universidade de aveiro
theoria poiesis praxis

**DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA**
**MESTRADO EM ENG. DE COMPUTADORES E TELEMÁTICA**
**ANO 2024/2025**

# REDES E SISTEMAS AUTÓNOMOS

# AUTONOMOUS NETWORKS AND SYSTEMS

## PRACTICAL GUIDE 3 – FEDERATED LEARNING

## Objectives

- Set up a Federated Learning (FL) cluster
- Use MobFedLS based on Flower to set up the FL cluster
- Perform the training in the clients and aggregation of model in the server
- Communication between clients and server is performed through WiFi ad-hoc network (batman)
- Observe the logs of the clients and the server to check the results of the federated learning training

## Duration

2 weeks

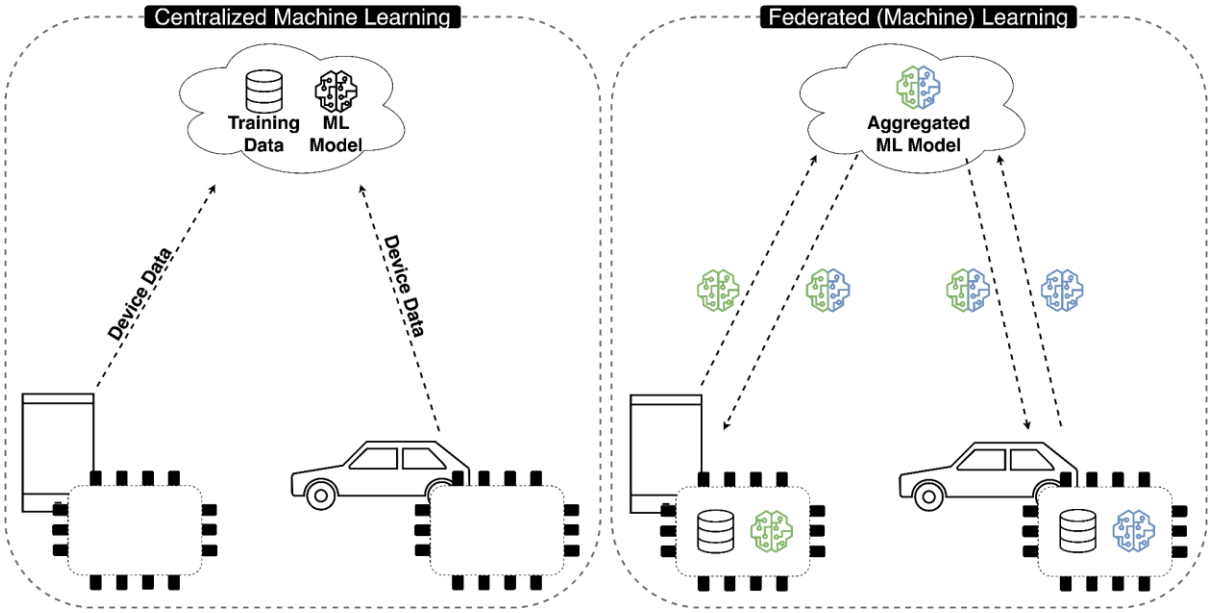
# Introduction - Theory Recap

## Classic Machine Learning (Centralized)

- In traditional machine learning, a model - such as a neural network or linear regression - is trained using data that originated from various and distributed sources such as smartphones, cars, or sensors. This data is gathered on a central server, often in a data center or cloud, where machine learning algorithms process it to perform tasks such as object detection, or playing a game such as Chess.
- While effective in cases where data is naturally centralized (e.g., web analytics), on cases where the data is not available on a centralized server or cases where the data available on one server is not enough to train a good model, this approach faces major **challenges**:
    - **Regulations** (e.g., GDPR) limit data transfer.
    - **User Privacy** expectations restrict data sharing.
    - **Data Volume** from high-frequency sensors makes centralized storage impractical and result in overhead in the network to send the data to the server.
- As a result, centralized machine learning is not feasible for many real-world applications where data is distributed across multiple devices or organizations.

## Federated Learning

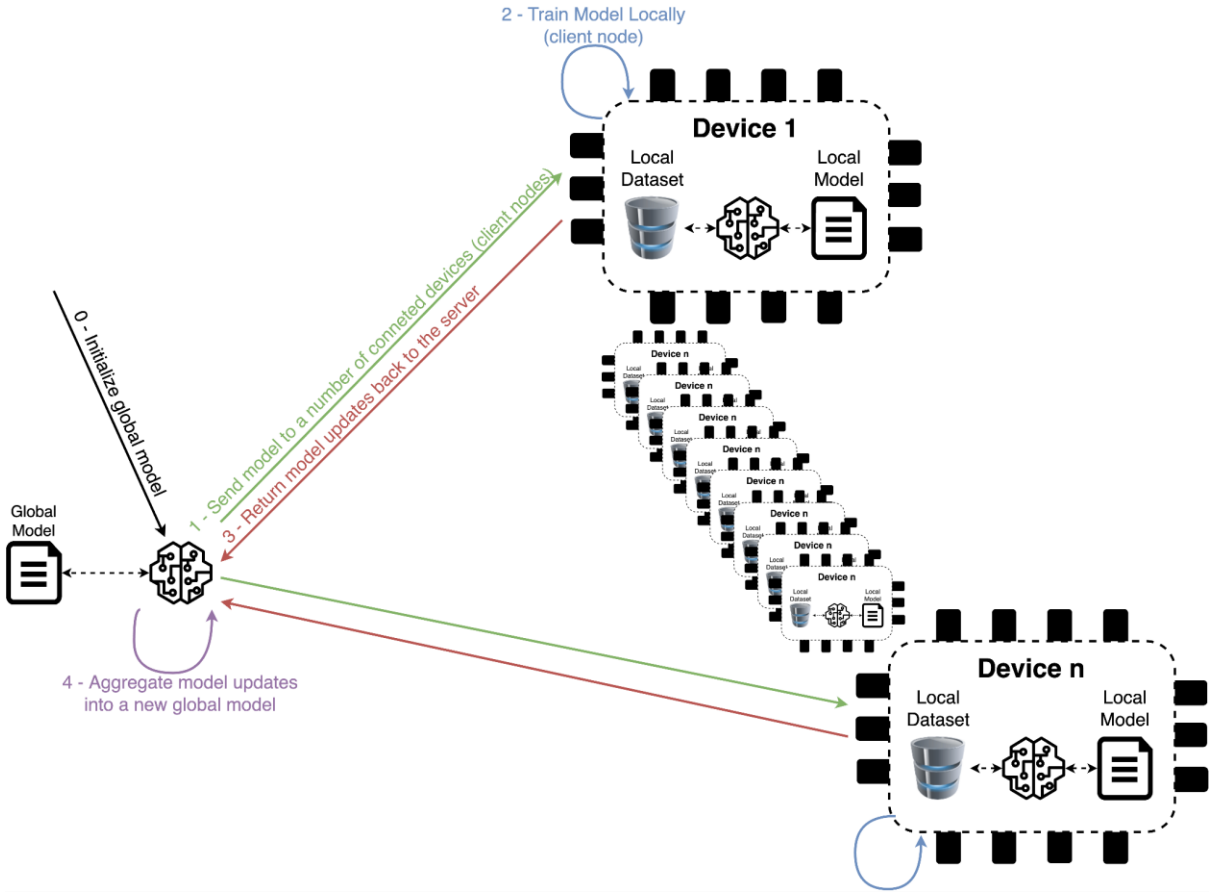
- Federated Learning flips the traditional approach by bringing computation to the data instead of moving data to a central server.
    - **Centralized Machine Learning**: Move data to the computation.
    - **Federated (Machine) Learning**: Move computation to the data.
- This approach unlocks machine learning in scenarios where data was previously inaccessible, enabling privacy-preserving and distributed model training across multiple devices or organizations.




- **Federated Learning in 5 Steps:**

0. **Initialize Global Model** - The server initializes the model parameters, either randomly or from a checkpoint.

1. **Send model to a number of connected devices (client nodes)** - The global model is distributed to a subset of client devices, ensuring all selected nodes start from the same parameters.
2. **Train model locally on the data of each device (client node)** - Each client trains the model on its own local data for a short period (e.g., one epoch or a few mini-batches).
3. **Return model updates back to the server** - Clients return their model updates, either as full parameters or gradients.
4. **Aggregate model updates into a new global model** - The server combines client updates using Federated Averaging (FedAvg) or another aggregation method to form a new global model.
5. **Repeat** - Steps 1 to 4 are what we call a single round of federated learning. Throughout the federated learning process there are several rounds.




- Since each client trains for only a short time per round, the aggregated model incorporates data from all participating nodes but remains partially trained. By repeating this process over multiple rounds, the model gradually improves until it performs well across all client data.

**In this guide, we will work with a framework built to operate several clients and a server in a Federated Learning scheme – the MobFedLS, based on Flower (https://doi.org/10.1016/j.future.2024.107514)**

# 1. The ML Model

1.1. In this guide you will use the MNIST dataset which consists in a collection of handwritten digit images (0-9). In a FL scenario, instead of storing the entire dataset in a central server, the data is distributed across multiple clients (such as different devices).

1.2. In this case each client holds a portion of the MNIST dataset and trains a local model on its own data. Each client loads its data from a config file, which contains only samples of certain digits assigned to that client. As discussed previously, instead of sharing raw data, clients only send model updates to a central server, which aggregates them to improve a global model.

1.3. So in other words, instead of one big dataset composed with all the digits, the clients will only have a small dataset composed with just some digits.

1.4. This dataset is broadly used to evaluate the performance and how the FL approach can be used.

# 2. Connecting your PC to the RPi board

2.1. Like in the previous guide, you will need to connect your PC to the RPi boards. Configure your PC with an IP in the **same LAN** of the RPi Ethernet interface (192.168.3.0/24) and connect an Ethernet cable between your PC and RPi. **Example**: 192.168.3.1/24 (**no** gateway needed)

2.2. Use ssh to connect to the RPi, using the IP and Username: `nap`, Password: `openlab`. In Windows use Putty or ssh through WSL. In Linux use ssh directly.

# 3. Prepare the Batman Network

3.1. Like in the previous guide, you will form batman networks in the classroom, each with at least 4 nodes (one RPi node per student). Run the script with the same settings as the other 3 students in your group (do not forget to change the IP, where the id is the *raspberrypi-7id*, channel number, frequency and SSID):

```
./setup_batman.sh wlan0 10.1.1.id/24 36 5180 rsatestbatman
```

3.2. Validate that all the nodes have connectivity (using the ping command).

# 4. Prepare and Launch the MobFedLS

4.1. Follow these steps:

```
cd ~/MobFedLS/deployment
vim .env
```

4.2. Edit the .env file, where **hostname** should be *raspberrypi-7id* **(**replace id with the id of your board), and in *configX.csv*, replace X with your assigned student number [1,4]:

```
MACHINE_ID=raspberrypi-7id
LOGGING_LEVEL=DEBUG
PLOT_PERFORMANCE=True
REGISTRY_URL=code.nap.av.it.pt:5050/ai4sme/mobfedlearnsys/
SERVER_IMG=flower-server
CLIENT_IMG=flower-ghostclient
ML_APP_IMG=ml-app-mnist
ML_BASE_DIR=mnist
DATASET=configX.csv
```

4.3. Now launch the base infrastructure containers of the framework, which are the *mfl-manager-ml-app-mnist-hostname*, *around-hostname*, *ml-app-mnist-hostname*, with:

```
cd ~/MobFedLS/deployment
docker compose up -d
```

4.4. Validate that all containers are up and running with:

```
docker compose logs -f
```

Keep this command running on the terminal to see the progress of your work.

4.5. Now open the browser in the following link, replacing **id** according to your board:

http://192.168.3.id:5001/docs - **ml-app API**
http://192.168.3.id:5101/docs - **mfl-manager API**

You should be able to see the 2 corresponding FastAPI documentation. We will explore the usage of several API endpoints to operate the MobFedLS (description of each one in the Appendix as well as the screenshots of what you are supposed to see at this point on your browser).

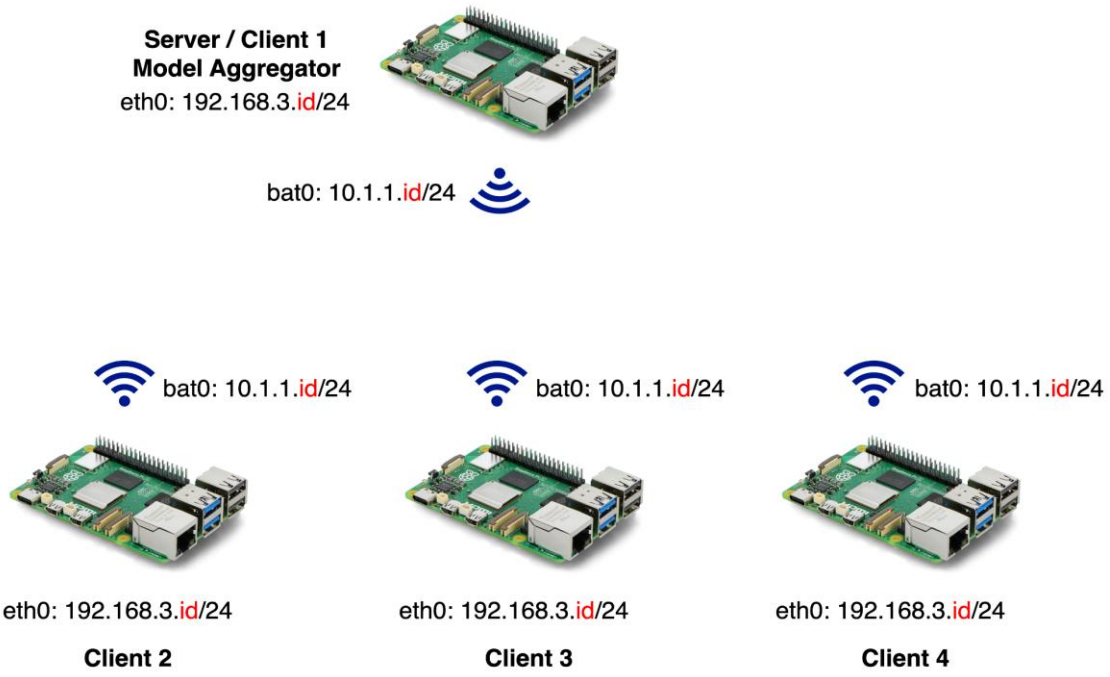
4.6. To do a first try of the tool, you will simulate that each node is already a running ML model that is fitted and it has context of the data produced by a "sensor".
In order to do this, you need to initialize the parameters according to the dataset of each client. Use the endpoint `/internal/get_data` and then the endpoint `/internal/fit` of the **ml-app API** to do an initial training of the model. Change only the client number field to your number on both endpoints and leave the rest default.

## 5. Clean the environment (Optional, if needed)

5.1. If there is some container with the images *flower-server* or *flower-ghostclient* hanging with an exit code, the environment must be cleaned. In order to do that, run the `/external/clean_environment` endpoint in the **mfl-manager API**.

## 6. Mount the first cluster of Federated Learning

6.1. First, discuss with the other students of your group, who will be the server node, to form the cluster.




6.2. Only on the server node will browse to the folder that contains the files with neighbours information that are passed to the around module

```
cd ~/MobFedLS/cmd/find-neighbours/cmd/findNeighbours/neighbours_lists/
```

6.3. Again only on the server node creates and modifies the neighbours file with `vim test1.json` where each line is the IP of the batman network interface of each node:

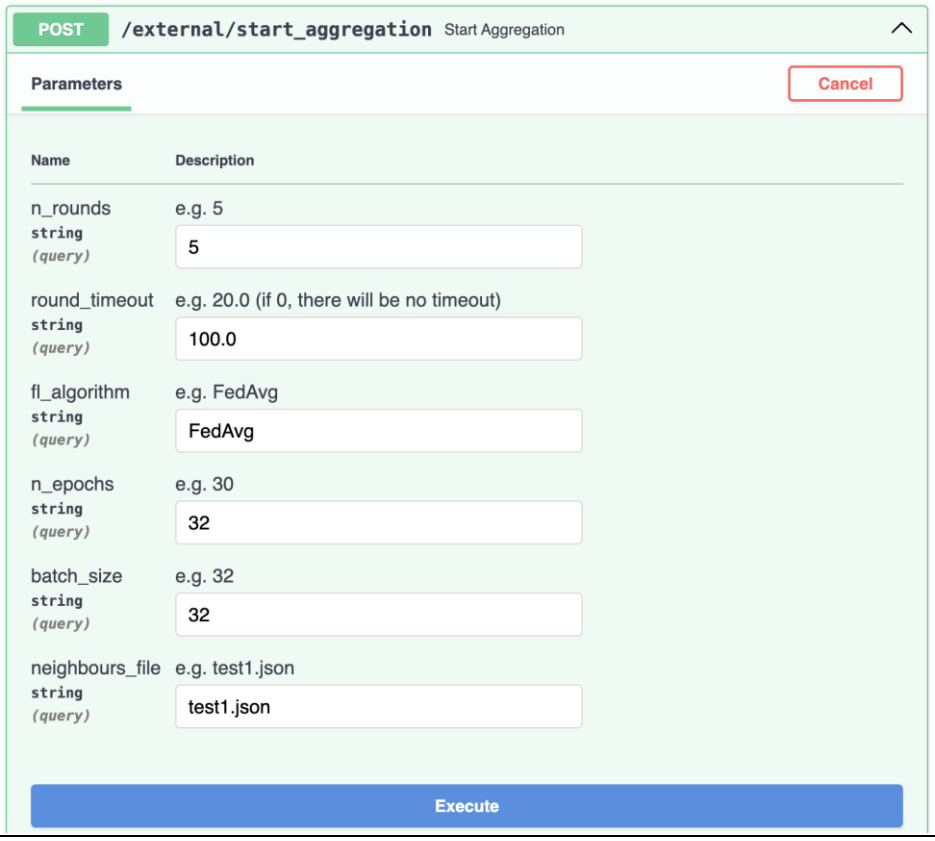
```
{
   "0": "10.1.1.id_server_node",
   "1": "10.1.1.id_server_node",
   "2": "10.1.1.id_student2",
   "3": "10.1.1.id_student3",
   "4": "10.1.1.id_student4"
}
```

6.4. In order to see the new containers being created, in parallel, in a new terminal window, run the following command on each node:

```
sudo watch -n 1 docker ps
```

6.5. In order to start the aggregation, the server node must use the endpoint `/external/start_aggregation` (**ml-app API**) and update the following fields with correct information, and after press the Execute button:

```
n_rounds: 5
round_timeout: 100.0
fl_algorithm: FedAvg
n_epochs: 32
batch_size: 32
neighbours_file: test1.json
```




## 7. Federated Learning Process Explanation with the MobFedLS

7.1. When the aggregation is started, and with all the nodes connected, the *ml-app* in the server node makes a request to the *mfls-manager* also in the server node, to trigger the process in the neighbours present in the neighbours file. This is made through a request to the corresponding *mfls-managers*. When the clients are up and running the Federated Learning Process happens between the now created server and clients. Check the ReadMe or **https://doi.org/10.1016/j.future.2024.107514** for more detail.

7.2. Explore the logs in the server node through the endpoint `/external/show_logs` in the Manager API. This endpoint will output the logs of the last run of the Federated Learning Process. and all the previous information of how the FL Process happens can be checked.

## 8. Evaluate the final/aggregated ML model

8.1. Generate plots with `/external/predict/local` and `/external/predict/aggregated` in the ML-App API with the fields **plot_graphs** and **centralized_predict** set to "*true*". This endpoint will create plots where you can observe the performance of the model to predict all the digits, before and after the FL process.

8.2. Use the scp command to copy the generated plots to your computer, so you can visualize them. Run the command on a terminal in your computer:

```
scp -r nap@192.186.3.id:~/MobFedLS/assets/logs/today_date/runX/ .
```

8.3. Additionally, look at the output of the request that you made to the endpoints where the real value of the accuracy metrics are written for each model. Check how different the accuracy values are before and after the aggregation.

8.4. **Note**: With this last comparison you can see in a Practical POV how the ML vs. FL approach works as discussed in the Theory Recap. Because before the aggregation process you train the ML model with each client separated, it is classic ML. Then, you trained with other clients and that is what we call FL.

## 9. Repeat the FL Process with other conditions

9.1. Now that the Initial experiment is done successfully, you will explore how the FL and the MobFedLS and see how this approach can be useful to overcome data or client problems.

9.2. **Note**: Before running the FL Process with different characteristics run the following commands `docker compose down` and then `docker compose up -d` (as in step 2.3). After that repeat the steps 2.4 (`docker compose logs -f`) to validate that the containers are up and running, and the step 2.6 (make a request to the endpoints `/internal/get_data` and `/internal/fit`) to initialize the parameters of each client model. For each of the following experiments repeat the steps 3, 4, 5 and 6:

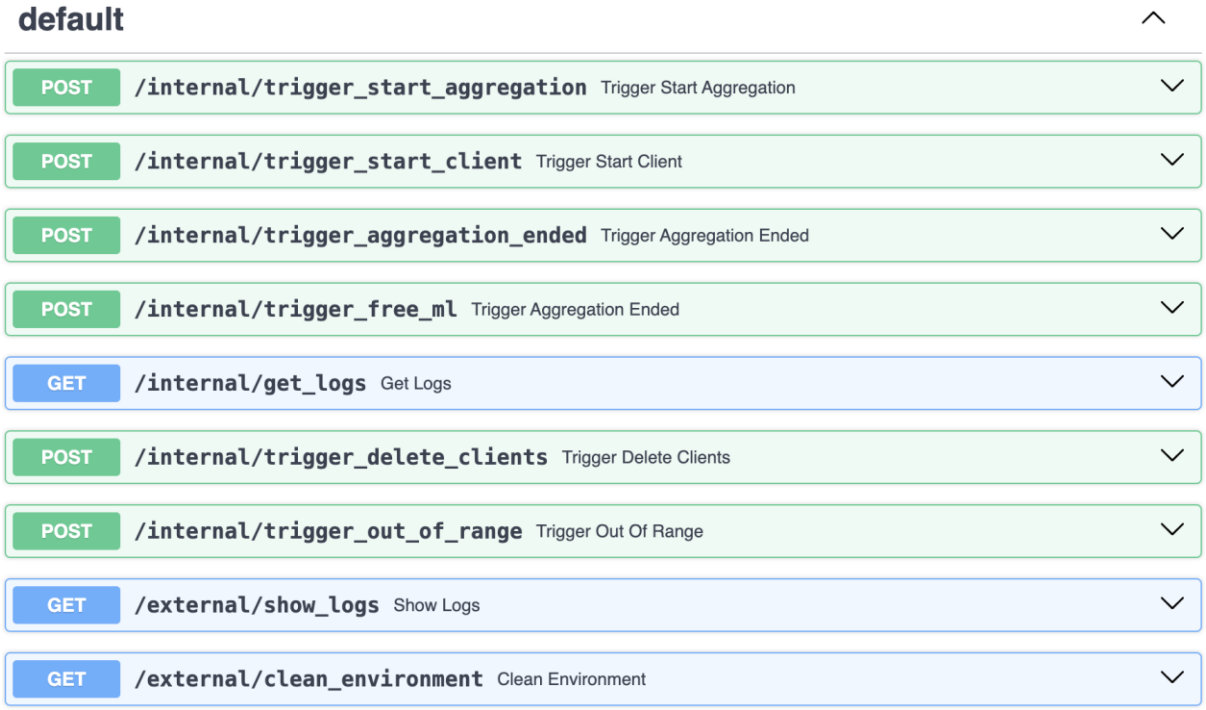
9.3. Connect 4 clients from the beginning (as you did on the initial experiment) and during the training process (when the ghost-client container is running) shutdown one of the boards. When one client is disconnected from the server in the middle of a Federation Round the server will wait until the *round_timeout* if one or more clients are not answering and only after the timeout is reached it proceeds to the next round. Check the terminal where you are running the command `docker compose logs -f` and also you will be able to see this in the logs when the process is finished. And again check the accuracy results and comment about the performance of the aggregated model in these conditions.

9.4. Now compare the results from the Experiment in 8 and this one in 9. You will be able to see that although there was a problem on the client (and here you can assume that or the node is not able to perform or the sensor that the client is representing has a malfunction

and the data is not accurate), the final model is able to predict almost as accurately as if the client did not had problem.

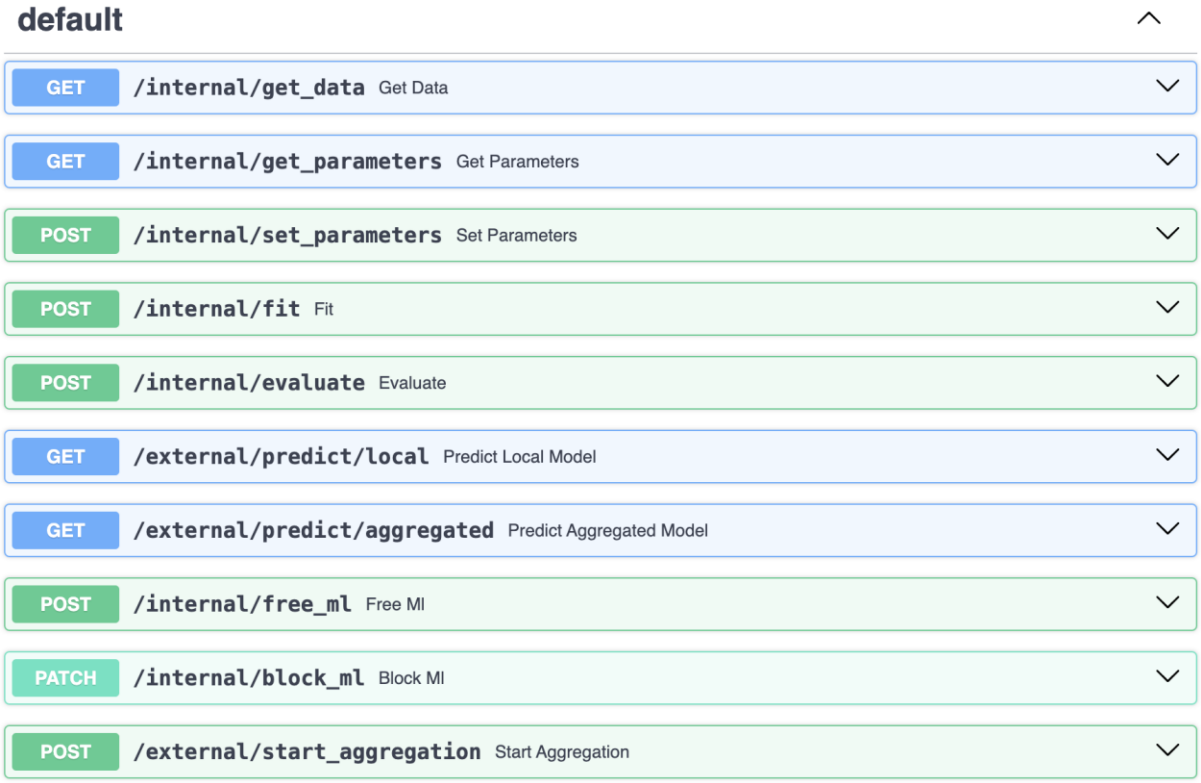
# Appendix

## API Documentation of *mfls-manager*




| Manager | | |
|---|---|---|
| **HTTP Method** | **Endpoint** | **Explanation** |
| POST | /internal/trigger_start_aggregation | This method is called by the ML-App when the respective MFL-Interface decides that wants to aggregate |
| POST | /internal/trigger_start_client | This method is used to start a MFL-GhostClient in a Mobile Clients |
| POST | /internal/trigger_aggregation_ended | This method is called by the MFL-Server to signal the Maestro's MFL-Manager that the FL process has ended |
| POST | /internal/trigger_free_ml | This method is used to signal the Mobiles MFL-Manager Mobiles to free the ML-App |
| GET | /internal/get_logs | This method is used to signal the Mobiles MFL-Manager to retrieve the logs of the MFL-GhostClient |
| POST | /internal/trigger_delete_clients | This method is used to signal the Mobiles MFL-Manager to delete the MFL-GhostClient |
| POST | /internal/trigger_out_of_range | This method is used by the MFL-GhostClient reaches a timeout without a connection from the MFL-Server |
| GET | /external/show_logs | This method is used to see the logs of previous FL runs |
| GET | /external/clean_environment | This method is used to clean the infraestructure at any time, if there is a MFL-Server or a MFL-GhostClient stopped with an error |

## API Documentation of *ml-app*




| ML-App | | |
|---|---|---|
| **HTTP Method** | **Endpoint** | **Explanation** |
| GET | /internal/get_data | This method is used by the MFL-GhostClient in order to prepare the dataset of the ML-App when the client is starting |
| GET | /internal/get_parameters | This method is used by to get the current parameters of the ML-App at any point |
| POST | /internal/set_parameters | This method is used by to set the current parameters on the ML-App at any point |
| POST | /internal/fit | This method is used in the FL process to train the ML-App |
| POST | /internal/evaluate | This method is used in the FL process to evaluate sets of parameters on the ML-App |
| GET | /external/predict/local | This method is used to predict using the set of parameters before the FL process |
| GET | /external/predict/aggregated | This method is used to predict using the set of parameters after the FL process |
| POST | /internal/free_ml | This method is used by the MFL-Manager to signal that the ML-App can be free |
| POST | /external/start_aggregation | This method is used by the ML-App when it decides that wants to start an aggregation |