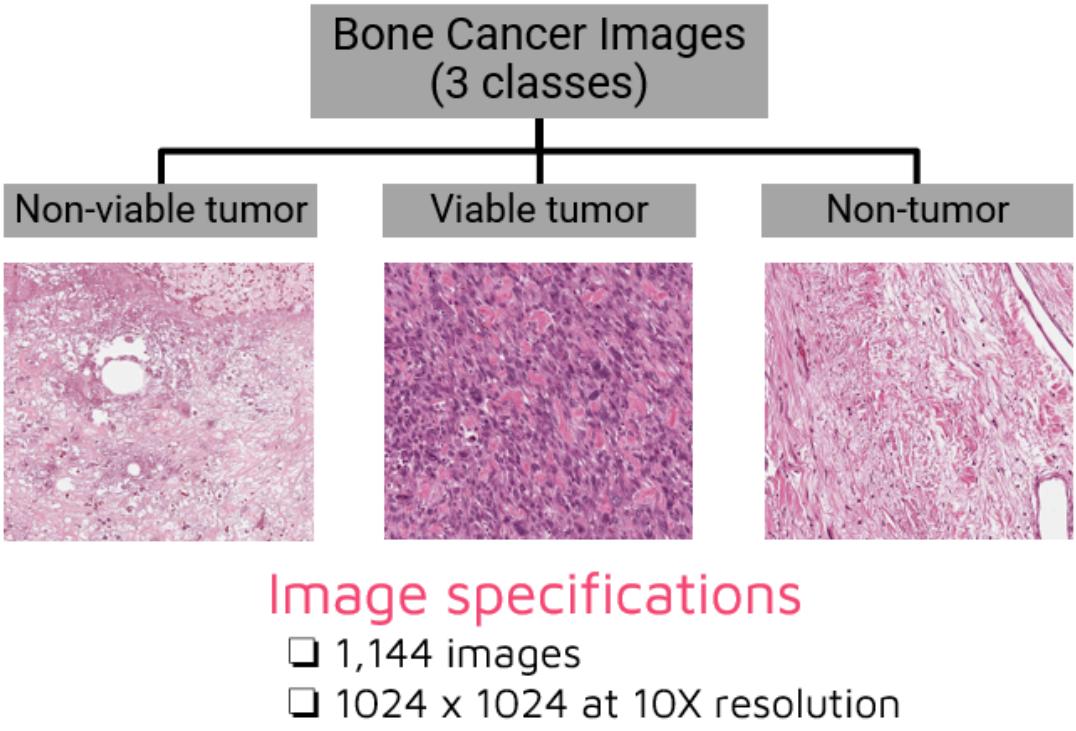


Figure 1. Osteosarcoma Dataset Overview

2.2 Scientific Objectives

This project aims to investigate the practical uses and benefits of decentralized learning while examining its key challenges, including communication efficiency and data diversity. We trained an image classification models using the Osteosarcoma dataset in both centralized and decentralized frameworks. The goal is to evaluate the effectiveness of the proposed decentralized approach to the model training process.

- Develop and implement a Federated Learning framework using Flower and transfer learning with `ResNet-34` for multi-class image classification.
- Evaluate the framework's performance in handling data heterogeneity and optimizing communication efficiency across multiple client nodes.

2.3 Tech Stack

- **Python and PyTorch:** Primary language and deep learning framework [9] for model development and training.
- **Flower:** Decentralized framework for Federated Learning, enabling communication between client nodes [6].
- **Hydra:** Configuration manager for customizing model parameters and experimental setups efficiently [12].
- **Ray:** Parallel programming framework for managing concurrent client model training and optimizing resource use [8].

This tech stack supports scalable, efficient, and robust federated learning implementations tailored for mining industry applications.

2.3.1 Advantages of Federated Learning using Flower

When comparing FL to centralized training, FL clearly stands out in several ways. It keeps data on local devices, enhancing data privacy. By sharing only model updates rather than raw data, it reduces bandwidth usage. Furthermore, FL allows for personalized models by training on local data and helps adhere to data protection laws by avoiding centralized storage of sensitive data.

The decentralized framework Flower is particularly advantageous for implementing FL. Flower supports multiple machine learning frameworks like TensorFlow, PyTorch, and Scikit-learn, providing greater flexibility and ease of integration compared to more specialized frameworks like PySyft and TensorFlow Federated. Flower also offers user-friendly APIs and comprehensive documentation, making it easier to scale federated learning experiments across various platforms and devices, whereas PySyft and TensorFlow Federated can be more complex to set up and manage. Additionally, Flower has a growing community and actively maintained ecosystem, fostering collaborative development and rapid iteration, which can be more robust and versatile compared to the more niche communities around PySyft and TensorFlow Federated.

2.4 Project Deliverables

- Detailed project report encompassing methodology, experiments, results, and conclusions.
- Comprehensive Git repository housing code, documentation, configurations, and results for reproducibility.

- Data pipeline outlining processes for effective data collection, processing, and preparation to maintain integrity and consistency throughout the project.

3. Data Science Methods

In our exploration of Federated Learning (FL) for enhancing model training efficiency in the mining industry, we have followed a systematic approach to develop and evaluate our Federated Learning Framework (Figure 2).

3.1 Overall Data Science Workflow

Our methodology can be broken down into several key steps:

1. **Data Augmentation:** Initially, we applied various data transformations to augment our dataset, effectively increasing its size to improve robustness and enhance the model's ability to generalize across diverse mining datasets.

2. **Centralized Training Workflow:** We began by establishing a baseline using a centralized training approach to benchmark our Federated Learning framework's performance. This step provided a comparative measure for evaluating the effectiveness of decentralized model training.

3. **Data Partitioning:** Our next critical step involved partitioning the dataset into client nodes, exploring both Independent and Identically Distributed (IID) and non-IID partitioning strategies. This segmentation was essential to simulate real-world scenarios where data distribution among clients can vary significantly.

4. **Simulation Framework:** Each client node independently trained a ResNet-34 model on its partition of the dataset. This decentralized training process allowed us to assess how effectively our framework handled distributed data and training across multiple nodes.

5. **Model Aggregation:** The weights trained by each client were then aggregated using four distinct strategies within our Federated Learning framework. These strategies enabled us to consolidate insights from individual clients while preserving data privacy and minimizing communication overhead.

6. **Evaluation:** Evaluate the global model using Balanced Accuracy, confusion matrices, and precision-recall curves.

Throughout these steps, we maintained a focus on optimizing communication efficiency, managing data heterogeneity, and ultimately demonstrating the utility of Federated Learning. Our methodology not only addresses the specific needs of ALS GoldSpot Discoveries Ltd. but also contributes to broader advancements in decentralized machine learning applications across diverse industrial domains.

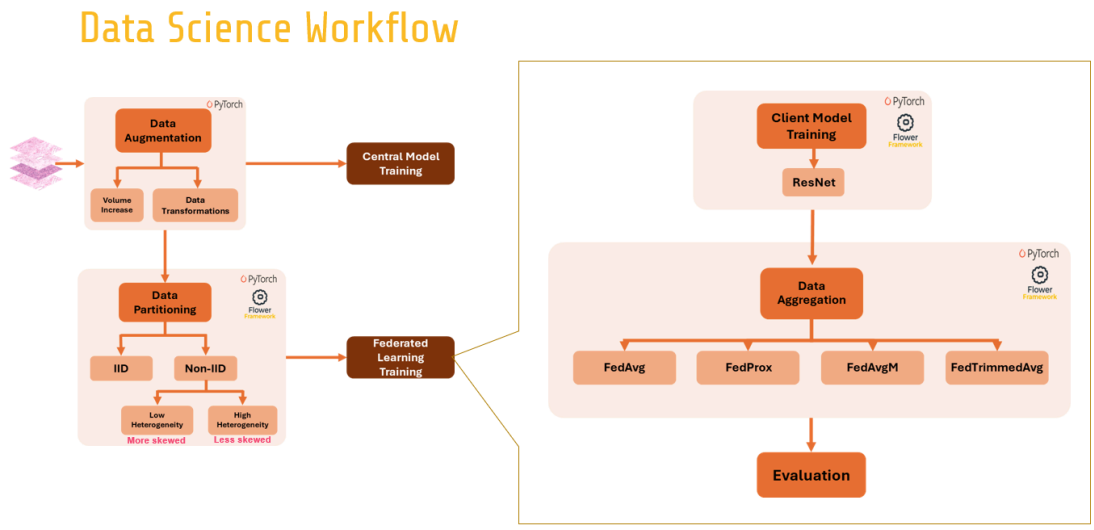


Figure 2. Data Science Workflow

3.2 Data Preprocessing

Given the smaller size of the training dataset, various data transformation and augmentation techniques were utilized to increase the dataset's volume and diversity.

Data Transformations: To enhance the training images, the following transformations were applied:

- **Random Horizontal Flip**: Flips images horizontally with a 50% probability.
- **Random Rotation**: Rotates images randomly by up to 10 degrees.
- **Random Cropping**: Crops up to 20% of the images to introduce variability.
- **Color Jitter**: Adjusts image brightness, contrast, saturation, and hue to simulate different lighting conditions.

Data Augmentation: These transformations were applied systematically to double the size of the training dataset, effectively increasing it by 100%. This augmentation helps in improving the model's robustness and performance by providing a more varied training set.

3.3 Central Model

The augmented dataset was utilized to train the central model. We selected the following models due to their suitability for multiclass image classification and their ability to capture spatial hierarchies in images [11]. For each model, the final few layers were unfrozen, and fine-tuning was performed:

- **ResNet-34**
- **MobileNet-v2**
- **VGG-16**

These models were chosen based on their proven effectiveness in image classification tasks, allowing us to leverage their pre-trained weights and adapt them to our specific dataset through fine-tuning.

3.4 Data Partitioning

To partition the data for different clients in the Flower framework, we have considered two scenarios to account for the variability in the datasets that our capstone partner might encounter:

1. **IID Partitioning**:

   Each client receives a random sample from the training set, ensuring that the data distribution remains similar across all clients. This is implemented using PyTorch's `random_split` function.

2. **Non-IID Partitioning**:

   We utilized the Flower framework's partitioning functionality, which employs the Dirichlet distribution to control the skewness of the class distribution among clients. The concentration parameter α of the Dirichlet distribution determines the degree of skewness. Lower α values (≤ 0.5) result in a more skewed distribution, while higher values lead to a more balanced distribution. For our experiments, we considered α values of 0.1 and 0.5 (Figure 3).

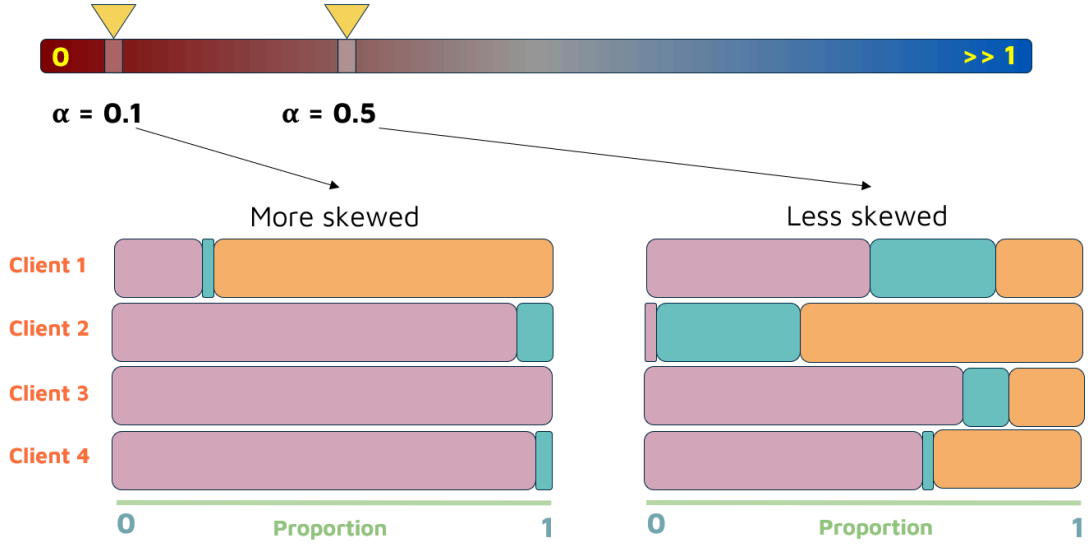


Figure 3. Non-IID Data Partitioning using the Dirichlet distribution with varying alpha values.

3.5 Data Aggregation

To consolidate the weights from different clients at the server/global level, we considered the following four aggregation strategies. These strategies were chosen to cover various distributions of datasets:

1. **Federated Averaging (FedAvg):** Computes a weighted average of the local updates from each client. It is ideal for Independent and Identically Distributed (IID) data. Each client k trains a local model w_k using its local dataset. This involves performing several iterations of stochastic gradient descent (SGD) on the local data. After local training, each client sends its local model updates (or parameters) back to the central server. The central server aggregates these updates by computing a weighted average. The formula for updating the global model w is:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} w_k$$

   where w_{t+1} is the updated global model parameters, K is the total number of clients, n_k is the number of data samples on client k, n is the total number of data samples across all clients, $n = \sum_{k=1}^{K} n_k$, w_k is the local model parameters from client k [2].

2. **Federated Proximal (FedProx):** Extends FedAvg by incorporating a momentum parameter, which accelerates convergence by considering past gradients. This addition helps mitigate the issues arising from non-IID (non-independent and identically distributed) data among different clients and system heterogeneity by ensuring that the local updates do not drift too far from the global model parameters. Ideally, this strategy is better for non-IID data. Local training and sending updates steps are similar to those described above in FedAvg. The server aggregates the updates with a proximal term to penalize large deviations from the previous global model:

$$w_{t+1} = \arg\min_{w} \left\{ \sum_{k=1}^{K} \frac{n_k}{n} \left(f_k(w) + \frac{\mu}{2} \|w - w_t\|^2 \right) \right\}$$

where μ is a regularization parameter and w_t is the previous global model [3].

3. **Federated Averaging with Momentum (FedAvgM):** Enhances FedAvg by incorporating momentum to stabilize the global model aggregation process. It adjusts the aggregation formula by adding a momentum term that includes a weighted average of previous global model states, improving convergence, especially in the presence of heterogeneous data. Works for both IID and non-IID data. Local training and sending updates steps are similar to those described above in FedAvg. The server aggregates the updates with momentum to stabilize the global model:

$$w_{t+1} = \sum_{k=1}^{K} \frac{n_k}{n} (\beta w_{t,k} + (1 - \beta) w_k)$$

where β is the momentum parameter, $w_{t,k}$ is the momentum term from the previous iteration [4].

4. **Federated Trimmed Averaging (FedTrimmedAvg):** Improves FedAvg by computing a trimmed mean of model updates, excluding extreme values from clients. This approach is effective for handling non-IID data with outliers, as it prevents these outliers from disproportionately influencing the global model, thereby improving robustness and convergence. Better for Non-IID data with outliers. Local training and sending updates steps are similar to those described above in FedAvg. The server aggregates updates by computing a trimmed mean, discarding extreme values:

$$w_{t+1} = \frac{1}{K - m} \sum_{k \in \mathcal{S}_t} w_k$$

where $\mathcal{S}_t$ is a subset of clients selected at iteration t, and m is the number of clients with the highest and lowest updates excluded [5].

3.6 Evaluation Metrics

We compared our models trained using centralized training and federated learning using several key metrics:

Metric	Description
Balanced Accuracy	The average recall obtained on each class, provides an unbiased performance measure for imbalanced classification datasets.
Precision	Measures the correctness achieved in classification by indicating the proportion of true positive instances among the retrieved instances.
Recall	Measures the ability to find all relevant cases within a dataset, reflecting the model's effectiveness in identifying positive instances.

Metric	Description
F1-Score	The harmonic mean of precision and recall, offering a comprehensive view of the model's performance by balancing the trade-off between precision and recall.
Confusion Matrix	A visualization of the model's performance across different classes, helping us understand whether the number of false positives and false negatives was within an acceptable range.

Table 1. Evaluation Metric Definitions

These metrics collectively offered a thorough evaluation of the models' performance, allowing us to assess the effectiveness of the federated learning approach compared to centralized training.

4. Results & Discussion

4.1 Central Model

We began our analysis by augmenting our dataset, as detailed in section 3.2. This augmentation process allowed us to double our dataset size, thereby increasing variability and improving model generalizability. As illustrated in Figure 4, this augmentation led to a 10% increase in both training and validation accuracies.

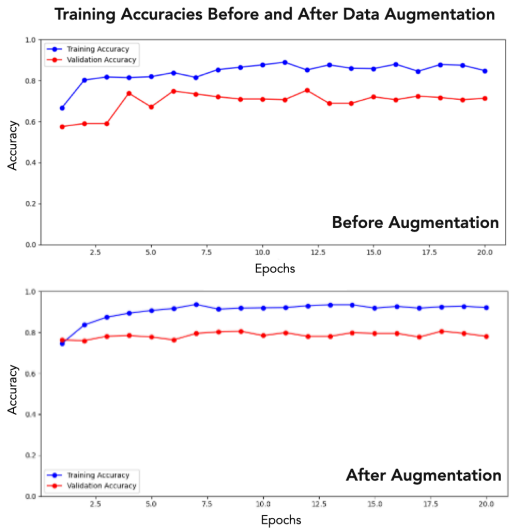


Figure 4. Training and validation accuracies before and after data augmentation.

Our model selection process was conducted within a centralized framework. We tested three models: `ResNet-34`, `VGG16`, and `MobileNetV2`. `ResNet-34` was selected based on its highest overall balanced accuracy and F1 scores. To minimize overfitting, we implemented a decaying learning rate. Among the models, `ResNet-34` was the most accurate in identifying viable tumors with minimal errors, although there was noticeable confusion between non-tumor and non-viable tumor classes. This indicates a potential area for model improvement, possibly by refining the significant features or rebalancing the training data. Additionally, the non-tumor class exhibited the highest number of false

positives, suggesting possible overfitting to features common in other tumor classes. The central model confusion matrix is also shown below for `Resnet-34` (Figure 5).

Model	Balanced Accuracy	F1 Score
VGG-16	0.72	0.71
ResNet-34	0.78	0.79
MobileNet-V2	0.78	0.77

Table 2. Central Model Results Comparison

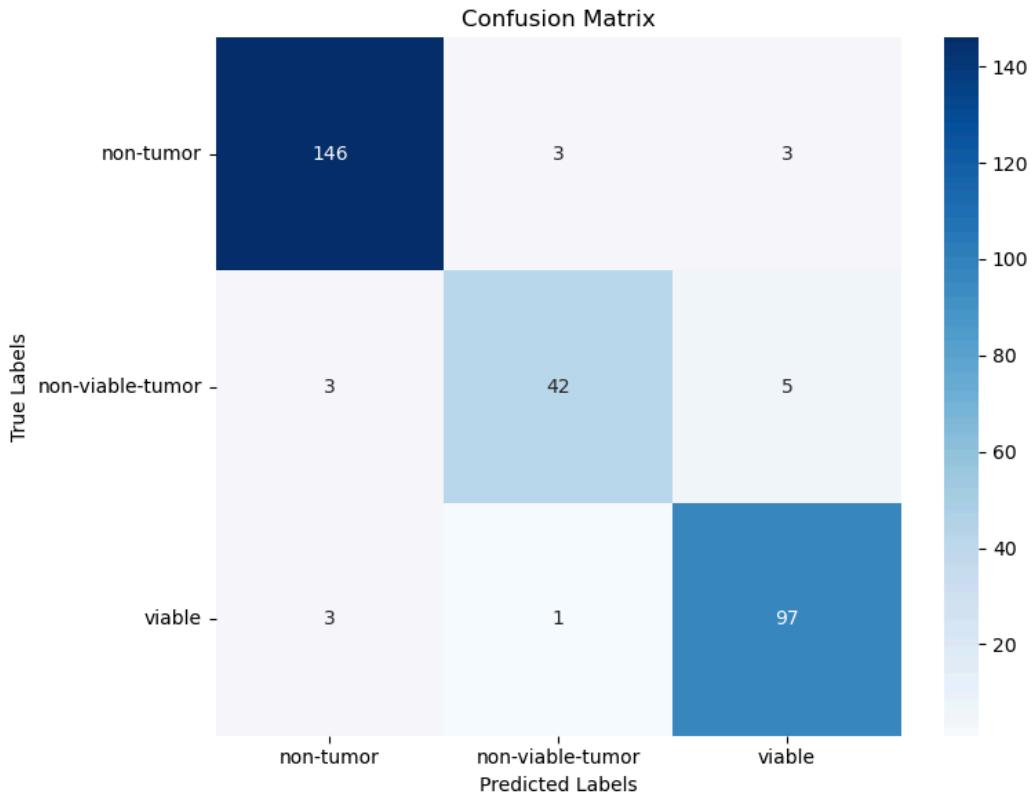


Figure 5. Confusion Matrix for Central Model

4.2 Federated Model

Using `Flower`, we developed our framework to simulate clients and partition the augmented data between them. To simulate real-world scenarios and assess their impact on federated learning, we modeled varying data distributions across clients. Data heterogeneity is a common issue in federated learning, where different clients might have different proportions of labels. To address this, we used the Dirichlet distribution with alpha values of 0.1 and 0.5 to produce skewed data distributions across clients. For IID (Independent and Identically Distributed) data, we used *random_split* from PyTorch to ensure minimal heterogeneity between clients. We then conducted 12 experiments to test the efficacy of different aggregation techniques with varying data distributions across clients. Aggregation strategies included Federated Average (FedAvg), Federated Proximal (FedProx), Federated Average M (FedAvgM), and Federated Trimmed Average (FedTrimmedAvg). All experiments were conducted with four clients (for both training and evaluation), five rounds of federated learning, and a learning rate of 0.002. Key

metrics such as global F1 score, balanced accuracy, precision, and recall were recorded. The results of these experiments are outlined in the following table and summarized in Figure 6.

Experiment	Partitioning Strategy	Aggregation Strategy	Learning Rate	Number of Rounds	Train Clients per Round	Evaluation Clients per Round	Global F1 Score Accuracy	Global Balanced Accuracy Score	Global Precision	Global Recall
1	Dirichlet(alpha=0.1)	FedAvg	0.002	5	4	4	0.79	0.84	0.81	0.80
2	Dirichlet(alpha=0.5)	FedAvg	0.002	5	4	4	0.77	0.77	0.81	0.77
3	IID (Random)	FedAvg	0.002	5	4	4	0.82	0.86	0.85	0.82
4	Dirichlet(alpha=0.1)	FedAvgM	0.002	5	4	4	0.76	0.80	0.79	0.76
5	Dirichlet(alpha=0.5)	FedAvgM	0.002	5	4	4	0.79	0.83	0.82	0.80
6	IID (Random)	FedAvgM	0.002	5	4	4	0.78	0.81	0.82	0.78
7	Dirichlet(alpha=0.1)	FedProx	0.002	5	4	4	0.83	0.86	0.85	0.83
8	Dirichlet(alpha=0.5)	FedProx	0.002	5	4	4	0.73	0.74	0.79	0.74
9	IID (Random)	FedProx	0.002	5	4	4	0.80	0.84	0.84	0.81
10	Dirichlet(alpha=0.1)	FedTrimmedAvg	0.002	5	4	4	0.77	0.81	0.78	0.77
11	Dirichlet(alpha=0.5)	FedTrimmedAvg	0.002	5	4	4	0.80	0.84	0.83	0.81
12	IID (Random)	FedTrimmedAvg	0.002	5	4	4	0.79	0.83	0.83	0.80

Table 3. Federated Learning Experiment Results

Figure 6 shows a comparison of the Global F1 Score (blue) and Global Balanced Accuracy Score (red) across different aggregation strategies (FedAvg, FedProx, FedAvgM, FedTrimmedAvg) for three partitioning strategies: IID (Random), Dirichlet(alpha=0.1), and Dirichlet(alpha=0.5). The star indicates the best score across the strategies.

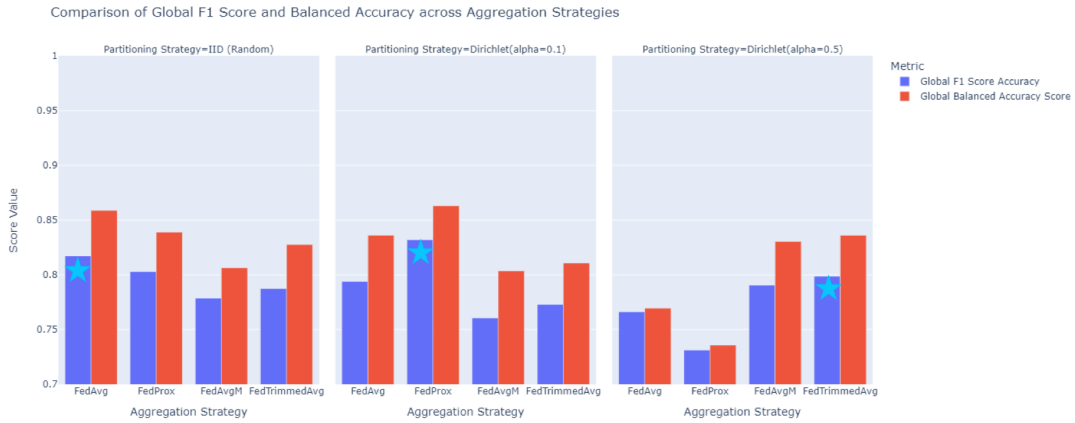


Figure 6. Comparison of global F1 score and balanced accuracies for different aggregation and partition strategies.

Our experimental results show that FedAvg works best with iid data, FedProx excelled with highly non-IID data i.e. with Dirichlet partitioning (alpha = 0.1), and finally, FedTrimmedAvg showed strong performance with moderately non-IID data i.e. with Dirichlet partitioning (alpha = 0.5). Notably, FedProx consistently excels across both skewed data scenarios (alpha=0.1 and 0.5), achieving high scores in both global F1 Score and balanced Accuracy, highlighting its effectiveness in handling non-IID data through robust regularization strategies. FedAvg shows superior performance in the IID setting, particularly in balanced accuracy, suggesting it is well-suited for balanced data distributions. FedAvgM and FedTrimmedAvg both perform commendably across scenarios, with FedTrimmedAvg notably effective in scenarios with moderate data imbalances and outlier presence, evidenced by its high F1 scores. Also, the confusion

matrices across various strategies were almost similar. Below are the confusion matrices for the FedAvg strategy Figure 7.

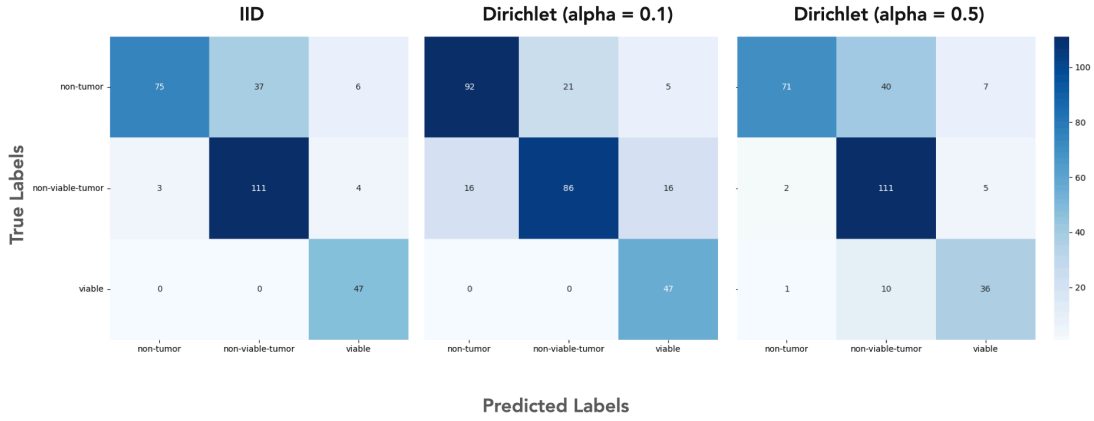


Figure 7. Confusion Matrices for FedAvg across different partition strategies.

Overall, both models showed competitive performance, suggesting that federated learning can achieve comparable results to centralized approaches while offering advantages in privacy and scalability.

5. Data Product

5.1 Product Usage

5.1.1 Make Updates Through Config File and Run Experiments:

In order for our client to test our framework, we are using *Hydra*, which allows the client to conveniently change parameters as per their requirements and dataset details. Some of the important parameters in the configuration file include:

- **data_dir**: Path to the data directory.
- **num_clients**: Number of total clients available.
- **num_rounds**: Number of Federated Learning rounds in the experiment.
- **data_partition**: The partitioning technique to be used along with the alpha value.
- **strategy_name**: The aggregation strategy to be used.

Below is the configuration file setup for one of the experiments for Federated Learning using the Osteosarcoma dataset:

```yaml
#Configurations for Federated Learning Simulation

data_dir: "./data/augmented"

batch_size: 4
class_names: ["non-tumor", "non-viable-tumor", "viable"]
input_size: 224

num_clients_per_round_fit: 4
num_clients_per_round_eval: 4
num_rounds: 5
num_clients: 4

num_cpus_per_client: 4
gpu_usage_per_client: 0.4

strategy_name: "FedAvg"
fraction_fit: 0.0
fraction_evaluate: 0.0
proximal_mu: 0.001

config_fit:
  lr: 0.002
  momentum: 0.9
  local_epochs: 1
  batch_size: 4
  decay_rate: 0.1
  step_size: 5
  gamma: 0.1

data_partition:
  strategy: "dirichlet"
  alpha: 0.5
```

Figure 8. Example base.yaml configuration file

In Figure 8, the path to the data directory is specified using *data_dir*, indicating where the training and evaluation data is located. Further, it shows that the experiment involves a total of 4 clients, each utilizing 4 CPUs and 20% of a GPU per client. Training will proceed over 5 federated learning rounds. Data partitioning follows a Dirichlet distribution with an alpha value of 0.1, ensuring a non-uniform distribution among clients. The aggregation strategy employed is FedAvg, a common technique in federated learning for averaging model updates from clients. Other important parameters include a batch size of 4, a learning rate of 0.002, momentum of 0.9, and additional training specifics like local epochs, decay rate, step size, and gamma, optimizing the model's performance across the distributed system.

5.1.2 Selecting Appropriate Data Partitioning and Aggregation Strategy

Choosing the most appropriate data partitioning technique and aggregation strategy based on the user's data and class distribution is crucial for optimizing the training and validation outcomes of a Federated Learning model. We have explained the selection for these two parameters in a case-by-case manner below.

1. **Data Partitioning:** Based on whether the data has already been partitioned at the client level or not, there are two scenarios. In Scenario 1, client-level partitioned data is not available. If the client wants to test the Federated Learning simulation framework and has their dataset, we need to partition the data, i.e., distribute data to various clients using our data partitioning techniques. There are two techniques available: the Dirichlet or IID method, and based on the heterogeneity or skewness in class distribution, the client will choose the relevant partitioning technique. In Scenario 2, client-level partitioned data is already available. In this case, we have n as the total number of clients, and each client already has data. We don’t need to use any data partitioning strategy here. We just need to get the class distributions from all the clients and observe them. Based on the patterns available in class distribution, we will select our aggregation strategy in the next step.

2. **Aggregation Strategy:** In Federated Learning, different aggregation strategies address various data distribution challenges. FedAvg is best for high variability data, focusing on balanced accuracy and providing general robustness, making it suitable

for scenarios with significantly varied data across clients. FedProx is ideal for highly non-IID data, emphasizing regularization to maintain consistency across metrics, and ensuring stability where data is not uniformly distributed. FedAvgM suits cases with moderate data imbalance, focusing on high recall requirements and balanced performance, effective where high recall is crucial while maintaining balanced metrics. FedTrimmedAvg is designed for moderate non-IID data, providing outlier robustness and high precision, ideal for applications requiring precision and moderate non-IID data distribution.

For a step-by-step installation and usage of our Federated Learning setup, please refer to the detailed documentation here.

6. Conclusions & Recommendations

6.1 Summary

The primary challenge addressed by this framework is creating a flexible and scalable federated learning (FL) solution capable of simulating real-world data distribution scenarios and various aggregation strategies. This versatility facilitates extensive experimentation and the optimization of federated learning models to meet specific project needs. The framework supports simulating clients with diverse class distributions, suitable for both IID and non-IID data partitioning, with precision in class skew control via the alpha parameter for non-IID distributions. It incorporates four distinct aggregation strategies, each optimized for different data scenarios, and is designed to scale by accommodating a large number of clients. The model selection process demonstrated that ResNet-34 was optimal for the current dataset in a centralized setting, with provisions to test additional models as needed.

6.2 Effectiveness and Limitations

The framework successfully fulfills the need for a customizable and scalable FL solution, enabling clients to test various data distribution scenarios and aggregation strategies effectively. This approach helps in making informed choices about the most appropriate strategies for specific requirements. However, it faces limitations such as the variability in data quality, which may not perfectly represent real-world complexities. A potential enhancement could be to integrate a broader array of datasets and more sophisticated synthetic data generation methods. Resource constraints are another challenge due to the high computational and memory demands of large-scale federated learning experiments. Optimizing the framework for resource efficiency and exploring cloud solutions could mitigate this issue. While the framework is adaptable, incorporating advanced privacy-preserving methods requires additional development, suggesting a gradual integration of these technologies while consulting privacy experts to ensure effectiveness.

6.3 Extending the Framework and Future Improvements

To further the capabilities of the framework, new models can be added by defining their architecture in model.py and ensuring compatibility with existing training and evaluation functions. New aggregation strategies and data partitioning methods can also be introduced to enhance the framework's flexibility and adaptability. Looking ahead, the framework could benefit from testing on various datasets, implementing dynamic client participation, enhancing privacy mechanisms, improving scalability, developing real-time performance monitoring tools, and establishing fault tolerance. These enhancements will make the FL framework more robust and capable of accommodating a wide range of federated learning applications, potentially extending to cross-silo scenarios involving different organizations or institutions.

Differential Privacy: Implementing differential privacy can enhance the security of the federated learning pipeline by ensuring that individual data points cannot be re-identified from the model updates. However, adding noise to the data can affect model accuracy and requires careful tuning to balance privacy and utility.

Client-Level Evaluation: Introducing client-level evaluation allows for the assessment of model performance on each client's local data, providing more granular insights into model effectiveness. This approach, though beneficial, increases the computational overhead on client devices and adds complexity in aggregating and interpreting diverse evaluation metrics from different clients.

Implementing differential privacy and client-level evaluation could further secure and refine the federated learning processes, despite the challenges in balancing privacy with model accuracy and the increased computational demand on client devices.

7. Appendices

7.1 Codebase - Key Components

1. Configuration (`conf.yaml`) The configuration file (`conf.yaml`) sets essential parameters for the FL simulation, such as data directories, batch size, number of clients, partitioning strategies, and learning rates. This centralized configuration ensures that experiments are easily reproducible and modifiable.

2. Data Preparation (`dataset.py`) The `dataset.py` script prepares the dataset for federated learning by partitioning it into multiple subsets (one per client). It supports both IID and non-IID data distributions, with the ability to control the skewness of class distributions using the Dirichlet distribution.

Key functions:

- `prepare_dataset` : Prepares the dataset for FL, applying transformations and partitioning the data according to the specified strategy.
- `CustomDataset` : A custom dataset class to apply transformations to the data.

3. Client Implementation (`client.py`) The `client.py` script defines the behavior of individual clients in the federated learning process. It includes methods for setting and getting model parameters, training the model locally (`fit`), and evaluating the model (`evaluate`).

Key classes and functions:

- `FlowerClient` : Implements the FL client using the Flower framework, handling model training and evaluation.
- `generate_client_fn` : A helper function to generate client instances for the simulation.

4. Model Definitions (`model.py`) The `model.py` script contains definitions for various neural network models (e.g., ResNet34, VGG16, MobileNetV2) adapted for transfer learning. It includes functions for training and evaluating the models.

Key classes and functions:

- `ResNet34` , `VGG16` , `MobileNetV2Test` : Model classes adapted for the specific number of output classes.
- `new_trainer` : Trains the model on the client's local data.
- `test_model` : Evaluates the model on the test dataset.

5. Helper Functions (`helper.py`) The `helper.py` script provides utility functions for plotting various metrics and visualizing the results of the FL experiments.

Key functions:

- `plot_fed_confusion_matrix` : Plots the confusion matrix.
- `plot_fed_learning_curve` : Plots the learning curve showing training and validation losses over rounds.
- `plot_fed_roc_curve` : Plots the ROC curve.
- `plot_fed_pr_curve` : Plots the precision-recall curve.
- `plot_class_distribution` : Visualizes the class distribution across clients.

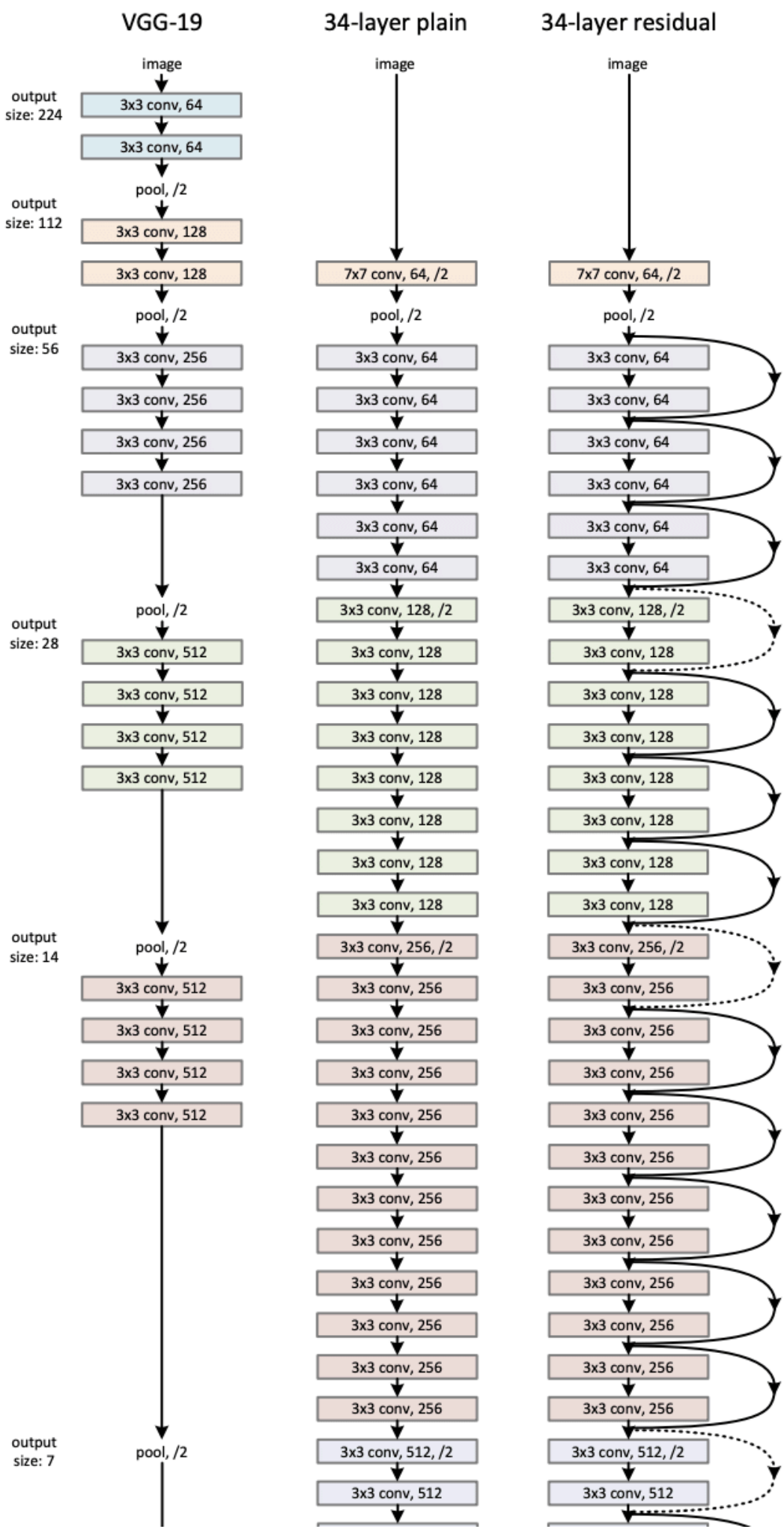
6. Server Implementation (`server.py`) The `server.py` script defines the server-side logic of the federated learning framework. It includes functions for configuring client training, evaluating global model performance, and defining aggregation strategies.

Key functions:

- `get_on_fit_config` : Generates client configurations for each training round.
- `get_evaluate_fn` : Defines the evaluation function for the global model.
- `agg_strategy` : Selects the aggregation strategy based on the specified configuration.

7. Main Script (`main.py`) The `main.py` script orchestrates the entire FL process. It loads the configuration, prepares the dataset, initializes clients and the server, and starts the FL simulation.

7.2 ResNet-34 Architecture [13]


VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
7x7 conv, 64, /2
3x3 conv, 128, /2
3x3 conv, 256, /2
3x3 conv, 512, /2

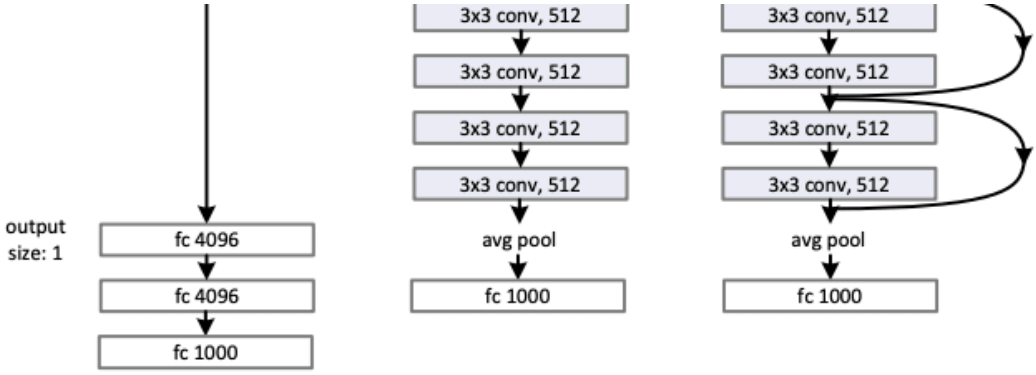


Figure 9. ResNet-34 Architecture

8. References

[1] Beltrán, E. T. M., Pérez, M. Q., Sánchez, P. M. S., Bernal, S. L., Bovet, G., Pérez, M. G., ... & Celdrán, A. H. (2023). Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. IEEE Communications Surveys & Tutorials. https://arxiv.org/abs/2211.08413

[2] Sun, T., Li, D., & Wang, B. (2022). Decentralized federated averaging. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(4), 4289-4301. https://ar5iv.labs.arxiv.org/html/2104.11375

[3] McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017, April). Communication-efficient learning of deep networks from decentralized data. In Artificial intelligence and statistics (pp. 1273-1282). PMLR. https://proceedings.mlr.press/v54/mcmahan17a?ref=https://githubhelp.com

[4] McMahan, H. B., Ramage, D., Talwar, K., & Zhang, L. (2017). Learning differentially private recurrent language models. arXiv preprint arXiv:1710.06963. https://arxiv.org/abs/1710.06963

[5] Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., & Bacon, D. (2016). Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492. https://arxiv.org/abs/1610.05492

[6] Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Kwing, H. L., Parcollet, T., Gusmão, P. P. B. de, & Lane, N. D. (2020). Flower: A friendly federated learning research framework. arXiv preprint arXiv:2007.14390. https://arxiv.org/abs/2007.14390

[7] Leavey, P., Sengupta, A., Rakheja, D., Daescu, O., Arunachalam, H. B., & Mishra, R. (2019). Osteosarcoma data from UT Southwestern/UT Dallas for Viable and Necrotic Tumor Assessment (Osteosarcoma-Tumor-Assessment) [Data set]. The Cancer Imaging Archive. https://doi.org/

[8] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M., I., & Stoica, I. (2017). Ray: a distributed framework for emerging AI

applications. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1712.05889

[9] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., . . . Chintala, S. (2019b). PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1912.01703

[10] Qi, P., Chiaro, D., Guzzo, A., Ianni, M., Fortino, G., & Piccialli, F. (2024). Model aggregation techniques in federated learning: A comprehensive survey. Future Generation Computer Systems, 150, 272–293. https://doi.org/10.1016/j.future.2023.09.008

[11] Vezakis, I. A., Lambrou, G. I., & Matsopoulos, G. K. (2023). Deep Learning Approaches to Osteosarcoma Diagnosis and Classification: A Comparative Methodological approach. Cancers, 15(8), 2290. https://doi.org/10.3390/cancers15082290

[12] Yadan, O. (2019). Hydra - A framework for elegantly configuring complex applications. GitHub. https://github.com/facebookresearch/hydra

[13] Dipu, N. M., Shohan, S. A., & Salam, K. (2021). Ocular disease detection using advanced neural network based classification algorithms. Asian Journal of Convergence in Technology, 7(2), 91–99. https://doi.org/10.33130/ajct.2021v07i02.019