AIoTwin

Twinning action for spreading excellence in Artificial Intelligence of Things

Hands On: AIoTwin orchestration middleware

Ivan Čilić, Ana Petra Jukić, Katarina Vuknić (UniZg-FER, Croatia)

Grant agreement ID: 101079214

Agenda

- Federated Learning: Overview, Hands On Task and Challenges
- Hierarchical Federated Learning: Overview and Hands On Task
- Framework for Adaptive Orchestration of FL Pipelines
- Orchestrating HFL pipelines with AIoTwin Middleware Hands On

Introduction to Federated Learning

- Only model updates are communicated to a central server
- Sensitive data never leaves user devices
- Bandwidth savings by avoiding large-scale data uploads
- Applications:
  - manifacturing; equipment failures prediction,
  - healthcare; diagnostics,
  - transportation; traffic prediction

Flower Framework for FL

- Modular design: separate server and client roles
- Server coordinates training rounds and aggregates model updates
- Clients perform local training and evaluation
- Communication via RPC between server and clients
- Scalable from simulation to real-world deployments

server.py

```python
import torch
import torch.nn as nn
import flwr as fl
from flwr.common import ndarrays_to_parameters, parameters_to_ndarrays, Metrics
from flwr.server.strategy import FedAvg
import yaml
from typing import Tuple, Optional
from task import Net, get_weights, load_data, test, set_weights

class LogAccuracyStrategy(FedAvg):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        _, self.testloader = load_data(
            partition_id=0,
            num_partitions=1,
            batch_size=32,
        )
        self.net = Net()
        self.device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")

    def evaluate(self, rnd: int, parameters,) -> Optional[Tuple[float, Metrics]]:
        ...

if __name__ == "__main__":

    # Initialize model parameters
    ndarrays = get_weights(Net())
    parameters = ndarrays_to_parameters(ndarrays)

    strategy = LogAccuracyStrategy(
        fraction_fit=1,
        fraction_evaluate=1,
        min_fit_clients=2,
        min_evaluate_clients=2,
        min_available_clients=2,
        initial_parameters=parameters,
    )

    # Start Flower server
    fl.server.start_server(
        server_address="10.19.4.113:8080",
        config=fl.server.ServerConfig(num_rounds=15),
        strategy=strategy,
    )
```

client.py

```python
class FlowerClient(fl.client.NumPyClient):
    def __init__(self, trainloader, valloader, epochs, learning_rate, parti-
tion_id):
        self.net = Net()
        self.trainloader = trainloader
        self.valloader = valloader
        self.epochs = epochs
        self.lr = learning_rate
        self.partition_id = partition_id
        self.device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")

    def fit(self, parameters, config):
            """Train the model with data of this client."""
      ...
    def evaluate(self, parameters, config):
        """Evaluate the model on the data this client has."""
      ...
# --------------------------
# Main
# --------------------------
if __name__ == "__main__":

    partition_id = 0
    num_partitions = 10
    batch_size = 32
    epochs = 3
    learning_rate = 0.1
    server_address = "10.19.4.113:8080"
    trainloader, valloader = load_data(partition_id, num_partitions, batch_size)


    client = FlowerClient(trainloader, valloader, epochs, learning_rate, parti-
tion_id).to_client()

    fl.client.start_numpy_client(server_address=server_address, client=client)
```

task.py

```python
class Net(nn.Module):
    def __init__(self):
  ...
    def forward(self, x):
  ...

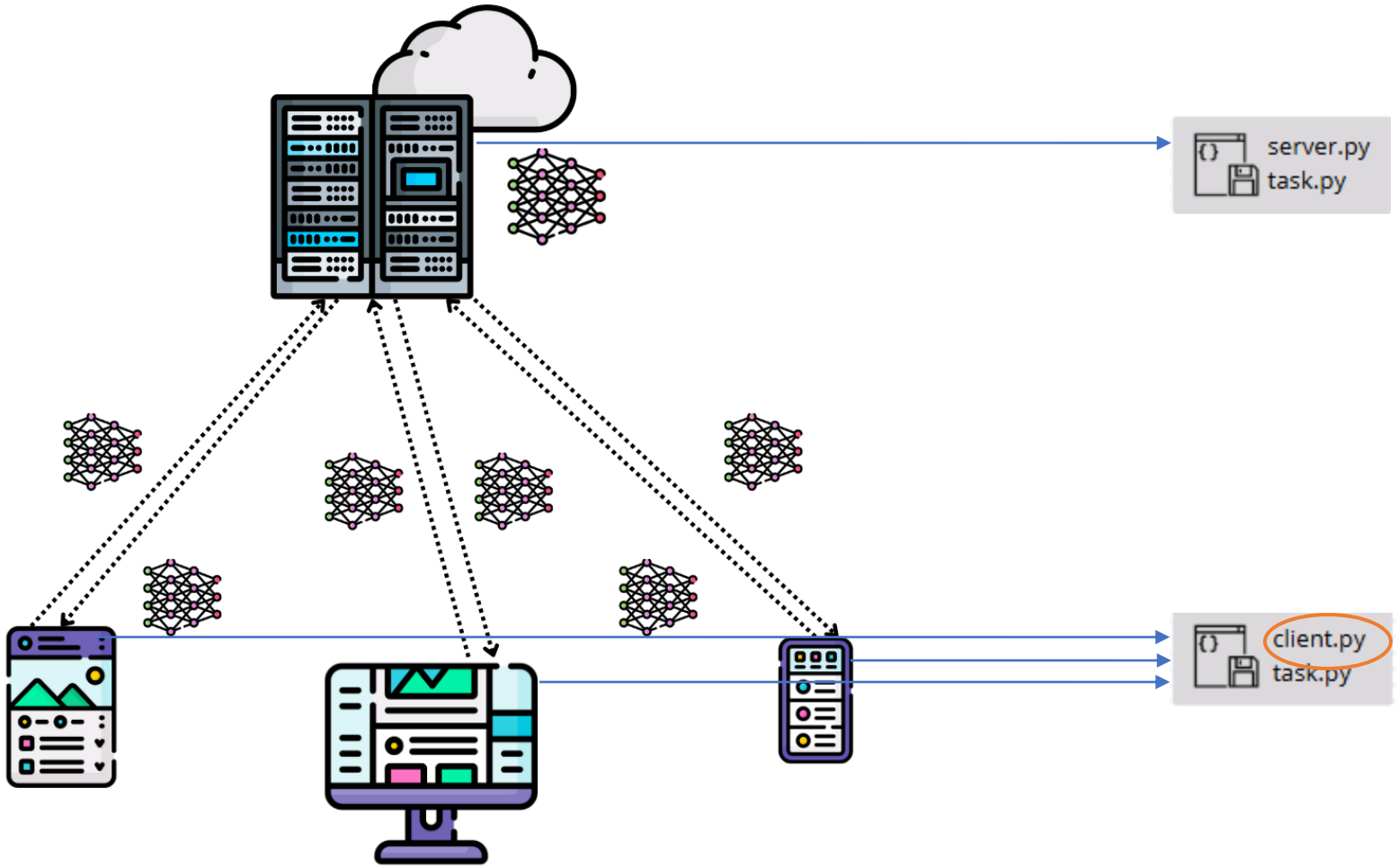
def get_weights(net: nn.Module) -> List[np.ndarray]:
    """Extract model weights as a list of NumPy arrays."""

def set_weights(net: nn.Module, parameters:  List[np.ndarray]) -> None:
    """Set model weights from a list of NumPy arrays."""

def load_data(partition_id: int, num_partitions: int, batch_size: int) -> Tuple[DataLoader, DataLoader]:
    """Load data."""
    # Return trainloader, testloader

def train (
    net: nn.Module,
    trainloader: DataLoader,
    valloader: DataLoader,
    epochs: int,
    learning_rate: float,
    device: torch.device
    ) -> Dict[str, float]:
    """Train the model on the training set."""
    # Return validation results

def test(
    net: nn.Module,
    testloader: DataLoader,
    device: torch.device
    ) -> Tuple[float, float]:
    """Evaluate the model on the test set."""
```


server.py
task.py
client.py
task.py

AIoTwin

Hands On Part I.
FL with Flower Framework

1. Go to: https://github.com/**AIoTwin/fl-service/tree/tutorial**
2. See **README**

Getting Started

- **Connect to the jump server**:

  ssh –p 22 iotlab@161.53.19.19

  password: SummerSchool!2025

- **Inside the jump server, connect to your VM**:

  ssh hfl-nX

  where X is number between 4-13

Task I. Centralized FL with Flower

- CIFAR10, IID
- ```
  ├── task1
  │   └── client.py
  │   └── task.py
  ```

- **Flower Server**: (hfl-n1, **"10.19.4.113:8080"**)

- **Flower Clients**: (hfl-n4) (hfl-n5) (hfl-n6) (hfl-n7) (hfl-n8) (hfl-n9) (hfl-n10) (hfl-n11) (hfl-n12) (hfl-n13)
  ```

Federated Learning challenges

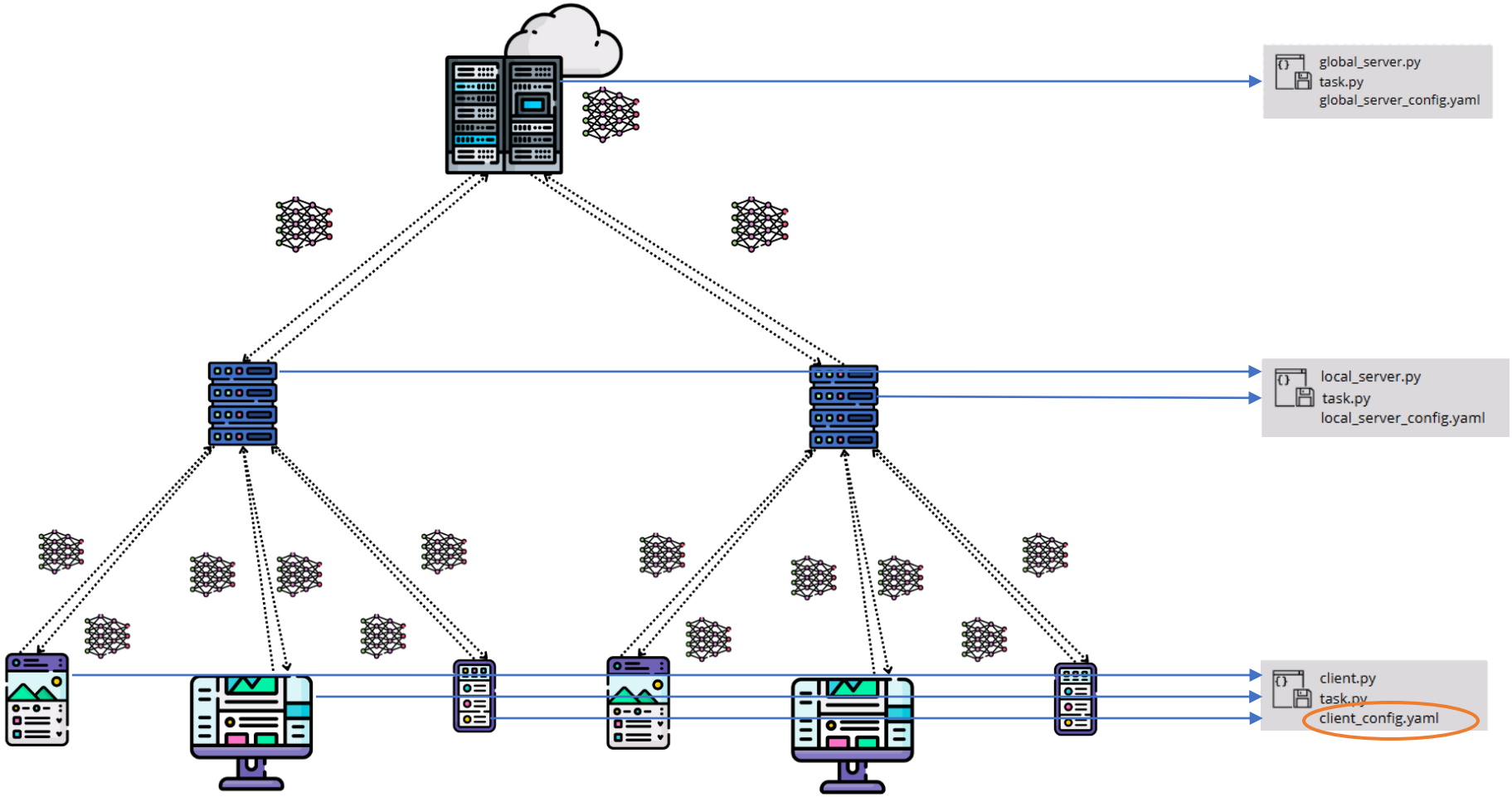
- Central server can struggle with many clients
- High bandwidth usage between server and edge devices
- Diverse data distributions degrade global model performance
- Not all clients are always reachable or stable
- Devices differ in resources and connectivity

Hierarchical Federated Learning

- Multi-level aggregation reduces central server load
- Clients communicate with nearby aggregators, not the central server
- Clustering clients allows better local adaptation
- Lightweight local servers adapt to clients' capabilities

Hierarchical Federated Learning

HFL with Flower Framework



Add On: Configuration Files

```yaml
server:
  address: "0.0.0.0:8080"
  global_rounds: 2

strategy:
  fraction_fit: 1.0
  fraction_evaluate: 1.0
  min_fit_clients: 2
  min_evaluate_clients: 2
  min_available_clients: 2
```

global_server_config.yaml

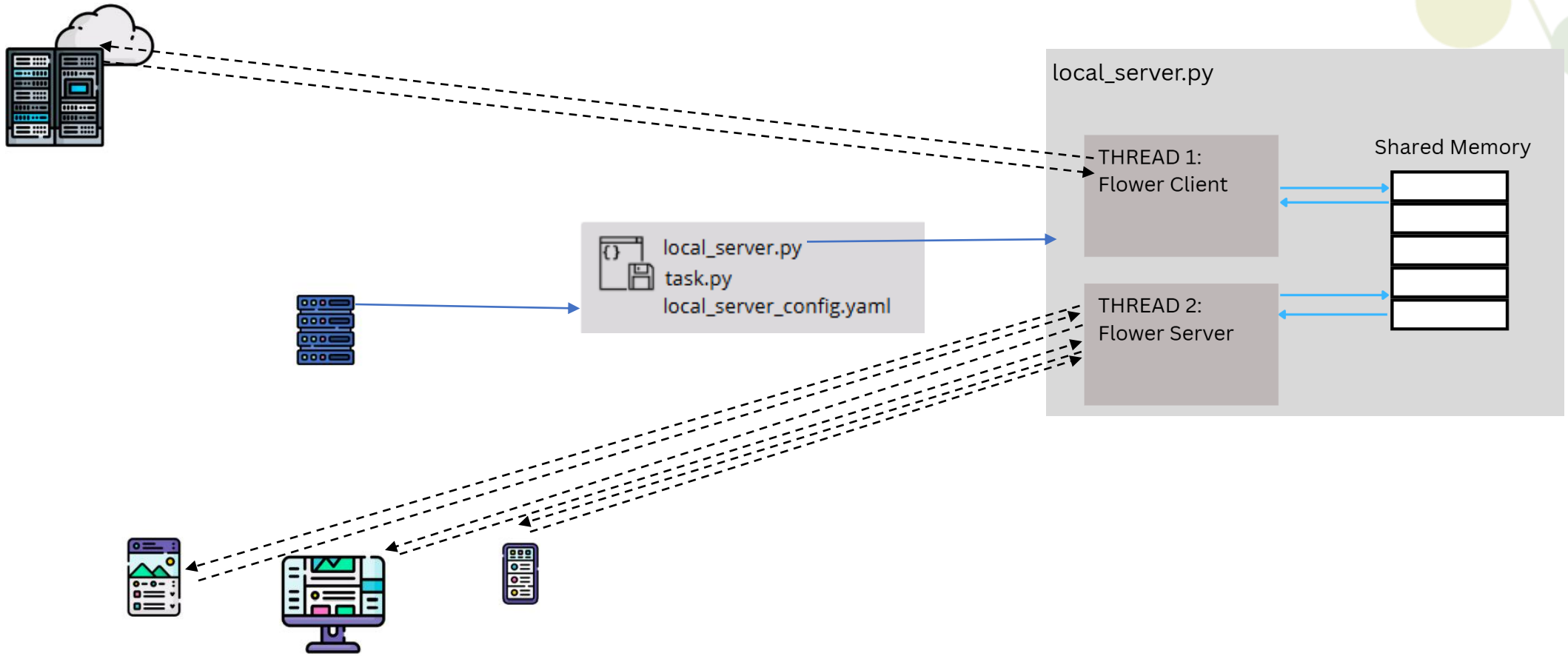
```yaml
server:
  address: "10.19.0.168:8080"

node_config:
  partition-id: 0
  num-partitions: 4

run_config:
  batch-size: 32
  local-epochs: 1
  learning-rate: 0.1
```

client_config.yaml

Add On: Local Aggregator Implementation



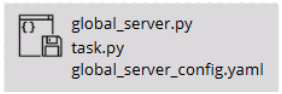
AIoTwin

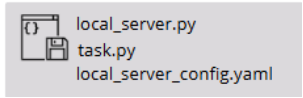
Hands On Part I. Hierarchical FL with Flower Framework

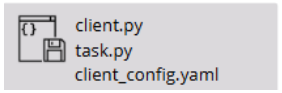
1. Go to: https://github.com/**AIoTwin/fl-service/tree/tutorial**
2. See **README**

Task II. HFL Setup

- CIFAR10, IID
- ```
  ├── task2
  │   ├── client.py
  │   ├── client_config.yaml
  │   └── task.py
  ```







- **Flower Global Server**: (hfl-n1, **"10.19.4.113:8080"**)
- **Flower Local Server**: (hfl-n2, 10.19.4.119:8080) (hfl-n3, 10.19.4.201:8080)
- **Flower Clients**: (hfl-n4) (hfl-n5) (hfl-n6) (hfl-n7) (hfl-n8) (hfl-n9) (hfl-n10) (hfl-n11) (hfl-n12) (hfl-n13)
  ```

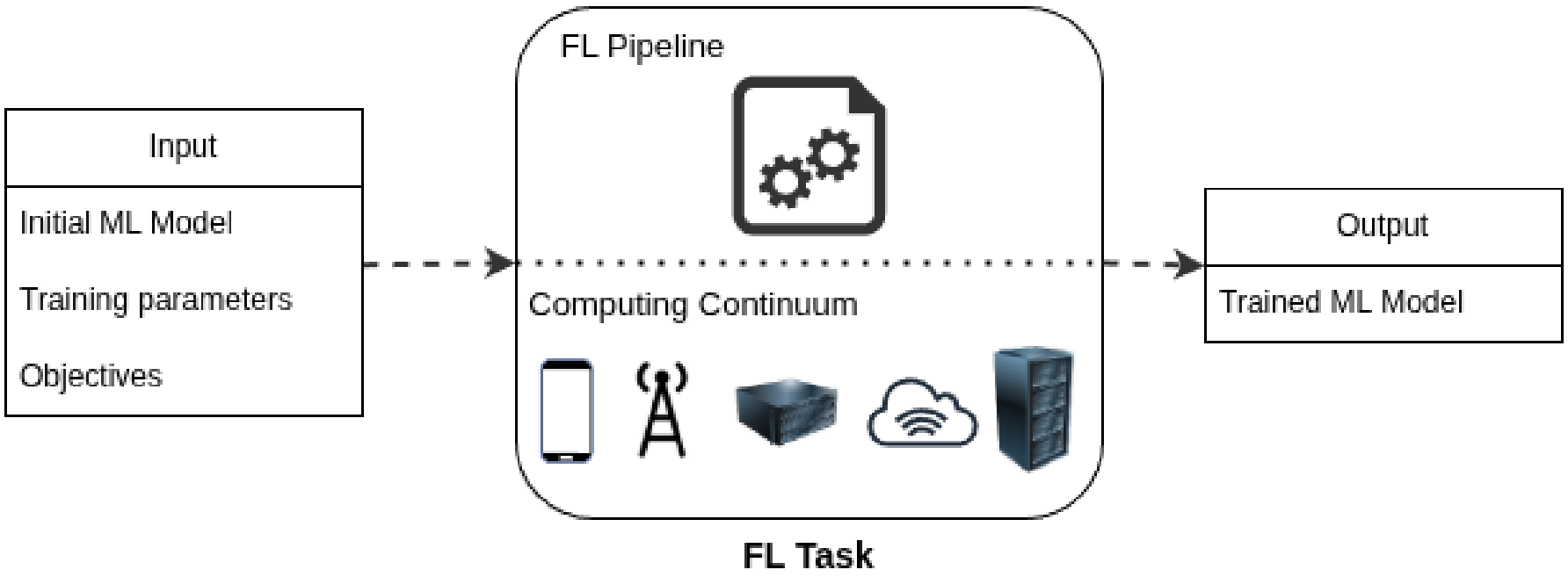
AIoTwin

Framework for Adaptive Orchestration of FL Pipelines

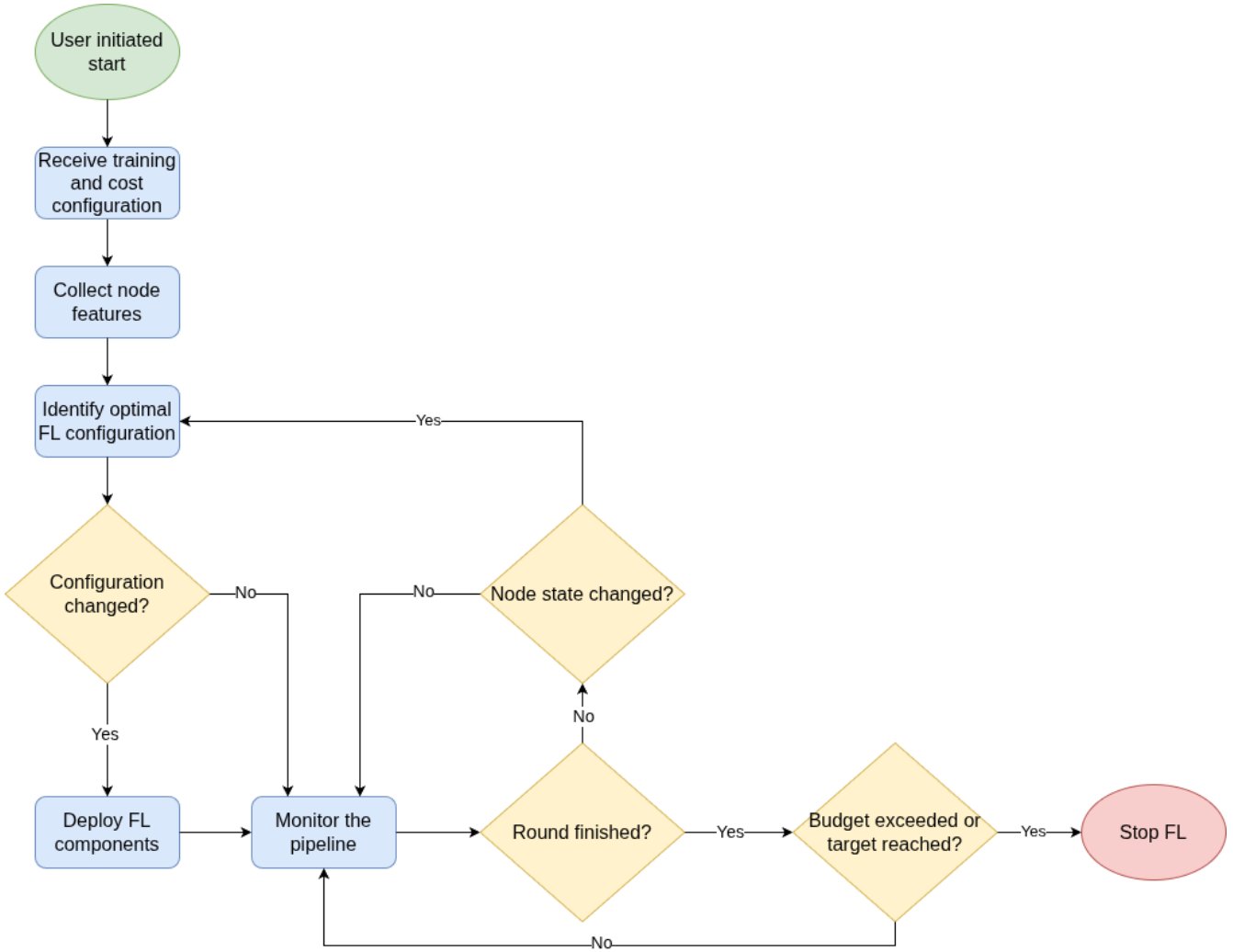
Framework for Adaptive Orchestration of FL Pipelines

- Main goal: automate deployment of FL services (clients, aggregators) and handle infrastructure changes that affect the FL performance
- Input:
  - ML task
    - Initial ML model
    - Training parameters
  - Objectives
    - Budget or cost minimization
    - Parameters
      - Communication
      - Energy

FL Task Definition

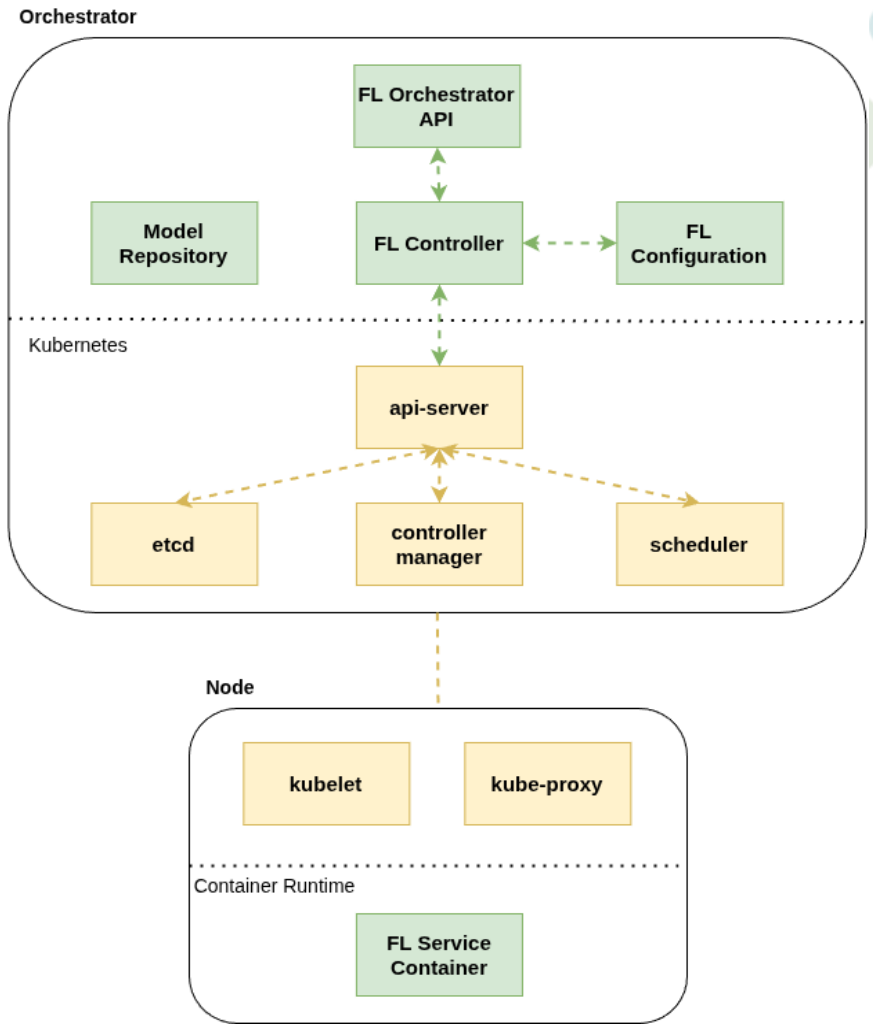


Orchestration Workflow



Framework implementation

- Kubernetes for orchestration
- FL controller
  - Main component that controls the pipeline
- Configuration strategy
  - Identifies best-fit configuration
  - Min. comm. cost, min. KLD, min. energy
- API for end users
- FL Service as a Flower wrapper (described previously)
- FL task defined in an external file
- Topology
  - Real – one FL Service per node
  - Simulated – simulated nodes and communication costs



AIoTwin

Hands On Part II. Orchestrating HFL pipelines with AIoTwin Midleware

Commands and instructions link:

https://github.com/AIoTwin/fl-orchestrator/blob/aiotwin-tutorial/tutorial/README.md

Tasks

1. Start your orchestrator
2. Check the topology and the FL task
3. Start a pipeline with the default setup and monitor the progress
4. Change the default setup and monitor progress
   - Change HFL parameters (epochs, local rounds)
   - Change the client partitioning
   - Change the dataset (optional)

```
$ ssh iotlab@161.53.19.19 -p 22
```

Password: **SummerSchool!2025**

Simulated topology