Robert E Dawkins

North Carolina A&T State University

Federated Learning Tutorial with

Flower AI

Table of Contents

[Introduction 2](#_Toc184162504)

[Pre-requisite 2](#_Toc184162505)

[Instillation 2](#_Toc184162506)

[Using Flower AI Framework 3](#_Toc184162507)

[Dataset 3](#_Toc184162508)

[Partitioners 3](#_Toc184162509)

[Federated Datasets 4](#_Toc184162510)

[Visualization 5](#_Toc184162511)

[Flower Client 5](#_Toc184162512)

[The Model 5](#_Toc184162513)

[The Client 6](#_Toc184162514)

[Global Server 8](#_Toc184162515)

[Configuration Management 8](#_Toc184162516)

[Strategy 8](#_Toc184162517)

[Running simulation 9](#_Toc184162518)

[Simulation Evaluation 9](#_Toc184162519)

[Custom Metrics 10](#_Toc184162520)

[Visualization 11](#_Toc184162521)

[Evaluating Flower AI Framework 12](#_Toc184162522)

[Advantages 12](#_Toc184162523)

[Disadvantages 13](#_Toc184162524)

[Conclusion 14](#_Toc184162525)

[References 14](#_Toc184162526)

# Introduction

Federated Learning is privacy preserving machine learning technique that mitigates data leakage by keeping training data decentralized and within their respective data sources. Federated learning seeks to create a global model that leverages the learning of distributed model that performs better than any localized models.

Flower AI Framework is a python framework that seeks to simplify the implementation of Federated Learning by abstracting the complexities of Federated learning like networking, through providing of commonly used classes.

## Pre-requisite

Flower Framework requires at least Python 3.9, but Python 3.10 or above is recommended.

To mitigate potential issues during instillation it is recommend installing, ray a Flower Framework dependency directly. The following pip install command can be used to install the package via command line:

pip install -q ray

## Instillation

Flower can be installed via pip or conda / mamba.

The following pip install command can be used to install the package via command line:

pip install -q flwr flwr\_datasets

The following conda install command can be used to install the package via command line:

conda install -q flwr flwr\_datasets

The following mamba install command can be used to install the package via command line:

mamba install -q flwr flwr\_datasets

Once installed the following python command can be used to verify if Flower was installed successfully.

python -c “ import flwr;print(flwr.\_\_version\_\_)”

# Using Flower AI Framework

## Dataset

Flower AI is a machine learning framework that enables users to easily run federated learning simulations. One key component of federated learning is dataset isolation by the clients. With this Flower AI provides tools to partition your data for it to be used in a federated learning simulation.

### Partitioners

Data in federated environment can be either independent and identically distributed amongst the clients (IID), meaning they have they have the roughly the same class distribution, or in a NON-IID fashion.

#### IID Partitioner

When running a federated simulation where data amongst the clients is IID we can utilize the built in IidPartitioner class which contains the logic to divvy the dataset into the number of partitions you provide

from flwr\_datasets.partitioner import IidPartitioner

partitioner = IidPartitioner(num\_partitions=10)

#### Non IID Partitioner

When running a federated simulation where data amongst the clients is Non-IID there a slew of different built in class you can utilize that will partition your data using various distribution algorithms. For this example, we will utilize SizePartitioner in which we will supply the number of samples per partition via an array, but will not guarantee the distribution of each partition will be identical

from flwr\_datasets.partitioner import SizePartitioner

partition\_sizes = [15\_000, 5\_000, 30\_000]

partitioner = SizePartitioner(partition\_sizes)

### Federated Datasets

Machine learning simulation are all about datasets the models are trained on. With Flower AI we can utilize our own data, as to be expect, or we can easily utilize preexisting datasets from the Hugging Face Hub.

#### Local Dataset

Although not necessary, it is recommended that we convert our dataset into a Hugging Face dataset for ease of use and compatibility with Flower AI’s FederatedDataset class.

We can utilize either local files or in-memory data to create our dataset.

##### Local Files

Hugging Face dataset support csv, json, images, and audio files. For this example, we will utilize csv files for our data source

from datasets import load\_dataset

from flwr\_datasets import FederatedDataset

data\_files = ["path-to-my-file-1.csv", "path-to-my-file-2.csv", ...]

dataset = load\_dataset('csv', data\_files=data\_files)

fds = FederatedDataset(dataset, partitioners={'train': partitioner})

##### Preexisting Hugging Face Dataset

It is common in machine learning to repeat or extend experiment. Hugging Face provides a repository for researchers to upload their datasets for said very reason. Flower AI makes it very easy to consume these preexisting Hugging Face datasets by simple providing the dataset name, as shown in the following example:

from flwr\_datasets import FederatedDataset

hugging\_face\_dataset\_name = 'mnist'

fds = FederatedDataset(dataset=hugging\_face\_dataset\_name, partitioners={'train': partitioner})

### Visualization

Visualization is a powerful to in a data scientist/machine learning engineers tool belt. It can help us make intuitive sense about data that would otherwise be comprehendible. Flower AI provides a built-in function to visualize the label distribution

from flwr\_datasets.visualization import plot\_label\_distributions

\_ = plot\_label\_distributions(

partitioner=fds.partitioners["train"],

label\_name="label",

legend=True,

)

A chart of different colored bars

Description automatically generated

## Flower Client

### The Model

The core of machine learning is the model. This holds true even in a federated learning environment. Flower AI does not put any limitations around models that can be utilized. For this example, we will create a simple function that returns a simple Convolutional Neural Network:

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dropout, Dense

from tensorflow.keras.optimizers import Adam

def load\_model(learning\_rate: float = 0.001, verbose = False):

model = Sequential(

[

Input(shape=(28, 28, 1)),

Conv2D(32, kernel\_size=(3, 3), activation="relu"),

MaxPooling2D(pool\_size=(2, 2)),

Conv2D(64, kernel\_size=(3, 3), activation="relu"),

MaxPooling2D(pool\_size=(2, 2)),

Flatten(),

Dropout(0.5),

Dense(10, activation="softmax"),

]

)

optimizer = Adam(learning\_rate)

model.compile(

optimizer=optimizer,

loss="sparse\_categorical\_crossentropy",

metrics=["accuracy"],

)

if verbose:

model.summary()

return model

### The Client

In a Federated Learning environment, the client is responsible for the training and evaluation of local model on localized data. Below is an example of setting up the client class with Tensorflow to achieve this:

from flwr.client import NumPyClient

class FlowerClient(NumPyClient):

def \_\_init\_\_(self, client\_id, model, data, verbose):

self.clinet\_id = client\_id

self.model = model

self.x\_train, self.y\_train, self.x\_test, self.y\_test = data

self.verbose = verbose

def fit(self, parameters, config):

self.model.set\_weights(parameters)

history = self.model.fit(

self.x\_train,

self.y\_train,

epochs=config['num\_local-epochs'],

batch\_size=config['batch-size'],

verbose=self.verbose,

)

res = {key: values[-1] for key, values in history.history.items()}

res["client-id"] = self.client\_id

res["current-round"] = config["current-round"]

return self.model.get\_weights(), len(self.x\_train), res

def evaluate(self, parameters, config):

self.model.set\_weights(parameters)

loss, accuracy = self.model.evaluate(self.x\_test, self.y\_test, verbose=0)

return loss, len(self.x\_test), {

"client-id": self.client\_id,

"current-round": config["current-round"],

"loss": loss,

"accuracy": accuracy

}

Note: The final dictionary in the return values of both the fit and evaluate function contain metrics that can be used to evaluate the performance of the model for each client sampled in each round

To build each client we define a client\_fn to be utilized by the simulation:

from flwr.common import Context

from flwr.client import Client

config = {

'num\_clients' : 10,

'num\_communication-rounds': 10,

'num\_local-epochs' : 10,

'learning-rate' : 0.01,

'batch-size': 128,

'client-sample\_fit': 1,

'client-sample\_evaluate': 1,

}

def client\_fn(context: Context) -> Client:

learning\_rate = config["learning-rate"]

verbose = config.get("verbose")

cnn = load\_model(learning\_rate, verbose)

partition\_id = context.node\_config["partition-id"]

data = fds.load\_partition(int(partition\_id))

return FlowerClient(partition\_id, cnn, data).to\_client()

## Global Server

In a Federated Learning environment, the global server is responsible to for orchestration of the network. It will broadcast the global model to all clients in the network, and sample clients in the network then aggregate their learnings into the global model, utilizing a sampling and aggregation strategy.

### Configuration Management

We often run simulations multiple times as we tweak various parameters to find the ideal ones that give us the best performance. With that it is best that we variablize these parameters such that we only need to change them in a single location. To do this we can define a config\_fn that we will pass our configurations into each client during the fit and evaluation steps of the learning. The config\_fn requires an integer as parameter. This integer represents the server communication round, enabling more dynamic configuration round to round if needed. For this example, we will keep it simple and return a static configuration

from typing import Dict

def config\_fn(server\_round: int) -> Dict[str, int]:

config['current-round'] = server\_round

return config

### Strategy

As mentioned before the global server is responsible for sampling clients and aggregating their learnings into a singular global model. There are many different sampling and aggregation algorithms, FlowerAI makes it simple for us by providing a set of prebuilt classes that implement common Federated Learning strategies. Below is an example of utilizing the FedAvg strategy:

from flwr.server.strategy import FedAvg

from flwr.common import ndarrays\_to\_parameters

server\_config = config\_fn(-1)

initial\_model = load\_model(learning\_rate=server\_config['learning-rate'])

strategy = FedAvg(

fraction\_fit=server\_config["client-sample\_fit"],

fraction\_evaluate=server\_config["client-sample\_evaluate"],

on\_fit\_config\_fn=config\_fn,

on\_evaluate\_config\_fn=config\_fn,

initial\_parameters=ndarrays\_to\_parameters(initial\_model.get\_weights()),

)

## Running simulation

FlowerAI provides a simple function for kicking off the simulation that returns history object that we can utilize to review the performance of the model throughout the simulation

from flwr.simulation import start\_simulation

from flwr.server import ServerConfig

history = start\_simulation(

client\_fn=client\_fn,

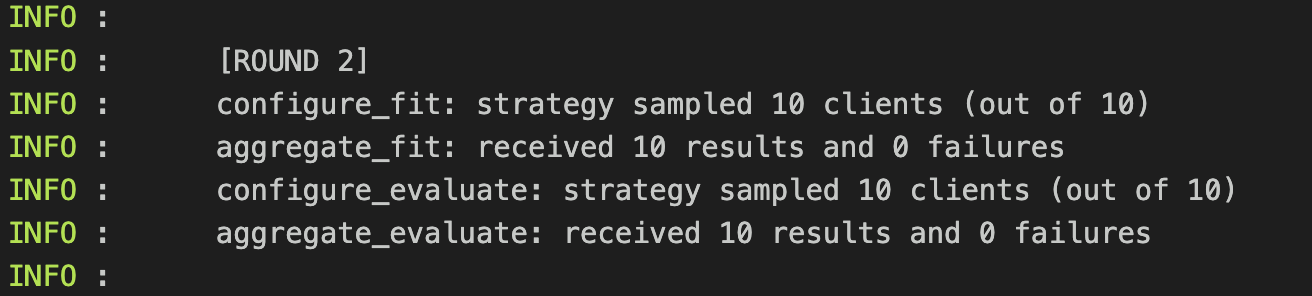
num\_clients=server\_config['num\_clients'],

config=ServerConfig(num\_rounds=server\_config['num\_communication-rounds']),

strategy=strategy

)

As the simulation is running FlowerAI will provide you a snapshot of whats happening each communication rounds. The snapshot will include the number of clients sampled for fit, the number of clients that failed during fit, the number of clients sampled during evaluation and the number of clients failed during evaluations.



## Simulation Evaluation

At the conclusion of the simulation FlowerAI will provide us a summary of the performance of the model throughout the training. By default, this will only provide us with the average loss across all sample client during evaluation per round.

A black screen with white text

Description automatically generated

### Custom Metrics

To collect more metrics beyond just the average loss during training per round we will need to supply a custom aggregation function. The flexibility of FlowerAI allows us the ability specify unique aggregation functions for the fit and evaluation portion of the learnings individually. For this example, we will write a common aggregation function to utilize during both fit and evaluation.

from typing import List, Tuple, Dict

import numbers

def metrics\_aggregation\_fn(

results: List[Tuple[int, Dict[str, float]]]

) -> Dict[str, float]:

metrics\_sum = {}

total\_examples = 0

for num\_examples, metrics in results:

total\_examples += num\_examples

for metric, value in metrics.items():

if not isinstance(value, numbers.Number):

continue

if metric not in metrics\_sum:

metrics\_sum[metric] = 0.0

metrics\_sum[metric] += value \* num\_examples

averaged\_metrics = {metric: value / total\_examples for metric, value in metrics\_sum.items()}

return averaged\_metrics

To utilize this aggregator, we pass it into the fit\_metrics\_aggregation\_fn and evaluate\_metrics\_aggregation\_fn parameters of the global server strategy.

strategy = FedAvg(

fraction\_fit=server\_config["client-sample\_fit"],

fraction\_evaluate=server\_config["client-sample\_evaluate"],

on\_fit\_config\_fn=config\_fn,

on\_evaluate\_config\_fn=config\_fn,

fit\_metrics\_aggregation\_fn=metrics\_aggregation\_fn,

evaluate\_metrics\_aggregation\_fn=metrics\_aggregation\_fn,

initial\_parameters=ndarrays\_to\_parameters(initial\_model.get\_weights()),

)

Now at the conclusion of the simulation the summary provided by FlowerAI will include addition metrics such as average accuracy per round for both the fit and evaluation phase of the learning.

A computer screen shot of a black screen

Description automatically generated

### Visualization

Now that we have additional metrics, we can leverage the history object return from the start\_simulation function to visualization the performance of our model over the course of communication rounds. For this example, we are utilizing matplotlib for visualization.

import matplotlib.pyplot as plt

rounds = range(1, config['num\_communication-rounds']+1)

fig, (ax1, ax2 )= plt.subplots(2, 1, sharex=True)

ax1.plot(rounds, dict(history.metrics\_distributed['loss']).values(), label='Evaluation Loss', marker='o')

ax1.plot(rounds, dict(history.metrics\_distributed\_fit['loss']).values(), label='Fit Loss', marker='x')

ax1.set\_ylabel('Loss')

ax1.legend()

ax1.set\_title('Loss Over Rounds')

ax2.plot(rounds, dict(history.metrics\_distributed['accuracy']).values(), label='Evaluation Accuracy', marker='o')

ax2.plot(rounds, dict(history.metrics\_distributed\_fit['accuracy']).values(), label='Fit Accuracy', marker='x')

ax2.set\_xlabel('Round')

ax2.set\_ylabel('Accuracy')

ax2.legend()

ax2.set\_title('Accuracy Over Rounds')

plt.tight\_layout()

plt.show()

# Evaluating Flower AI Framework

## Advantages

The key advantages of the Flower AI Framework are its flexibility, simplicity, and extensibility.

### Flexibility

Flower AI Framework most impactful advantage is its flexibility. Although, for this example I chose to implement my model utilizing Keras via Tensorflow, the Flower AI Framework is model framework agnostic. To clarify, Flower AI does not dictate which underlying framework data scientist and engineers must use when designing models with Flower AI. Flower AI is developed in such a way that it is compatible with all the common machine learning frameworks. From building a model from scratch using numpy to PyTorch, Flower AI robust documentation on how to get started using the model framework of your choice. This notion of flexibility with Flower AI extends to other components such as infrastructure configuration. Because Flower AI uses Ray as its AI compute engine under the hood, it provides us with the same flexibility in optimizing things like type of processing units to run the simulation on, and how much compute power should be allocated to each client and much more.

Flower AI enables scientist the ability to run simulation in the environment of their choice. Flower AI has support for running in either a Python Notebook like Jupyter or Google Colab, or as a standalone python application initiated via the command line interface.

### Simplicity

Often when designing frameworks, we must compromise simplicity for flexibility. Flower AI framework does a good job at keeping the framework flexible without making it too complicated to use. They achieve this through the abstraction of some of the more complicated mechanism of Federated Learning like networking and client authentication. They also implemented many commonly used aggregation algorithms such as FedAvg, FedAdam and more. In doing so, they significantly reduce the time it would take to get someone unfamiliar with Federated Learning to ramp up and implement something meaningful in a Federated Learning Environment.

### Extensibility

Although there are many different commonly used algorithms and features implemented within the Flower AI framework, you are not locked into only just that. Flower AI framework provides base classes for engineers to extend in order to implement custom features such as custom model aggregation strategies, client specific behavior, metric aggregation algorithms and more.

##### Support

Flower AI is one of the more recent additions to the Federated Learning space and thus does not have the largest of communities, however, I would be remiss not to mention that during my discovery of finding a FL framework for my research, I often found that other frameworks such as TensorFlow Federate simply did not work on the Macs with Apple Silicon, and I needed to dive deep into the weeds of forums to uncover this limitation. With Flower AI I was able to get a sample project running on my Apple Silicon Mac without issue, which I’m attributing to Flower AI recent emergence.

## Disadvantages

There are two current disadvantages with Flower AI, absences of model performance monitoring, and limited federated learning architecture.

### Model Metrics

As shown in this tutorial, by default, there is little to no metrics provided about the model’s performance beyond the calculation of the loss for each round. To get additional metrics we had to create a custom aggregation function and inject it into the strategy. Then and only then were we able see additional metrics such as accuracy. Providing a base aggregation function that can be extended include addition metrics, or replaced with a custom metric aggregation function feels more in line with the flexibility and simplicity of the rest of the framework.

I also found that the Flower AI framework fell short in terms of built in visualization. Yes, the FederatedDataset package provided a function to visualize our data, but that was it. Any additional metrics that needed to visualize about the model’s performance needed to be custom created. Considering we typically evaluate Federate Learning model based on its performance over communication rounds, I would’ve expected a simple function that would’ve taken the output of the custom aggregation function, mention previously, and visualize them on a line plot.

### Limited Federated Learning Architecture

As of right now, Flower AI framework only supports a centralized Federated Learning Architecture in which there is a singular global server that orchestrates the learning in the network. However, as the field of Federated Learning is expanding researchers are design different types of Federated Learning architectures, ones that are more decentralized such as a clustered architecture or peer to peer architecture. Although it is indeed possible to implement these types of architectures in Flower AI, it requires extensive work around that may make the code base unmaintainable.

## Conclusion

In conclusion, while Flower AI offers significant advantages in terms of flexibility, simplicity, and extensibility, its current limitations in model performance monitoring and federated learning architecture may hinder its suitability for certain advanced use cases. However, as the field of federated learning continues to evolve, Flower AI's ongoing development and active community may help address these shortcomings, making it a strong candidate for federated learning projects today.

# References

**Flower Framework Documentation:**

<https://flower.ai/docs/framework/>

**Flower Datasets**

<https://flower.ai/docs/datasets/>

**FL Simulation with Flower**

<https://www.youtube.com/watch?v=cRebUIGB5RU&list=PLNG4feLHqCWlnj8a_E1A_n5zr2-8pafTB>