FEDERATED LEARNING

Presented by Vidit Khandelwal

Overview

- Introduction
- Flower
- Implementation
- Flower
- Results
- Results
- Conclusion
- Flower

Introduction

Centralized Learning is a paradigm of Machine Learning that has been widely explored and utilized. But, due to the massive amount of computations required, it requires High Performance Computers and Cloud Computers for its training procedure, along with the need to collect huge amounts of data from devices across the globe.

Federated Learning provides an interesting approach to this problem of machine learning. Rather than collecting the data and bringing it to server, it performs training on many edge devices and aggregates the changes in model parameters to train the model.

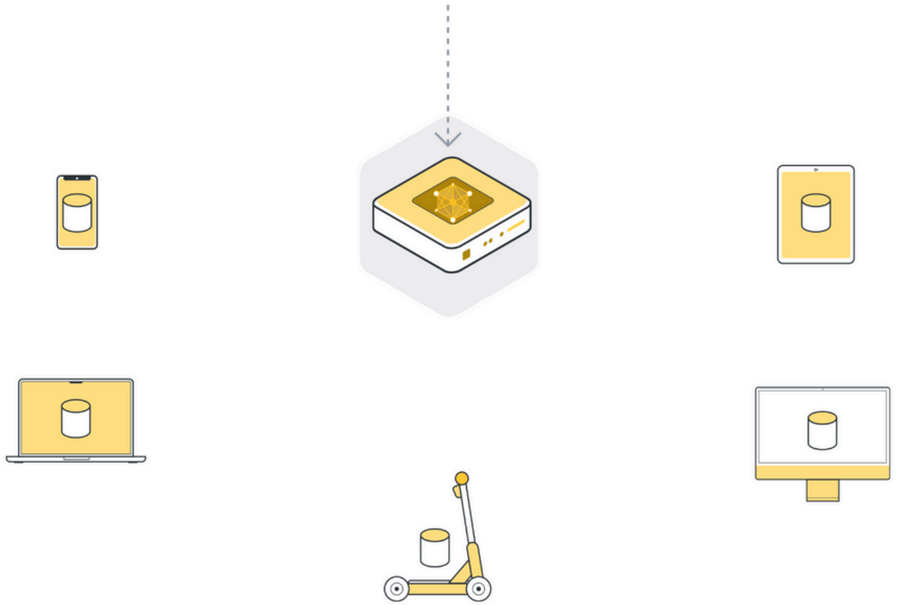
Problem

- We have one server, and N clients. Each client is an edge device that will have a locally trained model. Only these changes in parameters of model are transmitted back to the server.

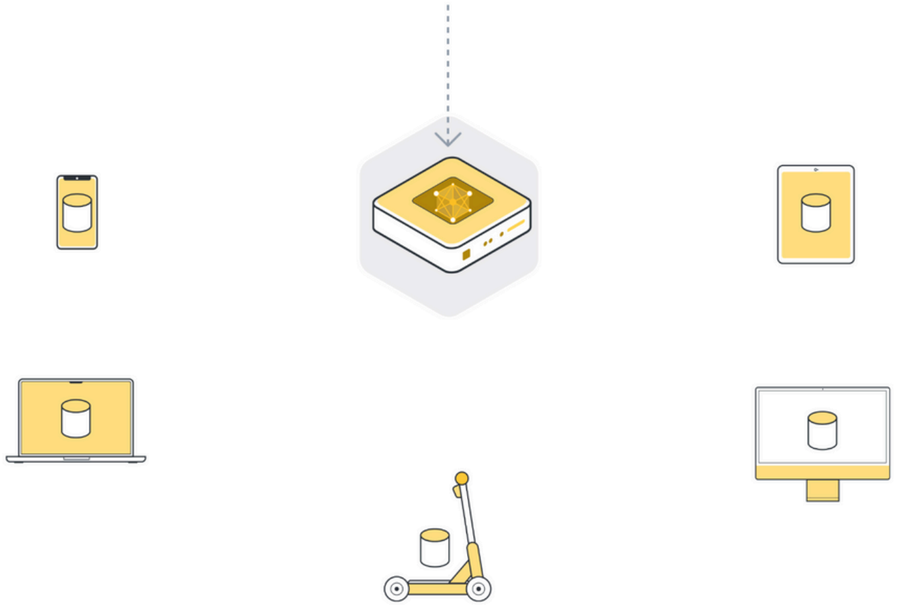
Thus, federated learning has advantages of:

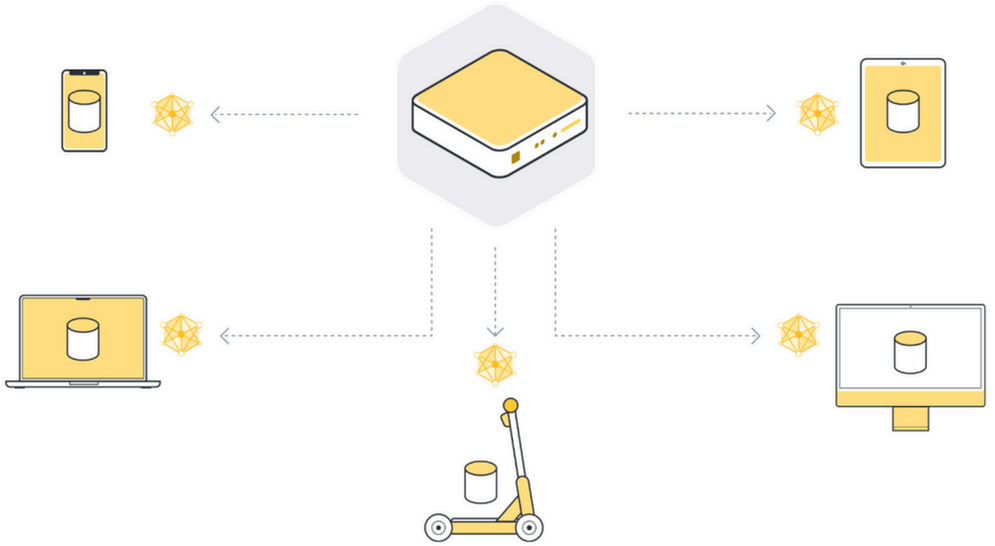
- Privacy, data stays at edge devices
- Diverse data from multiple devices
- Data security,

Step 1. Initialize Global model

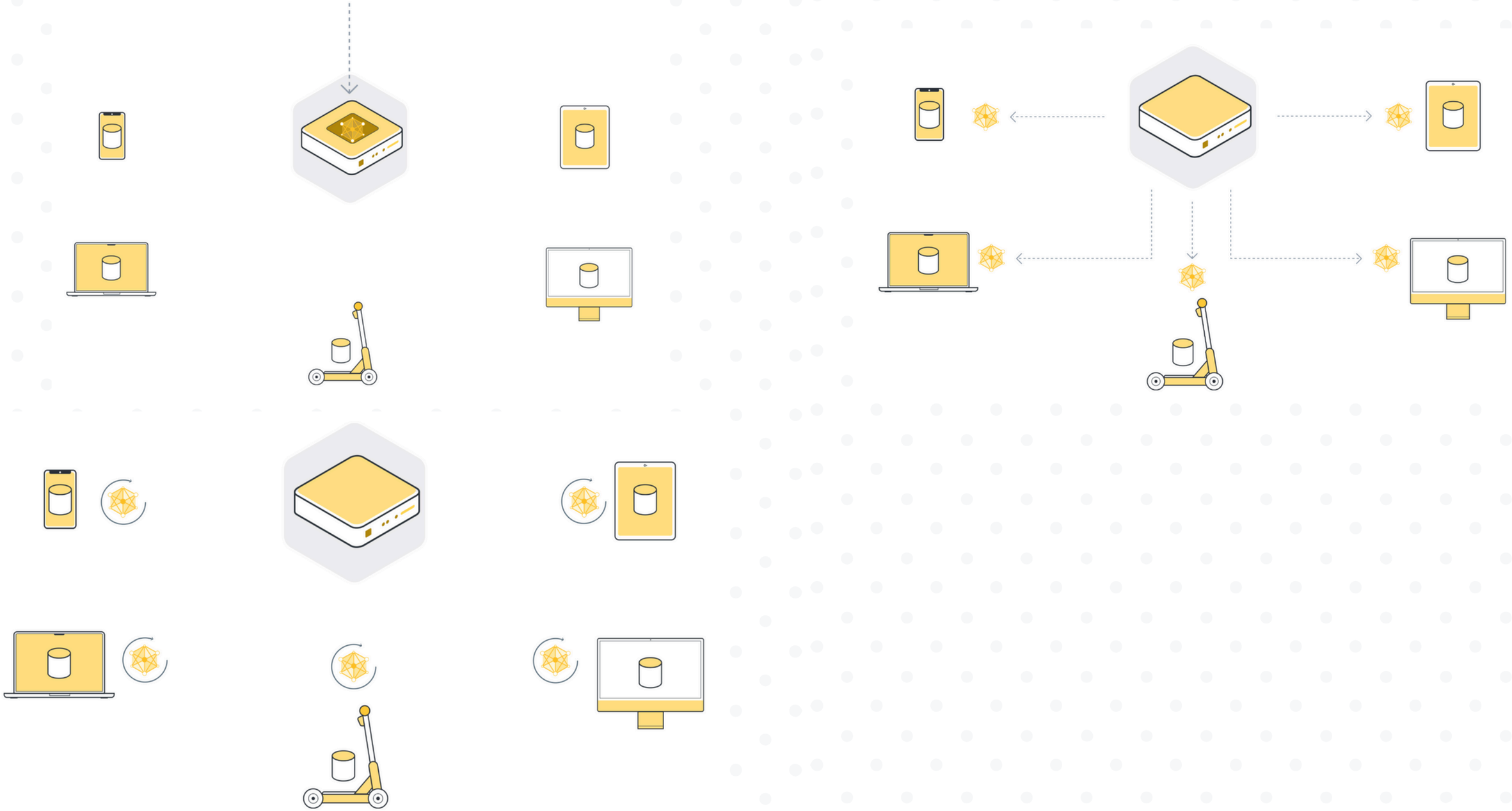


Step 2. Send model to a number of clients

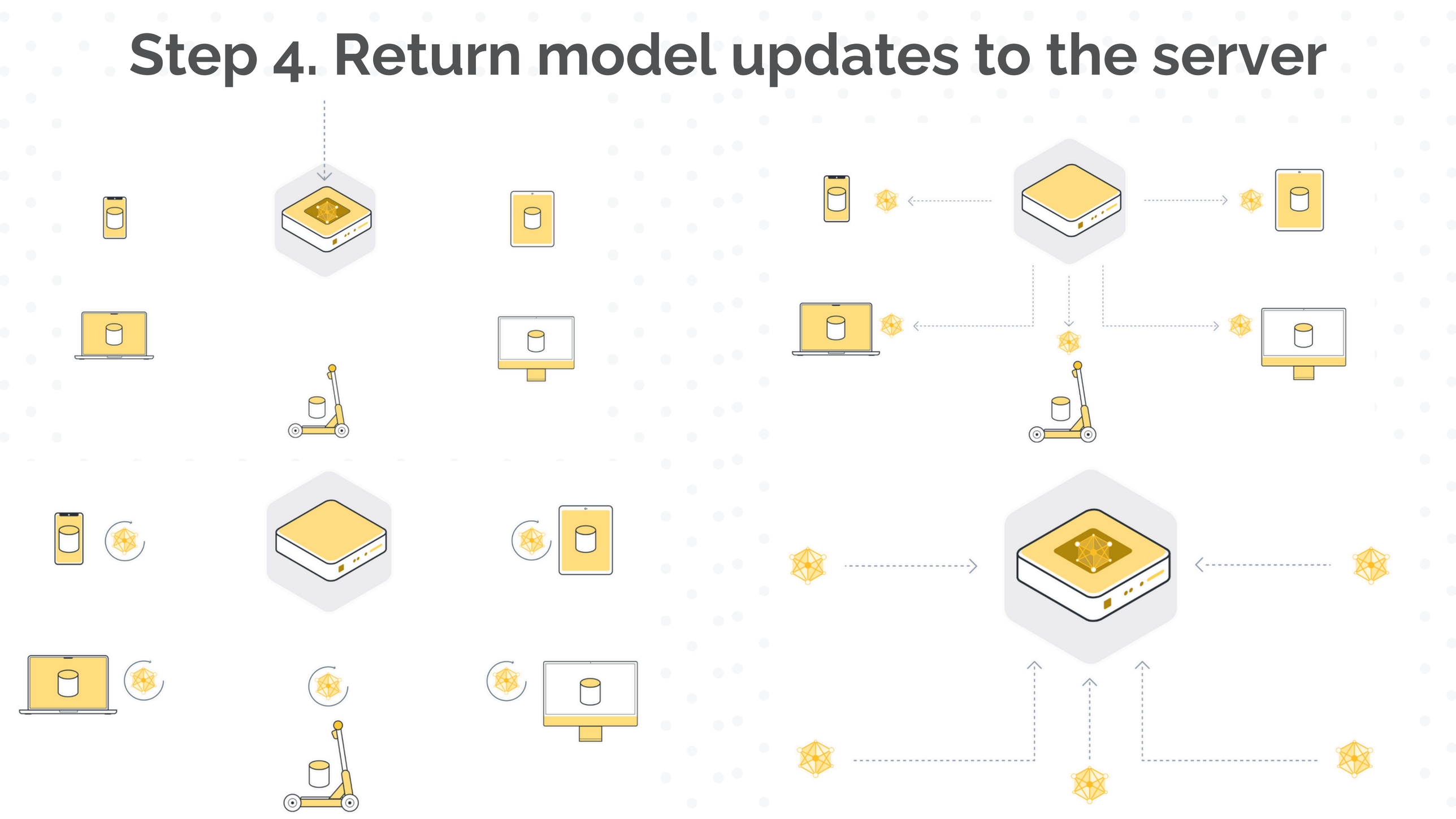




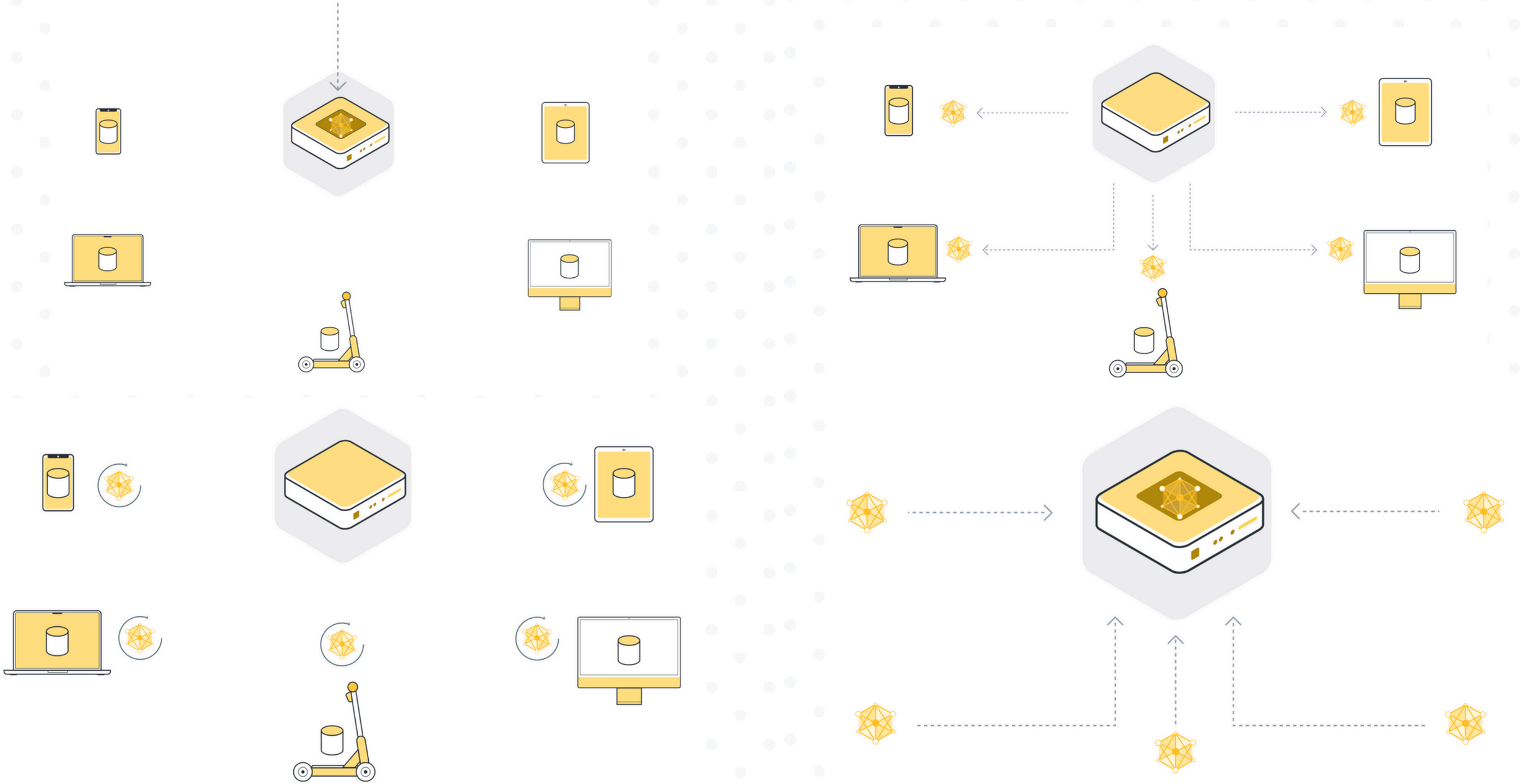
Step 3. Train the model locally on edge devices



Step 4. Return model updates to the server



Step 5. Repeat steps 1-4 for N iterations



Flower: A Friendly Federated AI Framework

Website | Blog | Docs | Conference | Slack

Flower (`flwr`) is a framework for building federated AI systems. The design of Flower is based on a few guiding principles:

- **Customizable**: Federated learning systems vary wildly from one use case to another. Flower allows for a wide range of different configurations depending on the needs of each individual use case.
- **Extendable**: Flower originated from a research project at the University of Oxford, so it was built with AI research in mind. Many components can be extended and overridden to build new state-of-the-art systems.
- **Framework-agnostic**: Different machine learning frameworks have different strengths. Flower can be used with any machine learning framework, for example, PyTorch, TensorFlow, Hugging Face Transformers, PyTorch Lightning, scikit-learn, JAX, TFLite, MONAI, fastai, MLX, XGBoost, Pandas for federated analytics, or even raw NumPy for users who enjoy computing gradients by hand.
- **Understandable**: Flower is written with maintainability in mind. The community is encouraged to both read and contribute to the codebase.

Objective

1. Using the Flower framework, simulate the process of Federated Learning.
2. Change the model and see changes
3. Change the dataset and show that federated learning still works
4. Compare the performance of federated with centralized learning
5. Finally work on prototype portraying FL

1.Implementing FL using Flower

A simple CNN model that can be trained to for classification of images

```python
def get_model(width: int = 32, height: int = 32, num_channels: int = 3) -> Any:

    model = models.Sequential([
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        layers.BatchNormalization(),
        layers.Conv2D(32, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.BatchNormalization(),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(10, activation='softmax')
    ])
    model.compile("adam", "sparse_categorical_crossentropy", metrics=["accuracy"])
    return model
```

CIFAR-10: an image classification dataset

```python
def load_data(
    partition_id: int, num_partitions: int
) -> tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]:
    """Load data with Flower Datasets (CIFAR-10)."""
    # Only initialize `FederatedDataset` once
    global fds
    if fds is None:
        partitioner = IidPartitioner(num_partitions=num_partitions)
        fds = FederatedDataset(
            dataset="uoft-cs/cifar10",
            partitioners={"train": partitioner},
        )
    partition = fds.load_partition(partition_id, "train")
    partition.set_format("numpy")

    # Divide data on each node: 80% train, 20% test
    partition = partition.train_test_split(test_size=0.2, seed=42)
    x_train, y_train = partition["train"]["img"] / 255.0, partition["train"]["label"]
    x_test, y_test = partition["test"]["img"] / 255.0, partition["test"]["label"]

    return x_train, y_train, x_test, y_test
```

The **SERVER** function defines the aggregation fn used for
FL, here used FedAvg

```python
def server_fn(context: Context) -> ServerAppComponents:
    """Construct components that set the ServerApp behaviour."""

    # Read from config
    num_rounds = context.run_config["num-server-rounds"]

    model = get_model()
    ndarrays = get_parameters(model)
    global_model_init = ndarrays_to_parameters(ndarrays)

    # Define strategy and the custom aggregation function for the evaluation metrics
    strategy = FedAvg(
        evaluate_metrics_aggregation_fn=average_metrics,
        initial_parameters=global_model_init,
    )
    config = ServerConfig(num_rounds=num_rounds)

    return ServerAppComponents(strategy=strategy, config=config)


# Create ServerApp
app = ServerApp(server_fn=server_fn)
```

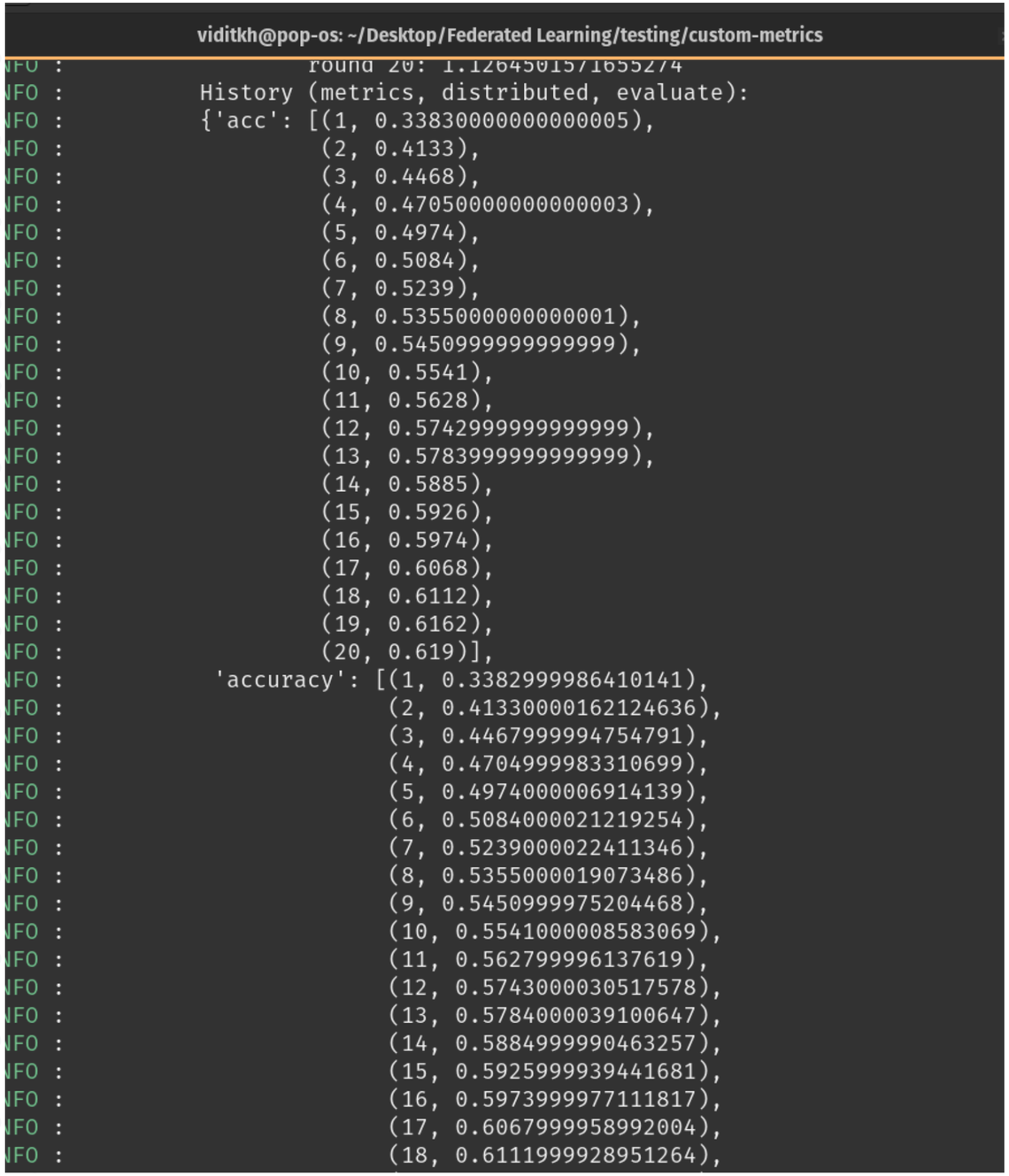
The **CLIENT** function performs the training(fitting) of the model

```python
# Define Flower client
class FlowerClient(NumPyClient):
    # pylint: disable=too-many-arguments
    def __init__(self, model, x_train, y_train, x_test, y_test):
        self.model = model
        self.x_train = x_train
        self.y_train = y_train
        self.x_test = x_test
        self.y_test = y_test

    def fit(self, parameters, config):
        self.model.set_weights(parameters)
        self.model.fit(
            self.x_train, self.y_train, epochs=1, batch_size=32, verbose=False
        )
        return self.model.get_weights(), len(self.x_train), {}

    def evaluate(self, parameters, config):
        self.model.set_weights(parameters)
        loss, accuracy = self.model.evaluate(self.x_test, self.y_test, verbose=False)
        y_pred = self.model.predict(self.x_test, verbose=False)
        y_pred = np.argmax(y_pred, axis=1).reshape(
            -1, 1
        )  # MobileNetV2 outputs 10 possible classes, argmax returns just the most probable

        acc, rec, prec, f1 = eval_learning(self.y_test, y_pred)
        output_dict = {
            "accuracy": accuracy,  # accuracy from tensorflow model.evaluate
            "acc": acc,
            "rec": rec,
            "prec": prec,
            "f1": f1,
        }
        return loss, len(self.x_test), output_dict
```

Experiments & Results

Dataset: CIFAR10
Model: CNN
20 rounds of FL, 10 Clients

Accuracy: 62%
f1 score:0.619



2. Changing the image classification model used

Dataset: CIFAR10
Model: MobileNetV2
5 rounds of FL, 10 Clients

Accuracy: 30.1%
f1-score:0.3

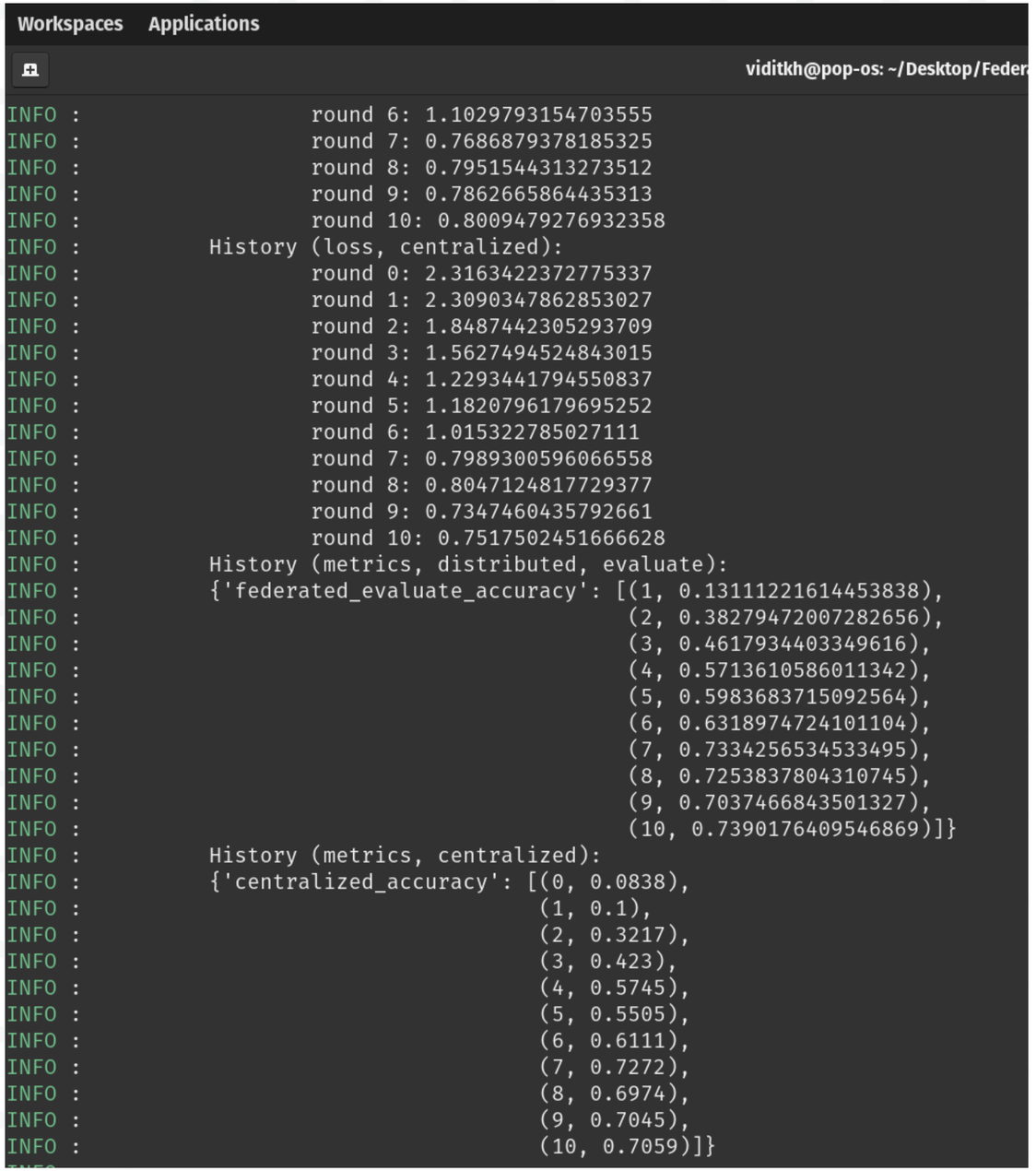
```
viditkh@pop-os: ~/Desktop/Federated Learning/testing/custom-metrics
INFO :
INFO :      [SUMMARY]
INFO :      Run finished 5 round(s) in 184.08s
INFO :          History (loss, distributed):
INFO :                  round 1: 2.1119752645492555
INFO :                  round 2: 2.015006446838379
INFO :                  round 3: 1.9740293979644776
INFO :                  round 4: 1.9505069851875305
INFO :                  round 5: 1.9351823449134826
INFO :          History (metrics, distributed, evaluate):
INFO :          {'acc': [(1, 0.26149999999999995),
INFO :                   (2, 0.27949999999999997),
INFO :                   (3, 0.28900000000000003),
INFO :                   (4, 0.2957),
INFO :                   (5, 0.3013)],
INFO :           'accuracy': [(1, 0.2615000009536743),
INFO :                        (2, 0.2794999986886978),
INFO :                        (3, 0.28899999856948855),
INFO :                        (4, 0.29570000171661376),
INFO :                        (5, 0.30129999816417696)],
INFO :           'f1': [(1, 0.26149999999999995),
INFO :                  (2, 0.27949999999999997),
INFO :                  (3, 0.28900000000000003),
INFO :                  (4, 0.2957),
INFO :                  (5, 0.3013)],
INFO :           'prec': [(1, 0.26149999999999995),
INFO :                    (2, 0.27949999999999997),
INFO :                    (3, 0.28900000000000003),
INFO :                    (4, 0.2957),
INFO :                    (5, 0.3013)],
INFO :           'rec': [(1, 0.26149999999999995),
INFO :                   (2, 0.27949999999999997),
INFO :                   (3, 0.28900000000000003),
INFO :                   (4, 0.2957),
INFO :                   (5, 0.3013)]}
INFO :
```

3. Changing the dataset used

Dataset: FashionMNIST
Model: CNN
10 rounds of FL, 50 Clients

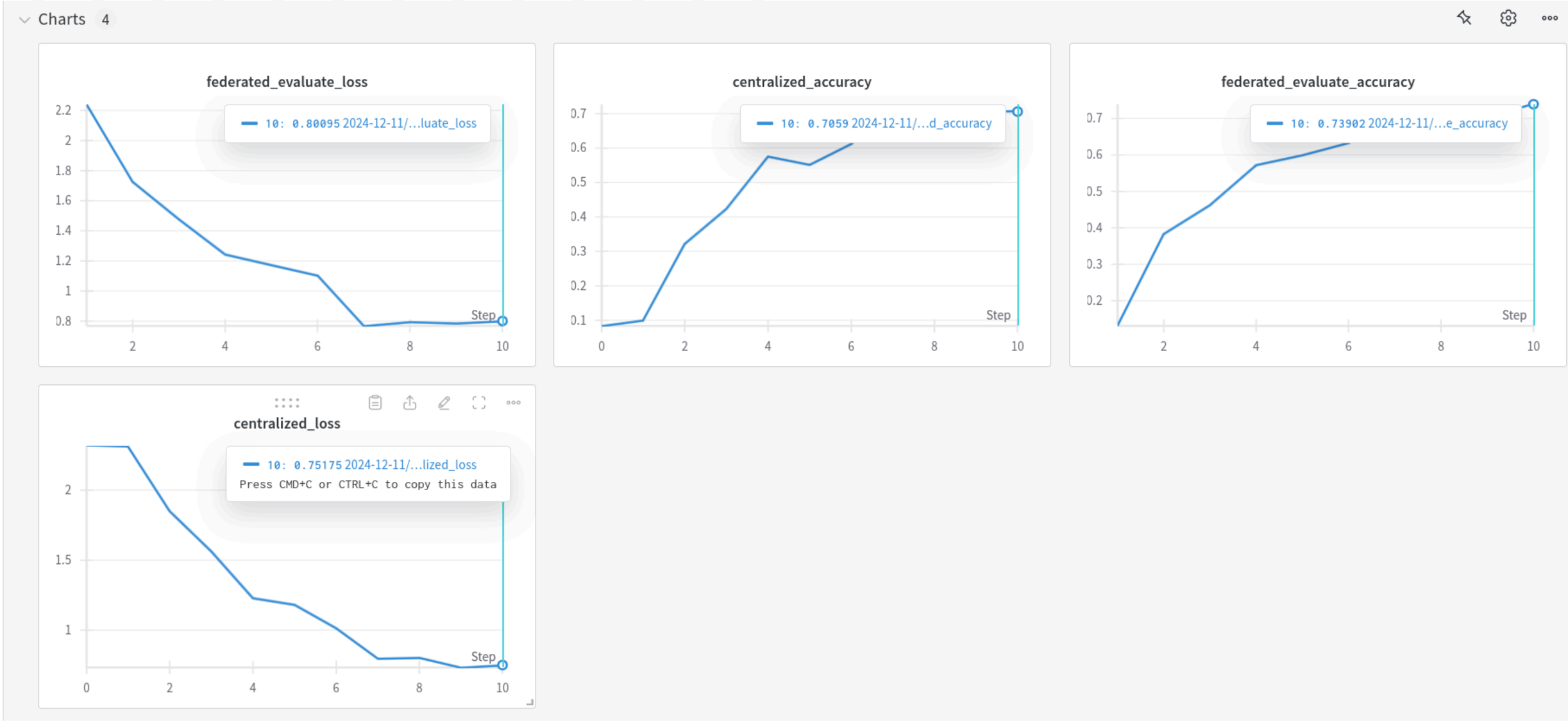
CL Accuracy: 70%

FL Accuracy: 73%



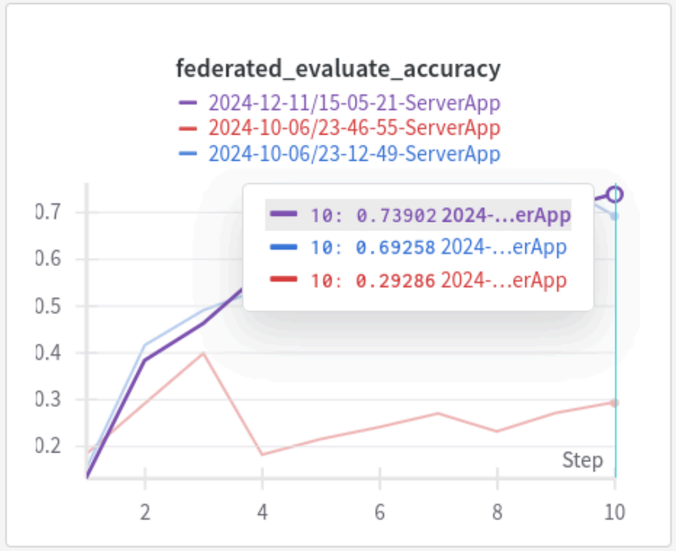
4. Comparing performance of FL v/s CL

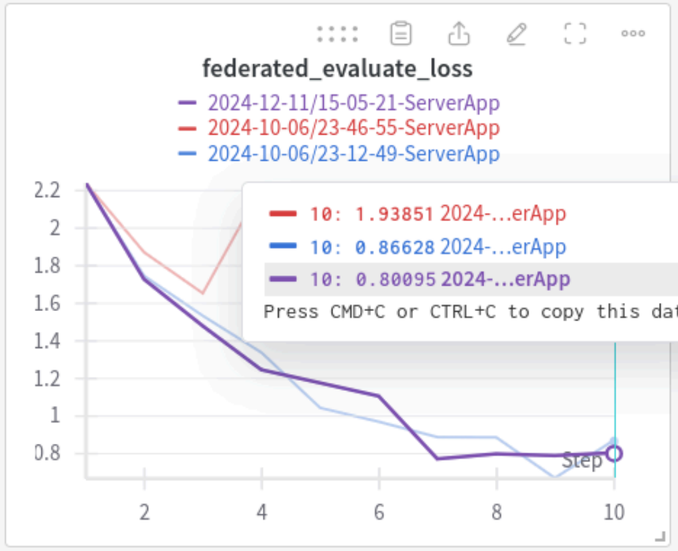
Graphs Plotted accuracy v/s Rounds

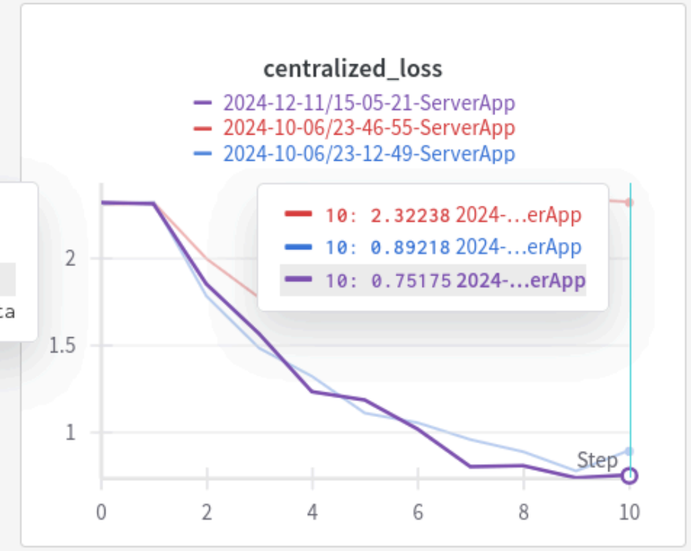


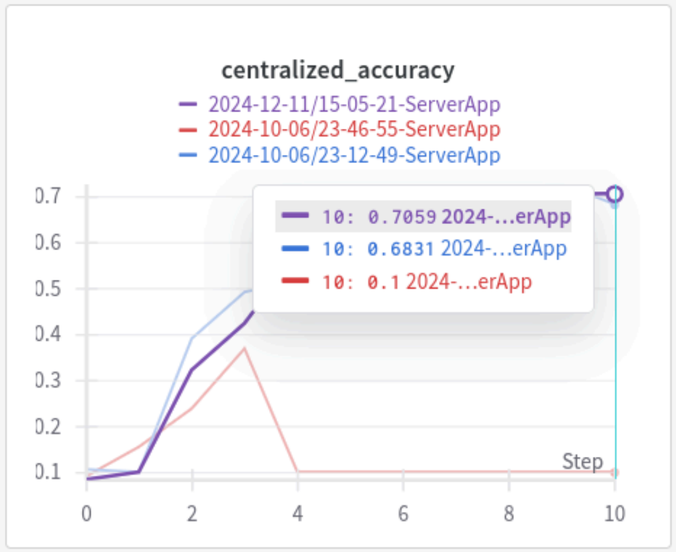
Graphs comparing FL v/s CL

Charts 4









System 18

Conclusion

- Using the Flower Framework for Federated Learing Simulation, implemented server and client simulating FL.
- Then achieved the above results in different scenarios like
- Changing the dataset
- Chaning the ML Model used
- Finally, compared performance of FL with Centralized Learning

- Now next step is to work on developing prototype for the simulations performed. Have learnt ways to connect devices wirelessly, will continue to work on this.

Larana University | 2025

Thank You

Presented by Aaron Loeb

Logistic Reg for MNIST Dataset

Accuracy: 83.9%

```
Workspaces  Applications
                                    viditkh@pop-os: ~/Desktop/Federated Learning/sklearn-logreg-mnist
INFO :      [ROUND 1]
INFO :      configure_fit: strategy sampled 5 clients (out of 10)
INFO :      aggregate_fit: received 5 results and 0 failures
WARNING :   No fit_metrics_aggregation_fn provided
INFO :      fit progress: (1, 1.3041110766418718, {'accuracy': 0.6769}, 8.102675664999879)
INFO :      configure_evaluate: strategy sampled 10 clients (out of 10)
INFO :      aggregate_evaluate: received 10 results and 0 failures
WARNING :   No evaluate_metrics_aggregation_fn provided
INFO :
INFO :      [ROUND 2]
INFO :      configure_fit: strategy sampled 5 clients (out of 10)
INFO :      aggregate_fit: received 5 results and 0 failures
INFO :      fit progress: (2, 0.5813381218367087, {'accuracy': 0.8197}, 17.217699679999896)
INFO :      configure_evaluate: strategy sampled 10 clients (out of 10)
INFO :      aggregate_evaluate: received 10 results and 0 failures
INFO :
INFO :      [ROUND 3]
INFO :      configure_fit: strategy sampled 5 clients (out of 10)
INFO :      aggregate_fit: received 5 results and 0 failures
INFO :      fit progress: (3, 0.5290841542400078, {'accuracy': 0.8397}, 20.32259933499995)
INFO :      configure_evaluate: strategy sampled 10 clients (out of 10)
INFO :      aggregate_evaluate: received 10 results and 0 failures
INFO :
INFO :      [SUMMARY]
INFO :      Run finished 3 round(s) in 22.34s
INFO :          History (loss, distributed):
INFO :                  round 1: 1.2923177606135794
INFO :                  round 2: 0.6141979962266947
INFO :                  round 3: 0.5612966216039441
INFO :          History (loss, centralized):
INFO :                  round 0: 2.3025850929940455
INFO :                  round 1: 1.3041110766418718
INFO :                  round 2: 0.5813381218367087
INFO :                  round 3: 0.5290841542400078
INFO :          History (metrics, centralized):
INFO :          {'accuracy': [(0, 0.098), (1, 0.6769), (2, 0.8197), (3, 0.8397)]}
INFO :
```