Federated Learning of Medical Image Reconstruction

Software Design Document

Joshua Sheldon
Izzy MacDonald
Tanuj Kancharla
Yash Jani

March 4, 2025

Table of Contents

1.0 INTRODUCTION.. 3
1.1 Purpose..3
1.2 Background and Objectives..3
1.3 Definitions and Acronyms..4
1.3.1 Acronyms..4
1.3.2 Definitions..4
2.0 SYSTEM OVERVIEW..4
2.1 Data Synthesis Pipeline..5
2.2 Machine Learning Model..6
2.3 Federated Learning Applications..6
2.3.1 Orchestrator Application..7
2.3.2 Contributor Application..8
3.0 SYSTEM ARCHITECTURE.. 8
3.1 Architectural Design..8
3.2 Decomposition Description.. 8
4.0 DATA DESIGN.. 8
4.1 Data Description.. 8
4.2 Data Dictionary..9
5.0 HUMAN INTERFACE DESIGN.. 9
5.1 Overview of User Interface..9
5.2 Screen Images.. 10
6.0 APPENDICES..10
6.1 Synthetic Sinogram Similarity Tables..11
6.2 System Architecture Diagram..12
6.3 Federated Learning Sequence Diagram..13
6.4 Orchestrator Application Class Diagram..14
6.5 Contributor Application Class Diagram.. 15
6.6 Orchestrator Application User Interface..16
6.7 Contributor Application User Interface.. 17
7.0 BIBLIOGRAPHY..18

1.0 INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the "Federated Learning of Medical Image Reconstruction" project, including both user-facing and backend software.

1.2 Background and Objectives

"Single photon emission computed tomography (SPECT) is a nuclear imaging technique using gamma rays" [1]. A SPECT scan is begun by injecting a patient with a gamma ray emitting pharmaceutical (a tracer). The patient then lies down on a table in a scanning room equipped with a gamma camera, which uses a collimator instead of a lens and creates images by detecting radioactivity instead of light. These images are monochromatic images, where brightness in any given pixel of the image is determined by the tracer detection count at that position on the collimator's surface.

The gamma camera is rotated around the patient, capturing projections of a portion of the patient's body at different angles, creating a unique type of image called a sinogram, "where the horizontal axis represents the count location on the detector [gamma camera], and the vertical axis corresponds to the angular position of the detector" [2].

These sinograms are then reconstructed, traditionally using analytic and iterative algorithms [2], to create a medical image easily interpretable by medical professionals (henceforth referred to as a "reconstruction"). However, these algorithms are slow. The "Tomographic Medical Image Reconstruction using Deep Learning" project (henceforth referred to as "year 1 project") attempts to develop a machine learning (ML) model that can perform the reconstruction much quicker than traditional methods. The year 1 project trains this model exclusively on synthetically generated data, due to the lack of availability of and privacy concerns surrounding real data. Additionally, due to the lack of information regarding the statistical distribution of tracer concentration in other organs, the year 1 project's data consists exclusively of sinograms and reconstructions of the human heart and liver.

The accuracy of the year 1 project's model is limited by (1) the quality of the synthetic data and (2) the exclusive use of synthetic data. Federated Learning of Medical Image Reconstruction (henceforth referred to as "year 2 project") aims to improve the accuracy of this existing ML model by addressing both concerns.

1.3 Definitions and Acronyms

1.3.1 Acronyms

Acronym	**Term**
SPECT	Single photon emission computed tomography
ML	Machine learning
SSIM	Structural similarity index measure
BP-shift-FP	Backproject-shift-forwardproject
FL	Federated learning
UI	User interface

1.3.2 Definitions

Term	**Definition**
Tracer	A gamma ray emitting pharmaceutical used in nuclear imaging to capture tomographic images.
Sinogram	An image consisting of multiple projections of the subject at different angles.
Reconstruction	A tomographic image, reconstructed from a sinogram. Typically used by medical professionals.
Year 1 project	Tomographic Medical Image Reconstruction using Deep Learning, a senior design project from 2024-2025. Working in tandem with the project this document belongs to.
Year 2 project	Federated Learning of Medical Image Reconstruction, the project this document belongs to.
Phantom	A digital stand-in for the human body for the sake of imaging [3].

2.0 SYSTEM OVERVIEW

The year 2 project consists of three components: the data synthesis pipeline, the machine learning model, and the federated learning applications.

2.1 Data Synthesis Pipeline

The data synthesis pipeline enables the generation of entirely artificial training data for use with the machine learning model.

The pipeline begins with the creation of human body phantoms, which are digital stand-ins for human tissues [3]. This is accomplished with the XCAT phantom program by Dr. Paul Segars. “Combined with accurate models of the imaging process, the XCAT can produce realistic multimodality imaging data close to that obtained from patient studies… Through user-defined parameters, any number of different anatomies, cardiac or respiratory motions or patterns, and spatial resolutions can be simulated to perform medical imaging research” [4].

However, while the phantoms contain all organs of the human body, we are exclusively interested in the heart and the liver, for which we have tracer concentration distributions. Therefore, after generating the phantom, we crop it on the sagittal plane to isolate the heart, accepting some portions of the liver and other organs will remain in the cropped phantom.

In the above quote, “accurate models of the imaging process” are mentioned. This is because the voxel values of the phantom are only used to identify the different types of tissues in the phantom, meaning there is no data regarding tracer concentration within the phantom. However, due to previous research [5], we do have statistical distributions for tracer concentration within the heart and the liver for both healthy and diseased patients at rest and under stress. For the purposes of this project, we only synthesize data modeling patients at rest. The next step after cropping is applying these statistical distributions to the phantom using XCAT+, a program developed in part by members of the year 1 and year 2 projects. We apply these distributions five separate times on one phantom, creating five tomographic images, hopefully similar to those reconstructed from sinograms of real patients. Comparisons using structural similarity index measure (SSIM) [6] suggest any two sinograms from the same phantom are about ~34% similar. These comparisons are illustrated in **Appendix 6.1: Synthetic Sinogram Similarity Tables**, where each figure compares sinogrmas from the same phantom with each other using SSIM.

Next, we use GATE, a Monte-Carlo simulation toolkit for medical physics applications [7]. GATE allows us to simulate performing a SPECT scan on the synthetic tomographic image. We simulate one SPECT scan on each image (meaning five scans per phantom), generating five sinograms and therefore five training data samples, where the input is the sinogram and the output is the tomographic image. However, with our current computational resources, it takes upwards of ten hours to generate one sinogram. While we hope to reduce this time by using AI.Panther [8], even if the time was reduced substantially, the rate of synthesis would not be sufficient to train the model within our allotted time.

The solution, therefore, is augmentation. The augmentation stage of the pipeline currently consists of two steps:

1. A Z-shift in Cartesian space, or a clockwise rotation of the sinogram and the tomographic image.
2. A step titled “backproject-shift-forwardproject” (henceforth referred to as “BP-shift-FP”), where the inverse radon transform is applied to the sinogram, both the sinogram and the tomographic image are shifted on the XY plane, and then the radon transform is applied again to the sinogram, bringing it back to its original form.

However, one consequence of the augmentation stage is that the BP-shift-FP step drastically reduces image quality. It was implemented in the year 1 project because of the lack of anatomical diversity in phantoms, meaning that without the shifting step the model would never generalize to different positions and orientations of the heart. The year 2 project hopes to eliminate this step by increasing the anatomical diversity of XCATs, and making up for the loss in augmentation factor by reducing sinogram generation time through phantom cropping and running GATE on AI.Panther.

This pipeline allows for the synthesis of training data with (theoretically) a high degree of similarity to real data. While the vast majority of this pipeline is carried over from the year 1 project, multiple steps have been added, removed, and modified to improve the quality of the data.

2.2 Machine Learning Model

The ML model is a convolutional encoder-decoder extended with attention (CEDA), modeled after prior research in performing the inverse radon transform with deep learning [9]. The model takes in sinograms and attempts to reconstruct them into tomographic images in a similar fashion to traditional iterative methods. While the train and validation sets are comprised entirely of synthetic data, the test set contains 39 samples of real data, ensuring that modifications that we make to the data synthesis pipeline and the model improve performance on real data.

2.3 Federated Learning Applications

The choice to train a model with synthetic data derives from the scarcity of real patient data, largely due to legal and privacy concerns. However, one interesting solution to this predicament is the federated learning paradigm. “Federated Learning (FL) is a machine learning technique that enables multiple entities to collaboratively learn a shared model without exchanging their local data” [10]. FL accomplishes this by executing the following procedure with one orchestrator machine and one or more contributor machines [11]:

1. The orchestrator initializes a model to train, and distributes the model to a set of contributor machines.

2. The contributor machines train this model with locally stored data.
3. The contributor machines send their respective trained models to the orchestrator.
4. The orchestrator uses an aggregation function to merge the models of the contributors into one model, which ideally benefits from the learning of each contributing model.

This paradigm has clear application in the medical field, as it can allow medical institutions to contribute to research using patient data without directly sharing data and/or accidentally violating the patient's privacy. Therefore, our objective is to build a set of applications that can apply FL to "fine-tune" our already trained ML model with real data.

To assist in the implementation of this paradigm, we use the Flower FL framework in our applications [12]. The Flower framework consists of five components relevant to the design of our applications:

- **Strategy** - A class, belonging to the ServerApp, detailing how an ML model being developed with FL is initialized, trained, aggregated, and evaluated.
- **ServerApp** - A class, belonging to the orchestrator, defining which Strategy will be used in FL and maintaining configuration options which modify the execution of that Strategy.
- **SuperLink** - A process that runs the ServerApp, sending tasks directed by it to the contributors, receiving the results of those tasks, and passing them back to the Server App.
- **ClientApp** - A class, belonging to the contributor, defining how data will be loaded to train the ML model.
- **SuperNode -** A process that runs the ClientApp, asks the SuperLink for tasks, passes tasks off to the ClientApp for execution, and returns the results to the SuperLink.

2.3.1 Orchestrator Application

The first application is the orchestrator application. This application would be managed by us, or any other individual or organization attempting to facilitate the training of this model. The orchestrator application allows the user to, for any given round of FL, (1) choose the initial model that will be distributed to contributors, (2) select which contributors will participate in the round, and (3) both start and monitor the status of the round. Additionally, the application allows the user to browse through the history of models trained through FL.

Enabling this functionality entails a host of other components within the orchestrator, including:

- Servers to facilitate communications between the orchestrator and the user and the orchestrator and its contributors.
- Components that keep track of contributors, models, and their associated data.
- The Strategy and ServerApp, detailing the exact methods of model initialization, training, aggregation, etc.

- A component which manages the SuperLink process and runs the ServerApp.

2.3.2 Contributor Application

The second application is the contributor application. This application would be managed by personnel at medical institutions contributing data for the improvement of our ML model. The contributor application allows the user to (1) submit training data for use in FL, (2) mark the machine as available or unavailable for participation in a FL round, and (3) monitor the status of the contributor as it participates in a round.

Enabling this functionality, again, entails other components within the contributor, including:

- A server to facilitate communication between the contributor and the user.
- A component that communicates information about the contributor (like number of training data samples and availability) to the orchestrator and receives tasks from it (like starting Flower framework components).
- A component that maintains, counts, and clears training data when appropriate.
- The ClientApp, detailing how the contributor will load data to use in FL.
- A component which manages the SuperNode process and runs the ClientApp.

3.0 SYSTEM ARCHITECTURE

3.1 Architectural Design

<See Appendix 6.2: System Architecture Diagram>

This diagram elucidates the components, subcomponents, responsibilities, and relationships defined in Section 2.0 System Overview.

3.2 Decomposition Description

<See Appendix 6.3: Federated Learning Sequence Diagram>
<See Appendix 6.4: Orchestrator Application Class Diagram>
<See Appendix 6.5: Contributor Application Class Diagram>

4.0 DATA DESIGN

4.1 Data Description

The primary data of the project is medical images (phantoms, tomographic images, sinograms, etc.) These medical images are stored as raw files (extensions may be none, '.raw', '.bin', '.Vol'), simply containing the voxel values without any other encoding or metadata. Medical images are stored in this format throughout the data synthesis pipeline, as well as within the machine learning model and contributor application.

The ML models, both trained on synthetic data and produced by the FL applications, are stored as PyTorch files (extensions may be '.pt' or '.pth').

The metadata associated with models stored by the orchestrator application is JSON data, stored in '.json' files.

Most other data, such as that involved in the communications between the orchestrator and its contributors, or between the applications and their users, is transferred through REST requests, and as such, is JSON data. This data should not be stored in files.

4.2 Data Dictionary

Name	**Information**
ML Model	PyTorch file. Size varies on the number of parameters of the model.
ML Model Metadata	JSON file.
Phantom	Raw file. 128x128x600. 600 transverse slices of the human body.
Sinogram	Raw file. 128x128x240 or 128x128x120. 120 images captured at different angles of rotation around the subject. Sometimes one image's content is split across two images, in which case the sinogram must be squashed.
Tomographic Image / Reconstruction	Raw file. 128x128x128 for real data. 128x128x~64 for synthetic data.

5.0 HUMAN INTERFACE DESIGN

5.1 Overview of User Interface

The data synthesis pipeline and ML model components do not have traditional user interfaces (UIs), as they are composed primarily of Python scripts and Jupyter Notebooks that will be run from the terminal or from integrated development environments.

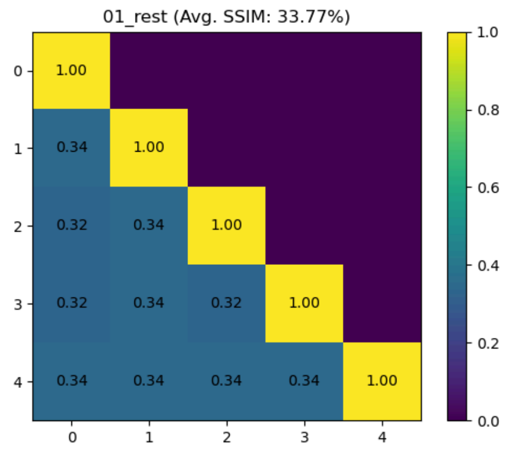
However, the FL applications do have UIs. While the applications themselves shall be daemons, they will start web application based UIs that can be accessed by the user at any time through a browser. These UIs will allow users to configure, submit data to, start, monitor, and browse the products of the FL system.

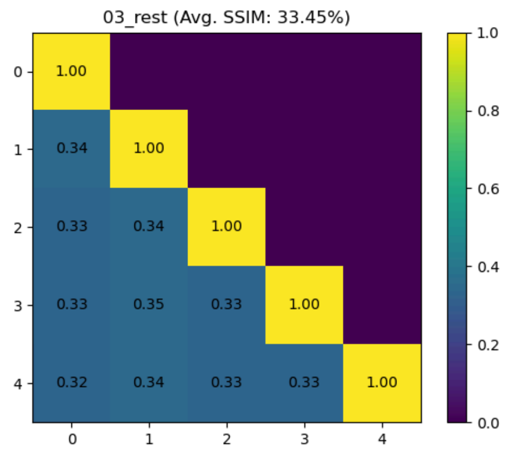
5.2 Screen Images

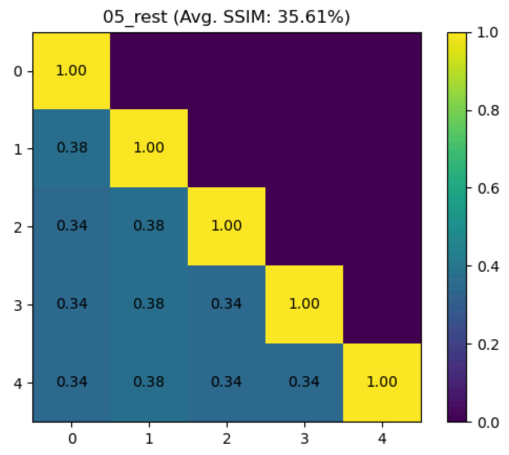
<See Appendix 6.6: Orchestrator Application User Interface>
<See Appendix 6.7: Contributor Application User Interface>

6.0 APPENDICES

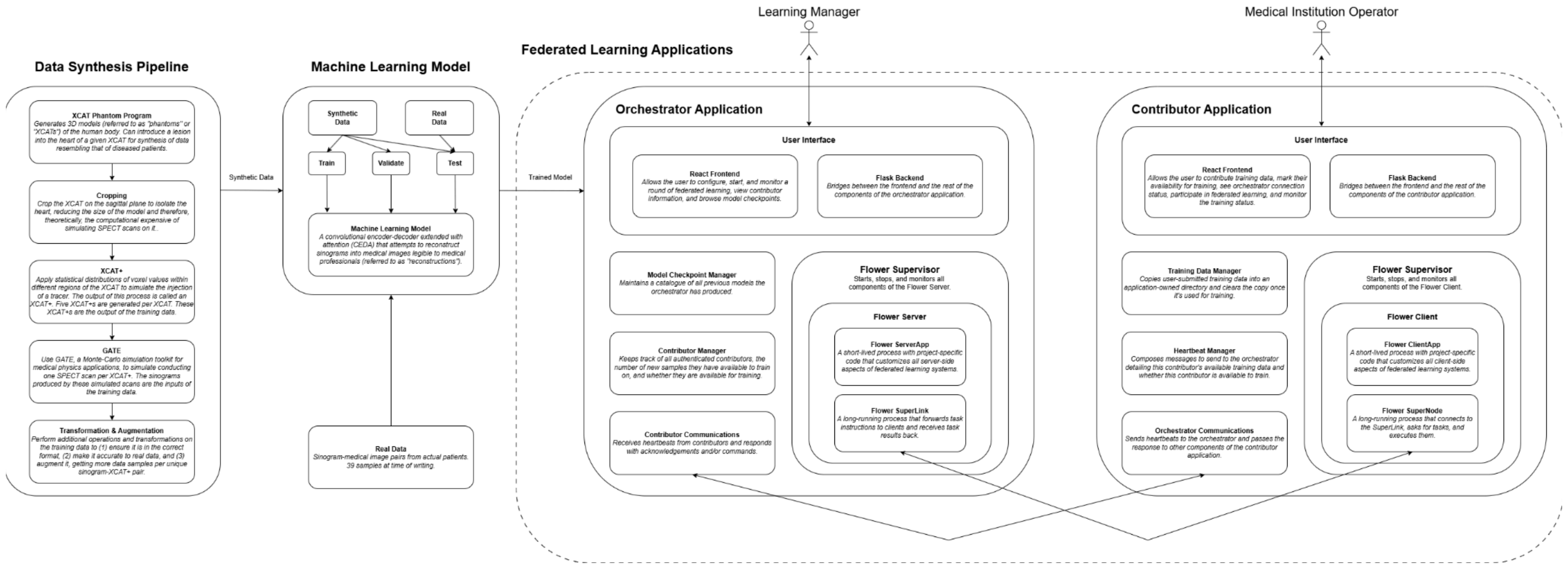
6.1 Synthetic Sinogram Similarity Tables







6.2 System Architecture Diagram



XCAT Phantom Program
Generates 3D models (referred to as "phantoms" or "XCATs") of the human body. Can introduce a lesion into the heart of a given XCAT for synthesis of data resembling that of diseased patients.

Cropping
Crop the XCAT on the sagittal plane to isolate the heart, reducing the size of the model and therefore, theoretically, the computational expensive of simulating SPECT scans on it..

XCAT+
Apply statistical distributions of voxel values within different regions of the XCAT to simulate the injection of a tracer. The output of this process is called an XCAT+. Five XCAT+s are generated per XCAT. These XCAT+s are the output of the training data.

GATE
Use GATE, a Monte-Carlo simulation toolkit for medical physics applications, to simulate conducting one SPECT scan per XCAT+. The sinograms produced by these simulated scans are the inputs of the training data.

Transformation & Augmentation
Perform additional operations and transformations on the training data to (1) ensure it is in the correct format, (2) make it accurate to real data, and (3) augment it, getting more data samples per unique sinogram-XCAT+ pair.

Machine Learning Model
A convolutional encoder-decoder extended with attention (CEDA) that attempts to reconstruct sinograms into medical images legible to medical professionals (referred to as "reconstructions").

Real Data
Sinogram-medical image pairs from actual patients. 39 samples at time of writing.

React Frontend
Allows the user to configure, start, and monitor a round of federated learning, view contributor information, and browse model checkpoints.

Flask Backend
Bridges between the frontend and the rest of the components of the orchestrator application.

Model Checkpoint Manager
Maintains a catalogue of all previous models the orchestrator has produced.

Contributor Manager
Keeps track of all authenticated contributors, the number of new samples they have available to train on, and whether they are available for training.

Contributor Communications
Receives heartbeats from contributors and responds with acknowledgements and/or commands.

Flower Supervisor
Starts, stops, and monitors all components of the Flower Server.

Flower ServerApp
A short-lived process with project-specific code that customizes all server-side aspects of federated learning systems.

Flower SuperLink
A long-running process that forwards task instructions to clients and receives task results back.

React Frontend
Allows the user to contribute training data, mark their availability for training, see orchestrator connection status, participate in federated learning, and monitor the training status.

Flask Backend
Bridges between the frontend and the rest of the components of the contributor application.

Training Data Manager
Copies user-submitted training data into an application-owned directory and clears the copy once it's used for training.

Heartbeat Manager
Composes messages to send to the orchestrator detailing this contributor's available training data and whether this contributor is available to train.

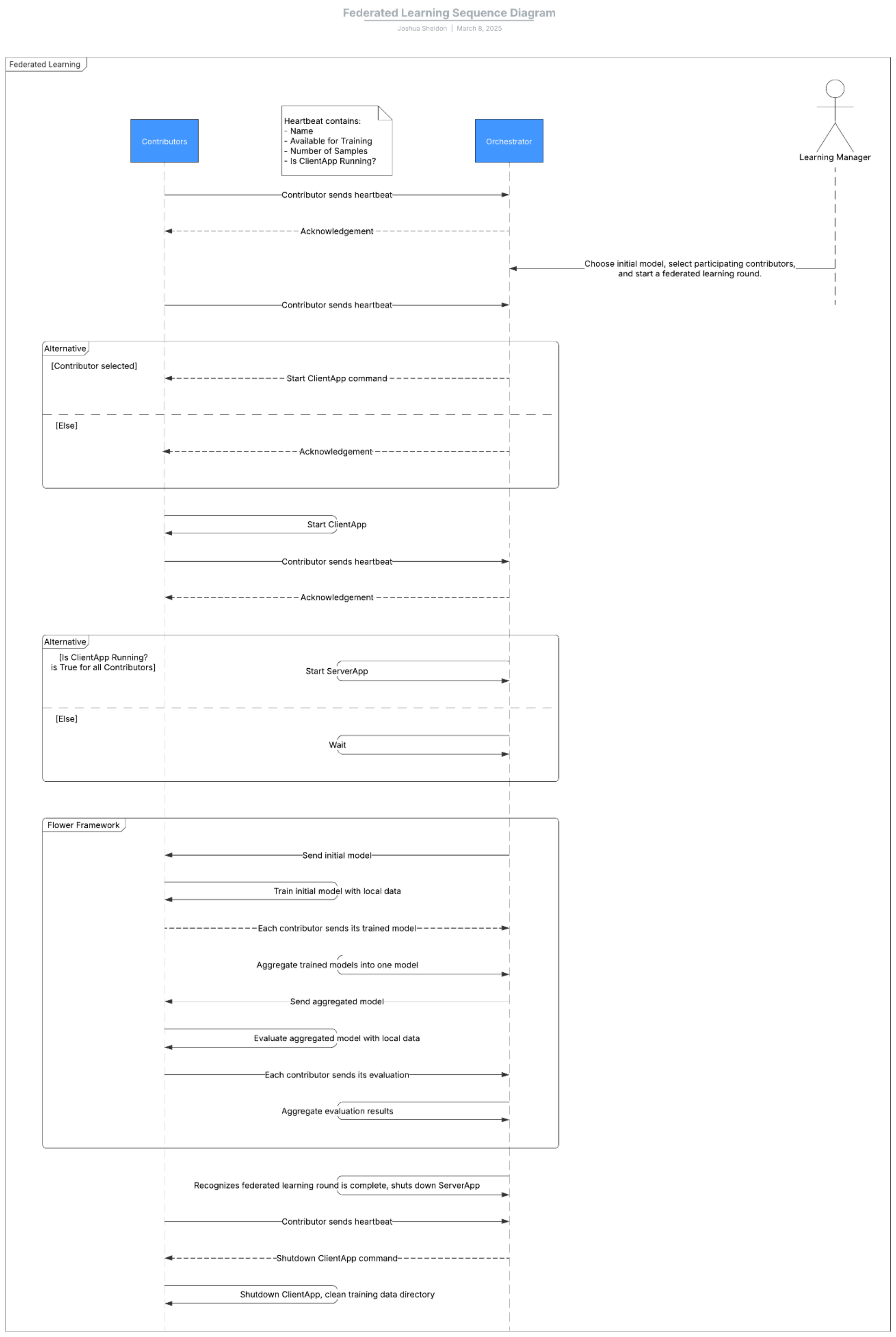
Orchestrator Communications
Sends heartbeats to the orchestrator and passes the response to other components of the contributor application.

Flower Supervisor
Starts, stops, and monitors all components of the Flower Client.

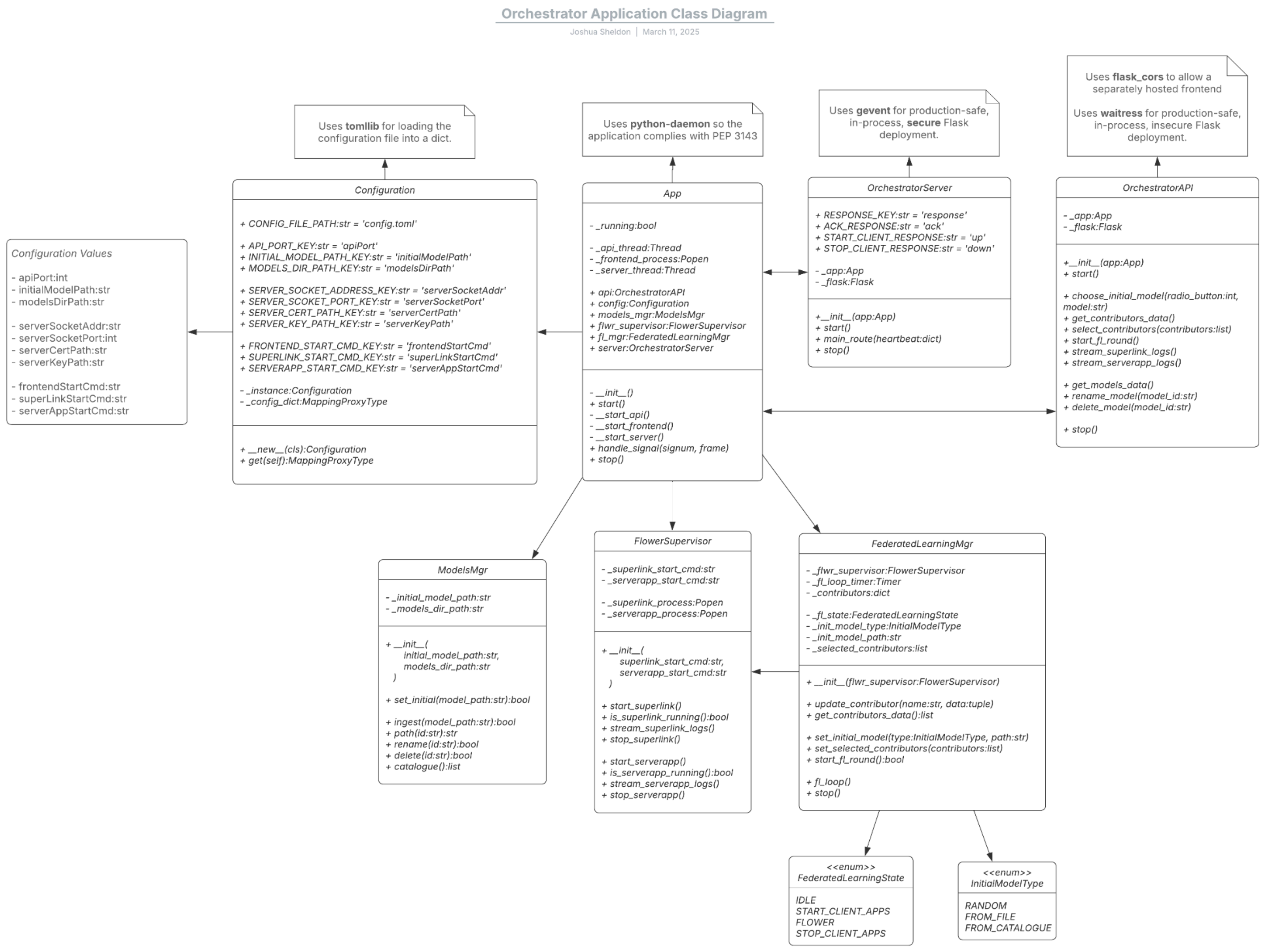
Flower ClientApp
A short-lived process with project-specific code that customizes all client-side aspects of federated learning systems.

Flower SuperNode
A long-running process that connects to the SuperLink, asks for tasks, and executes them.

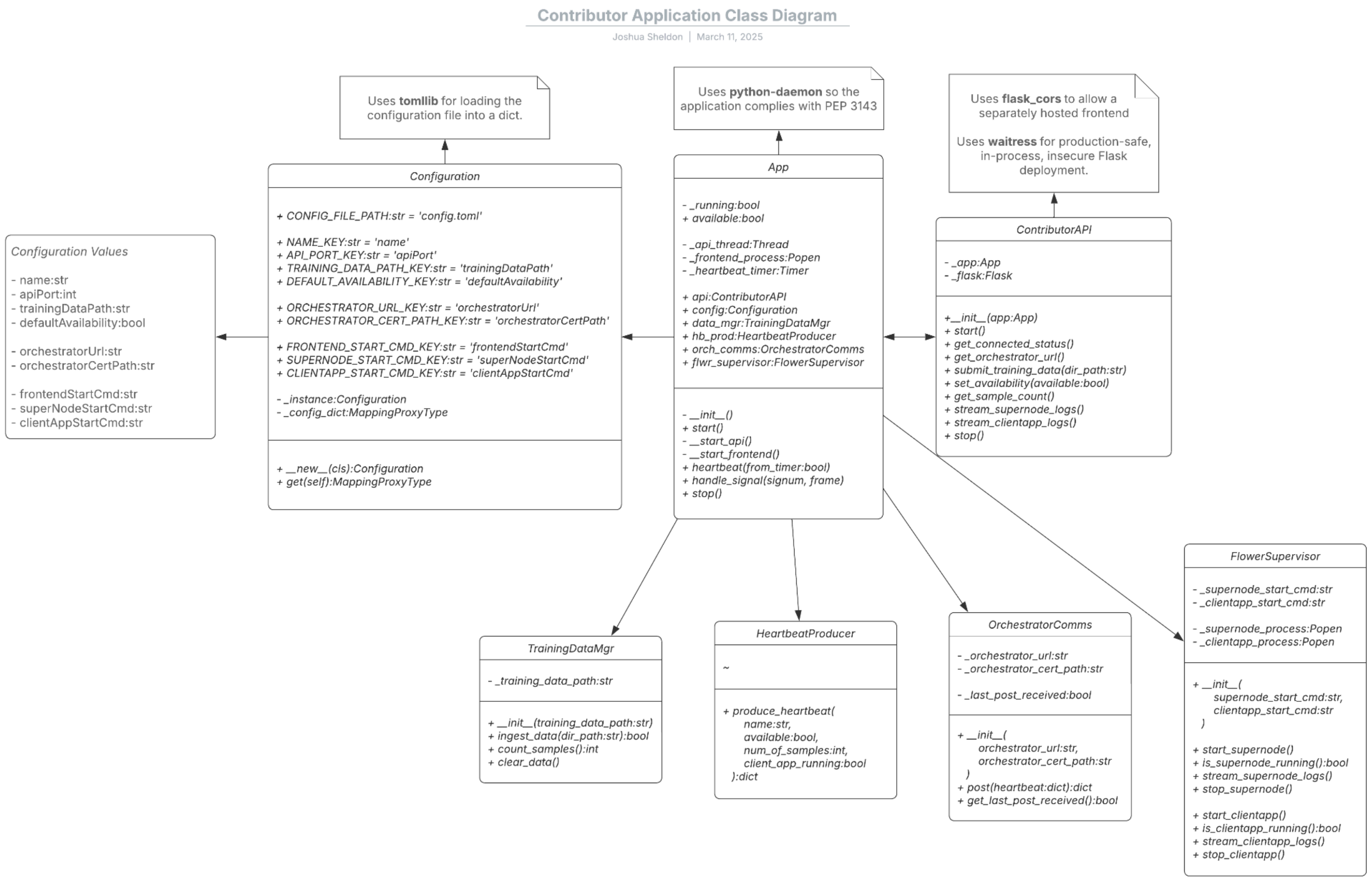
6.3 Federated Learning Sequence Diagram


Federated Learning Sequence Diagram
Joshua Sheldon | March 8, 2025
Federated Learning
Contributors
Heartbeat contains:
- Name
- Available for Training
- Number of Samples
- Is ClientApp Running?
Orchestrator
Learning Manager
Contributor sends heartbeat
Acknowledgement
Choose initial model, select participating contributors, and start a federated learning round.
Contributor sends heartbeat
Alternative
[Contributor selected]
Start ClientApp command
[Else]
Acknowledgement
Start ClientApp
Contributor sends heartbeat
Acknowledgement
Alternative
[Is ClientApp Running? is True for all Contributors]
Start ServerApp
[Else]
Wait
Flower Framework
Send initial model
Train initial model with local data
Each contributor sends its trained model
Aggregate trained models into one model
Send aggregated model
Evaluate aggregated model with local data
Each contributor sends its evaluation
Aggregate evaluation results
Recognizes federated learning round is complete, shuts down ServerApp
Contributor sends heartbeat
Shutdown ClientApp command
Shutdown ClientApp, clean training data directory

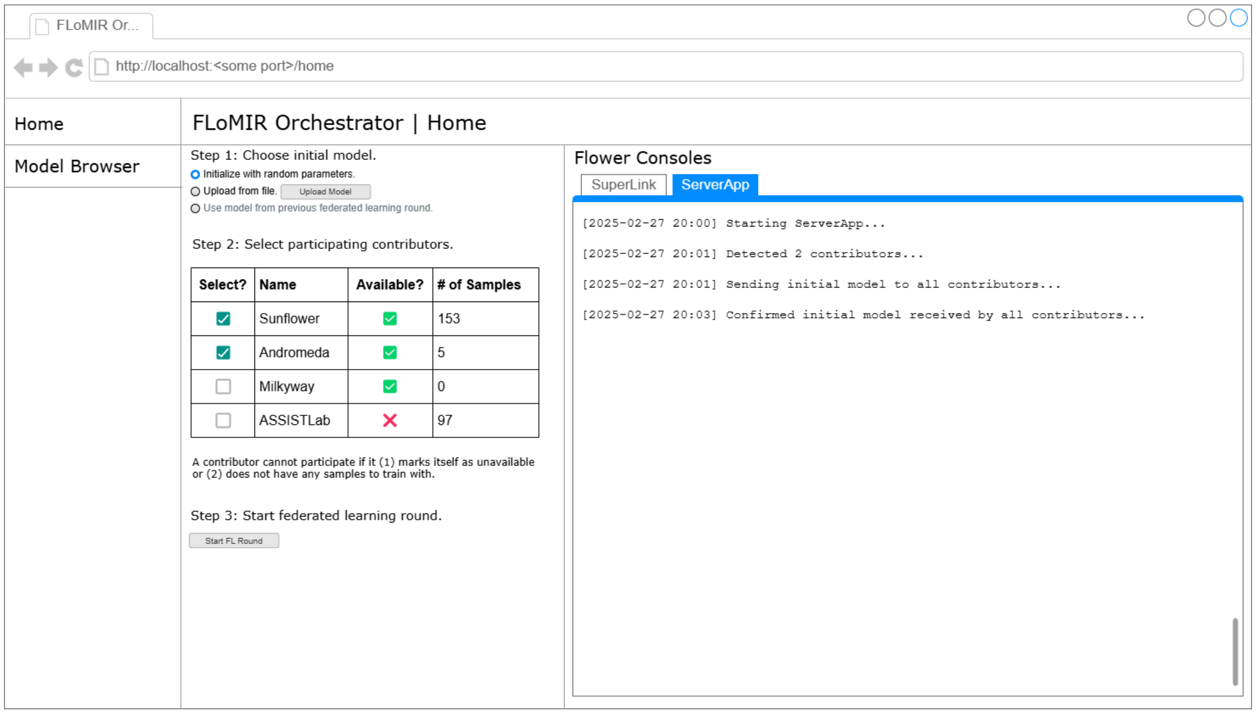
6.4 Orchestrator Application Class Diagram


Orchestrator Application Class Diagram
Joshua Sheldon | March 11, 2025
Uses **tomllib** for loading the configuration file into a dict.
Uses **python-daemon** so the application complies with PEP 3143
Uses **gevent** for production-safe, in-process, **secure** Flask deployment.
Uses **flask_cors** to allow a separately hosted frontend
Uses **waitress** for production-safe, in-process, insecure Flask deployment.
Configuration Values
- apiPort:int
- initialModelPath:str
- modelsDirPath:str
- serverSocketAddr:str
- serverSocketPort:int
- serverCertPath:str
- serverKeyPath:str
- frontendStartCmd:str
- superLinkStartCmd:str
- serverAppStartCmd:str
Configuration
+ CONFIG_FILE_PATH:str = 'config.toml'
+ API_PORT_KEY:str = 'apiPort'
+ INITIAL_MODEL_PATH_KEY:str = 'initialModelPath'
+ MODELS_DIR_PATH_KEY:str = 'modelsDirPath'
+ SERVER_SOCKET_ADDRESS_KEY:str = 'serverSocketAddr'
+ SERVER_SCOKET_PORT_KEY:str = 'serverSocketPort'
+ SERVER_CERT_PATH_KEY:str = 'serverCertPath'
+ SERVER_KEY_PATH_KEY:str = 'serverKeyPath'
+ FRONTEND_START_CMD_KEY:str = 'frontendStartCmd'
+ SUPERLINK_START_CMD_KEY:str = 'superLinkStartCmd'
+ SERVERAPP_START_CMD_KEY:str = 'serverAppStartCmd'
- _instance:Configuration
- _config_dict:MappingProxyType
+ __new__(cls):Configuration
+ get(self):MappingProxyType
App
- _running:bool
- _api_thread:Thread
- _frontend_process:Popen
- _server_thread:Thread
+ api:OrchestratorAPI
+ config:Configuration
+ models_mgr:ModelsMgr
+ flwr_supervisor:FlowerSupervisor
+ fl_mgr:FederatedLearningMgr
+ server:OrchestratorServer
- __init__()
+ start()
- __start_api()
- __start_frontend()
- __start_server()
+ handle_signal(signum, frame)
+ stop()
OrchestratorServer
+ RESPONSE_KEY:str = 'response'
+ ACK_RESPONSE:str = 'ack'
+ START_CLIENT_RESPONSE:str = 'up'
+ STOP_CLIENT_RESPONSE:str = 'down'
- _app:App
- _flask:Flask
+__init__(app:App)
+ start()
+ main_route(heartbeat:dict)
+ stop()
OrchestratorAPI
- _app:App
- _flask:Flask
+__init__(app:App)
+ start()
+ choose_initial_model(radio_button:int, model:str)
+ get_contributors_data()
+ select_contributors(contributors:list)
+ start_fl_round()
+ stream_superlink_logs()
+ stream_serverapp_logs()
+ get_models_data()
+ rename_model(model_id:str)
+ delete_model(model_id:str)
+ stop()
ModelsMgr
- _initial_model_path:str
- _models_dir_path:str
+ __init__(
initial_model_path:str,
models_dir_path:str
)
+ set_initial(model_path:str):bool
+ ingest(model_path:str):bool
+ path(id:str):str
+ rename(id:str):bool
+ delete(id:str):bool
+ catalogue():list
FlowerSupervisor
- _superlink_start_cmd:str
- _serverapp_start_cmd:str
- _superlink_process:Popen
- _serverapp_process:Popen
+ __init__(
superlink_start_cmd:str,
serverapp_start_cmd:str
)
+ start_superlink()
+ is_superlink_running():bool
+ stream_superlink_logs()
+ stop_superlink()
+ start_serverapp()
+ is_serverapp_running():bool
+ stream_serverapp_logs()
+ stop_serverapp()
FederatedLearningMgr
- _flwr_supervisor:FlowerSupervisor
- _fl_loop_timer:Timer
- _contributors:dict
- _fl_state:FederatedLearningState
- _init_model_type:InitialModelType
- _init_model_path:str
- _selected_contributors:list
+ __init__(flwr_supervisor:FlowerSupervisor)
+ update_contributor(name:str, data:tuple)
+ get_contributors_data():list
+ set_initial_model(type:InitialModelType, path:str)
+ set_selected_contributors(contributors:list)
+ start_fl_round():bool
+ fl_loop()
+ stop()
<<enum>>
FederatedLearningState
IDLE
START_CLIENT_APPS
FLOWER
STOP_CLIENT_APPS
<<enum>>
InitialModelType
RANDOM
FROM_FILE
FROM_CATALOGUE

6.5 Contributor Application Class Diagram


Contributor Application Class Diagram
Joshua Sheldon | March 11, 2025
Uses tomllib for loading the configuration file into a dict.
Uses python-daemon so the application complies with PEP 3143
Uses flask_cors to allow a separately hosted frontend
Uses waitress for production-safe, in-process, insecure Flask deployment.
Configuration Values
- name:str
- apiPort:int
- trainingDataPath:str
- defaultAvailability:bool
- orchestratorUrl:str
- orchestratorCertPath:str
- frontendStartCmd:str
- superNodeStartCmd:str
- clientAppStartCmd:str
Configuration
+ CONFIG_FILE_PATH:str = 'config.toml'
+ NAME_KEY:str = 'name'
+ API_PORT_KEY:str = 'apiPort'
+ TRAINING_DATA_PATH_KEY:str = 'trainingDataPath'
+ DEFAULT_AVAILABILITY_KEY:str = 'defaultAvailability'
+ ORCHESTRATOR_URL_KEY:str = 'orchestratorUrl'
+ ORCHESTRATOR_CERT_PATH_KEY:str = 'orchestratorCertPath'
+ FRONTEND_START_CMD_KEY:str = 'frontendStartCmd'
+ SUPERNODE_START_CMD_KEY:str = 'superNodeStartCmd'
+ CLIENTAPP_START_CMD_KEY:str = 'clientAppStartCmd'
- _instance:Configuration
- _config_dict:MappingProxyType
+ __new__(cls):Configuration
+ get(self):MappingProxyType
App
- _running:bool
+ available:bool
- _api_thread:Thread
- _frontend_process:Popen
- _heartbeat_timer:Timer
+ api:ContributorAPI
+ config:Configuration
+ data_mgr:TrainingDataMgr
+ hb_prod:HeartbeatProducer
+ orch_comms:OrchestratorComms
+ flwr_supervisor:FlowerSupervisor
- __init__()
+ start()
- __start_api()
- __start_frontend()
+ heartbeat(from_timer:bool)
+ handle_signal(signum, frame)
+ stop()
ContributorAPI
- _app:App
- _flask:Flask
+__init__(app:App)
+ start()
+ get_connected_status()
+ get_orchestrator_url()
+ submit_training_data(dir_path:str)
+ set_availability(available:bool)
+ get_sample_count()
+ stream_supernode_logs()
+ stream_clientapp_logs()
+ stop()
TrainingDataMgr
- _training_data_path:str
+ __init__(training_data_path:str)
+ ingest_data(dir_path:str):bool
+ count_samples():int
+ clear_data()
HeartbeatProducer
~
+ produce_heartbeat(
name:str,
available:bool,
num_of_samples:int,
client_app_running:bool
):dict
OrchestratorComms
- _orchestrator_url:str
- _orchestrator_cert_path:str
- _last_post_received:bool
+ __init__(
orchestrator_url:str,
orchestrator_cert_path:str
)
+ post(heartbeat:dict):dict
+ get_last_post_received():bool
FlowerSupervisor
- _supernode_start_cmd:str
- _clientapp_start_cmd:str
- _supernode_process:Popen
- _clientapp_process:Popen
+ __init__(
supernode_start_cmd:str,
clientapp_start_cmd:str
)
+ start_supernode()
+ is_supernode_running():bool
+ stream_supernode_logs()
+ stop_supernode()
+ start_clientapp()
+ is_clientapp_running():bool
+ stream_clientapp_logs()
+ stop_clientapp()

6.6 Orchestrator Application User Interface



Step 1: Choose initial model.

- [✓ Initialize with random parameters.]
- Upload from file.
- Use model from previous federated learning round.

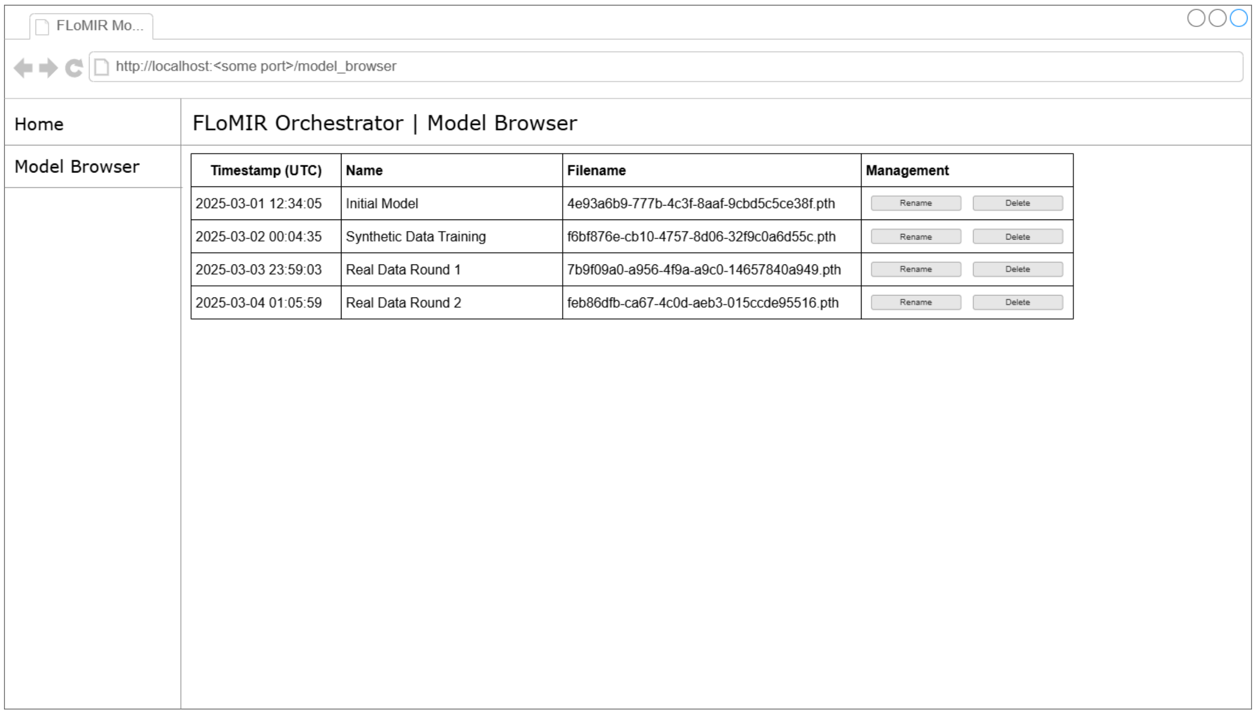
Step 2: Select participating contributors.

Select?	Name	Available?	# of Samples
✓	Sunflower	✓	153
✓	Andromeda	✓	5
☐	Milkyway	✓	0
☐	ASSISTLab	✗	97

A contributor cannot participate if it (1) marks itself as unavailable or (2) does not have any samples to train with.

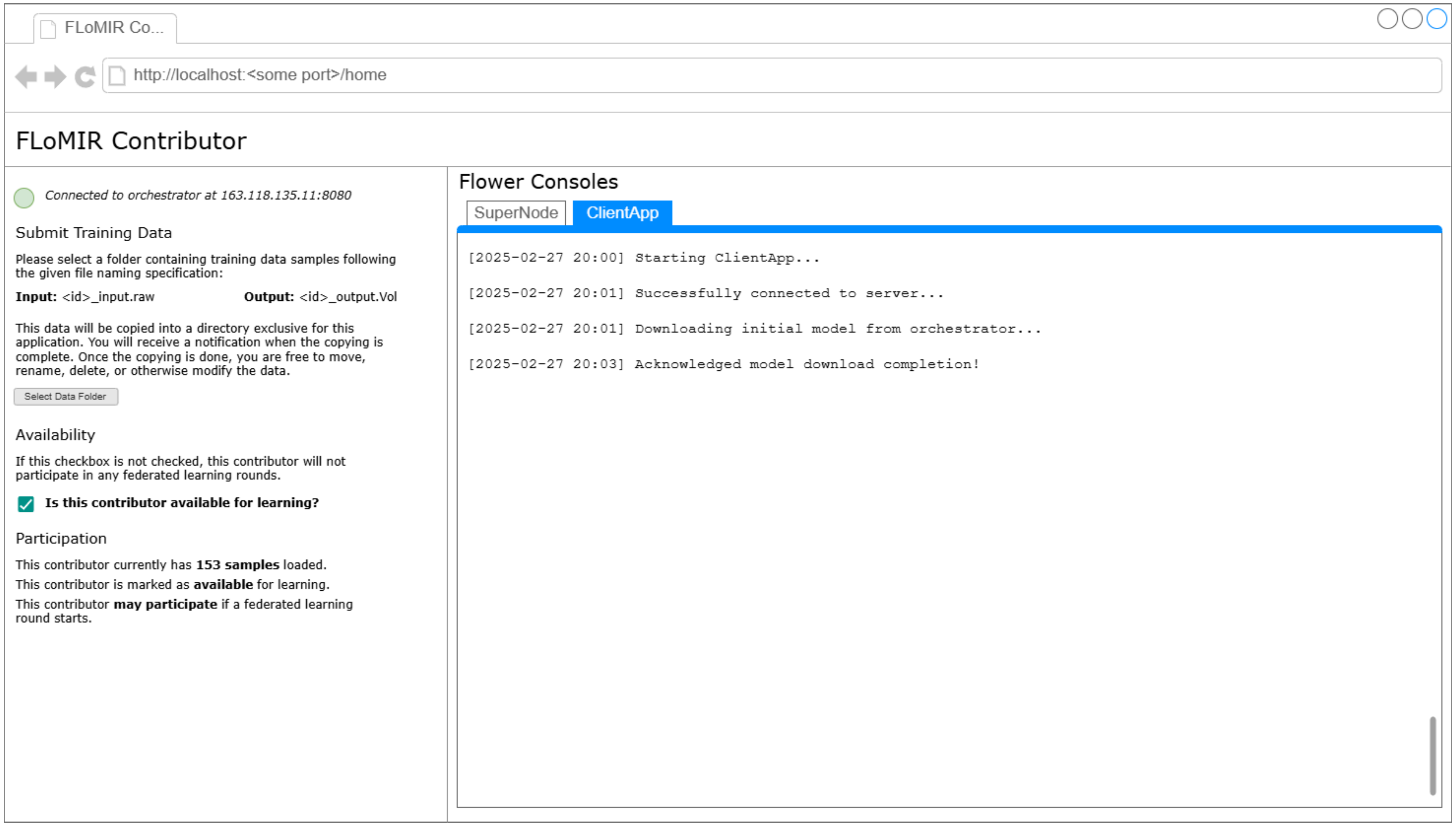
Step 3: Start federated learning round.

```
[2025-02-27 20:00] Starting ServerApp...
[2025-02-27 20:01] Detected 2 contributors...
[2025-02-27 20:01] Sending initial model to all contributors...
[2025-02-27 20:03] Confirmed initial model received by all contributors...
```


FLoMIR Mo...
http://localhost:<some port>/model_browser
Home
Model Browser
FLoMIR Orchestrator | Model Browser

Timestamp (UTC)	Name	Filename	Management
2025-03-01 12:34:05	Initial Model	4e93a6b9-777b-4c3f-8aaf-9cbd5c5ce38f.pth	Rename Delete
2025-03-02 00:04:35	Synthetic Data Training	f6bf876e-cb10-4757-8d06-32f9c0a6d55c.pth	Rename Delete
2025-03-03 23:59:03	Real Data Round 1	7b9f09a0-a956-4f9a-a9c0-14657840a949.pth	Rename Delete
2025-03-04 01:05:59	Real Data Round 2	feb86dfb-ca67-4c0d-aeb3-015ccde95516.pth	Rename Delete

6.7 Contributor Application User Interface



FLoMIR Contributor

Connected to orchestrator at 163.118.135.11:8080

Submit Training Data

Please select a folder containing training data samples following the given file naming specification:

Input: <id>_input.raw **Output:** <id>_output.Vol

This data will be copied into a directory exclusive for this application. You will receive a notification when the copying is complete. Once the copying is done, you are free to move, rename, delete, or otherwise modify the data.

Availability

If this checkbox is not checked, this contributor will not participate in any federated learning rounds.

✓ **Is this contributor available for learning?**

Participation

This contributor currently has **153 samples** loaded.

This contributor is marked as **available** for learning.

This contributor **may participate** if a federated learning round starts.

Flower Consoles

```
[2025-02-27 20:00] Starting ClientApp...

[2025-02-27 20:01] Successfully connected to server...

[2025-02-27 20:01] Downloading initial model from orchestrator...

[2025-02-27 20:03] Acknowledged model download completion!
```

7.0 BIBLIOGRAPHY

[1] University of Utah, "SPECT," *Radiology | U of U School of Medicine*. https://medicine.utah.edu/radiology/research/learn/spect

[2] P. Bruyant, "Analytic and Iterative Reconstruction Algorithms in SPECT," *Journal of Nuclear Medicine*, vol. 43, no. 10, pp. 1343–1358, Oct. 2002, Available: https://jnm.snmjournals.org/content/43/10/1343

[3] National Institute of Standards and Technology, "What Are Imaging Phantoms?," *NIST*, Jan. 24, 2024. https://www.nist.gov/node/1333486

[4] W. P. Segars, G. Sturgeon, S. Mendonca, J. Grimes, and B. M. W. Tsui, "4D XCAT phantom for multimodality imaging research," *Medical Physics*, vol. 37, no. 9, pp. 4902–4915, Aug. 2010, doi: https://doi.org/10.1118/1.3480985.

[5] C. T. Collins, C. Hinton, T. Galletta, P. Segars, and D. Mitra, "Distribution of 99mTc-tetrofosmin in Real SPECT Image Reconstructions," *2024 IEEE Nuclear Science Symposium (NSS), Medical Imaging Conference (MIC) and Room Temperature Semiconductor Detector Conference (RTSD)*, pp. 1–1, Oct. 2024, doi: https://doi.org/10.1109/nss/mic/rtsd57108.2024.10655744.

[6] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, Apr. 2004, doi: https://doi.org/10.1109/tip.2003.819861.

[7] OpenGATE Collaboration, "Introduction," *GATE documentation*, Feb. 07, 2025. https://opengate.readthedocs.io/en/latest/introduction.html

[8] Computation Research at Florida Tech, "AI.Panther," *Florida Tech*. https://research.fit.edu/craft/aipanther/ (accessed Feb. 24, 2025).

[9] H. Chang, V. Kobzarenko, and D. Mitra, "Inverse radon transform with deep learning: an application in cardiac motion correction," *Physics in Medicine & Biology*, vol. 69, no. 3, p. 035010, Jan. 2024, doi: https://doi.org/10.1088/1361-6560/ad0eb5.

[10] K. Daly, H. Eichner, P. Kairouz, M. H. Brendan, D. Ramage, and Z. Xu, "Federated Learning in Practice: Reflections and Projections," *arXiv (Cornell University)*, Oct. 2024, doi: https://doi.org/10.48550/arxiv.2410.08892.

[11] K. Bonawitz *et al.*, "Towards Federated Learning at Scale: System Design," *arXiv.org*, Mar. 22, 2019. http://arxiv.org/abs/1902.01046

[12] "Flower Framework Documentation," *Flower*. https://flower.ai/docs/framework/# (accessed Feb. 24, 2025).