Anomaly detection in smart buildings using federated learning

Tuhin Sharma | Binaize Labs

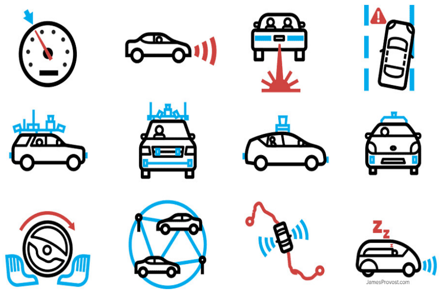
Bargava Subramanian | Binaize Labs

Outline

- What is Smart Building?
- Anomalies in Smart Building.
- Challenges in IoT.
- Federated Learning.
- Anomaly detection using Federated Learning
- Demo
- Types of Federated Learning.
- Pros and Cons.

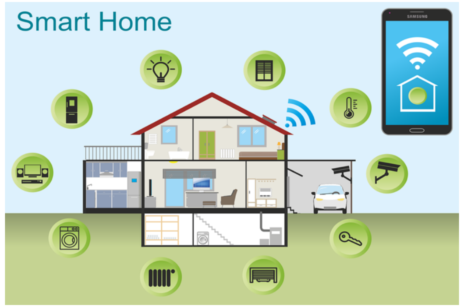
We are increasingly moving towards a smart inter-connected world

- Wearables
- Self-driving cars
- Healthcare
- Drone
- Smart Retail Store.
- Industrial IoT
- Smart Farm
- Smart Home and Building
- Smart City







10B+ IoT devices!!

What is Smart Building?

Smart buildings not only take complete care of tenants’ comfort and safety but also promote energy and financial savings. Now, AI also contributes to making buildings smarter and more intelligent than ever.

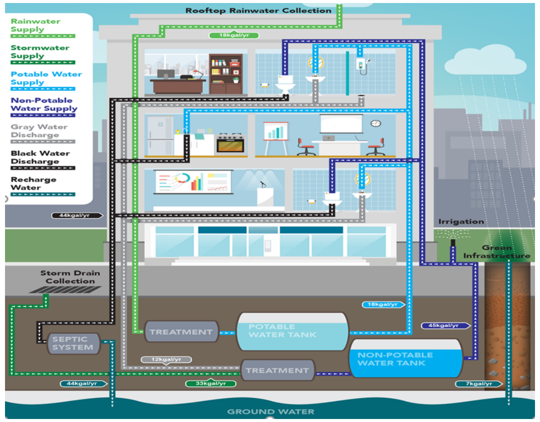
- Forbes 2019

ARTIFICIAL INTELLIGENCE

SMART BUILDING

SMARTER BUILDING

How AI is helping buildings become smarter

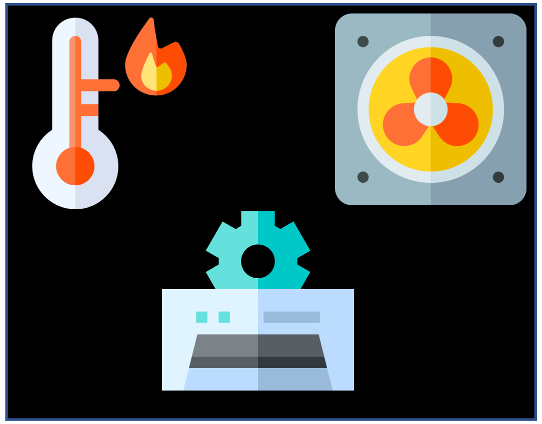


WATER MANAGEMENT

BUILDING MAINTENANCE

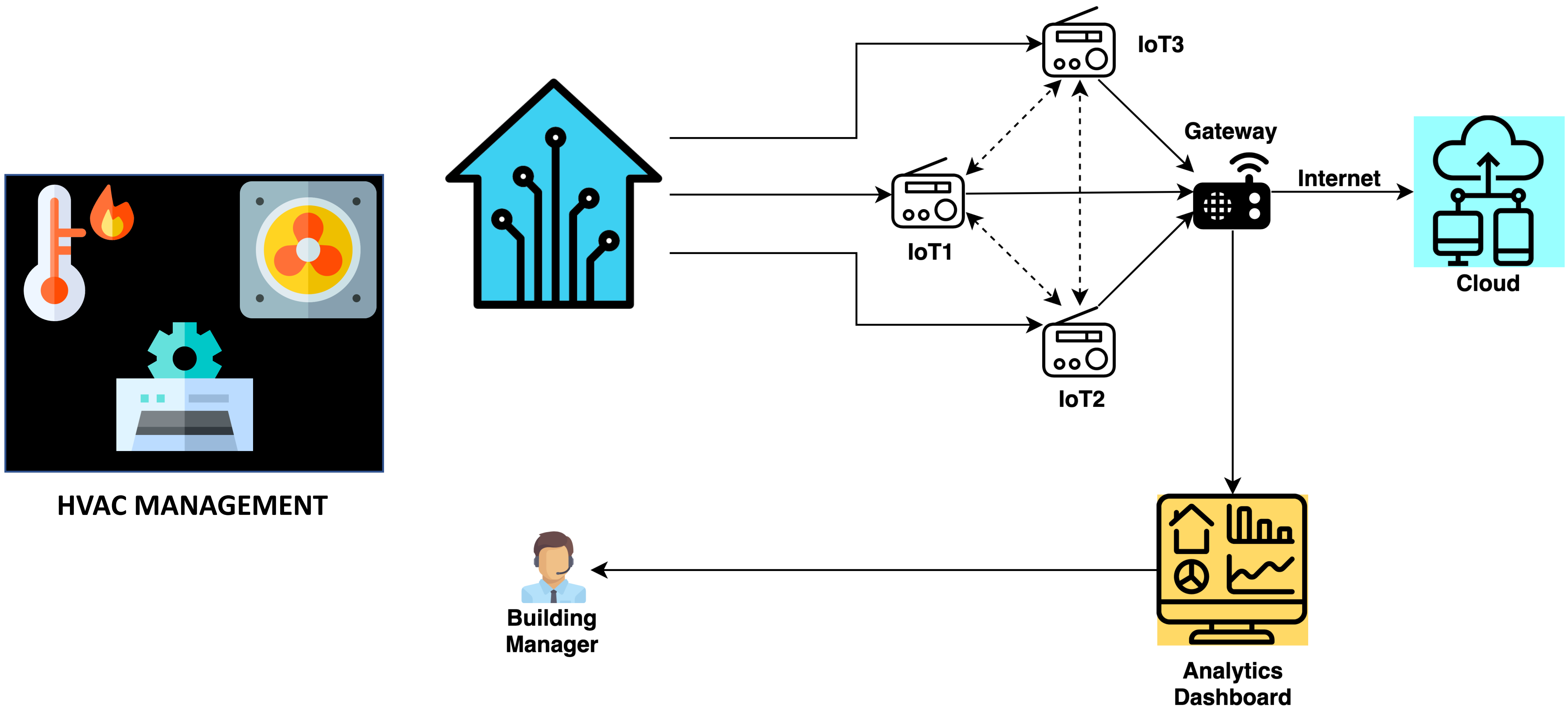
PARKING ASSISTANCE

SMART BULBS MANAGEMENT

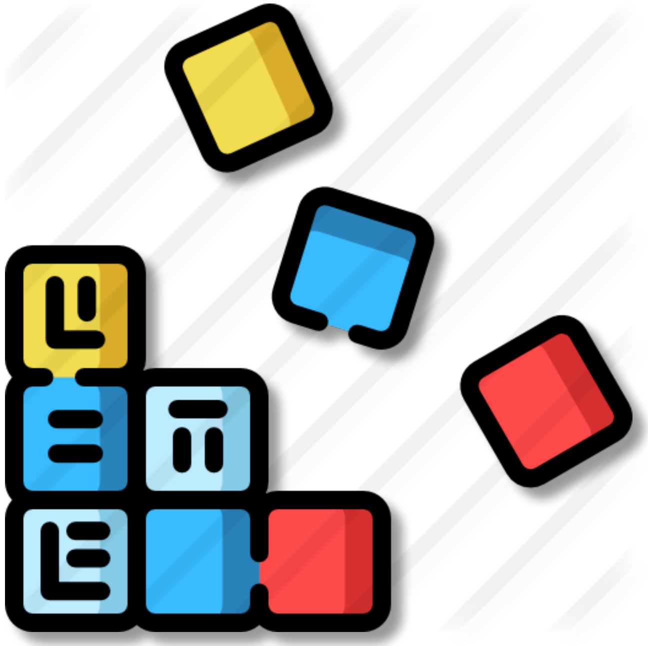


HVAC MANAGEMENT

Smart HVAC Management



Challenges in Smart Building



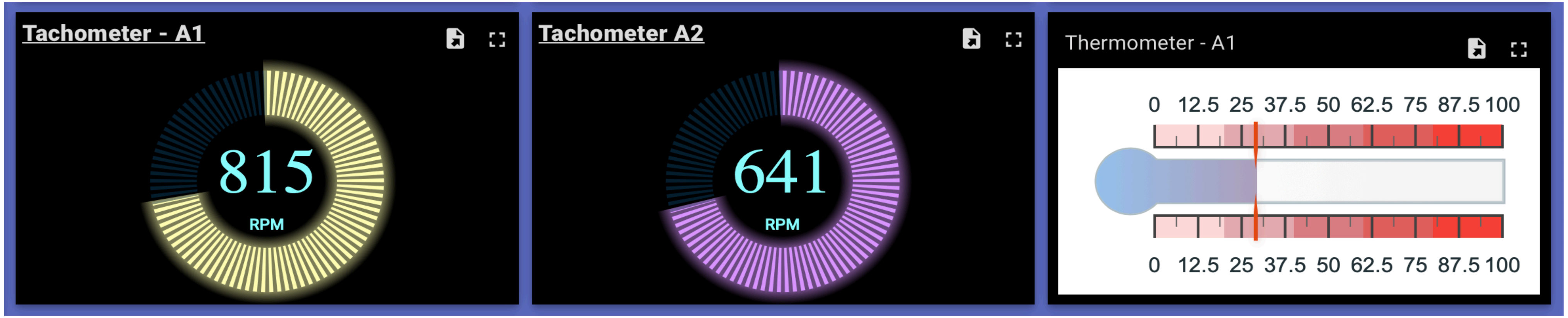
DATA CORRUPTION

CYBER BREACH

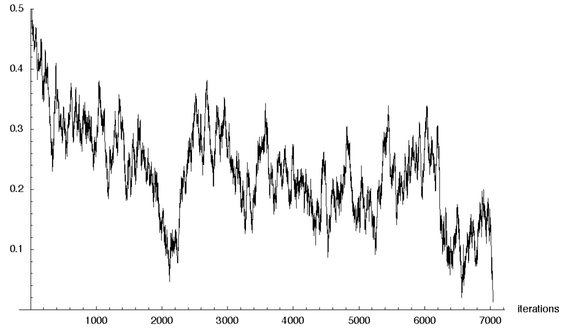
Anomaly detection is critical

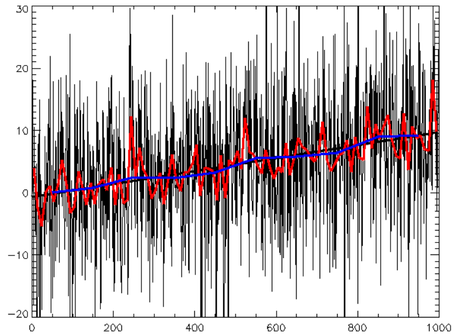
The core is a stream of time series events and the goal is to find anomalies in them

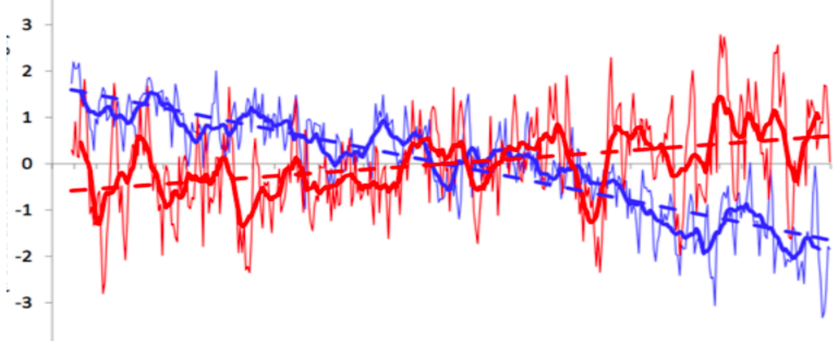
SENSORS' APPLICATION LEVEL DATA



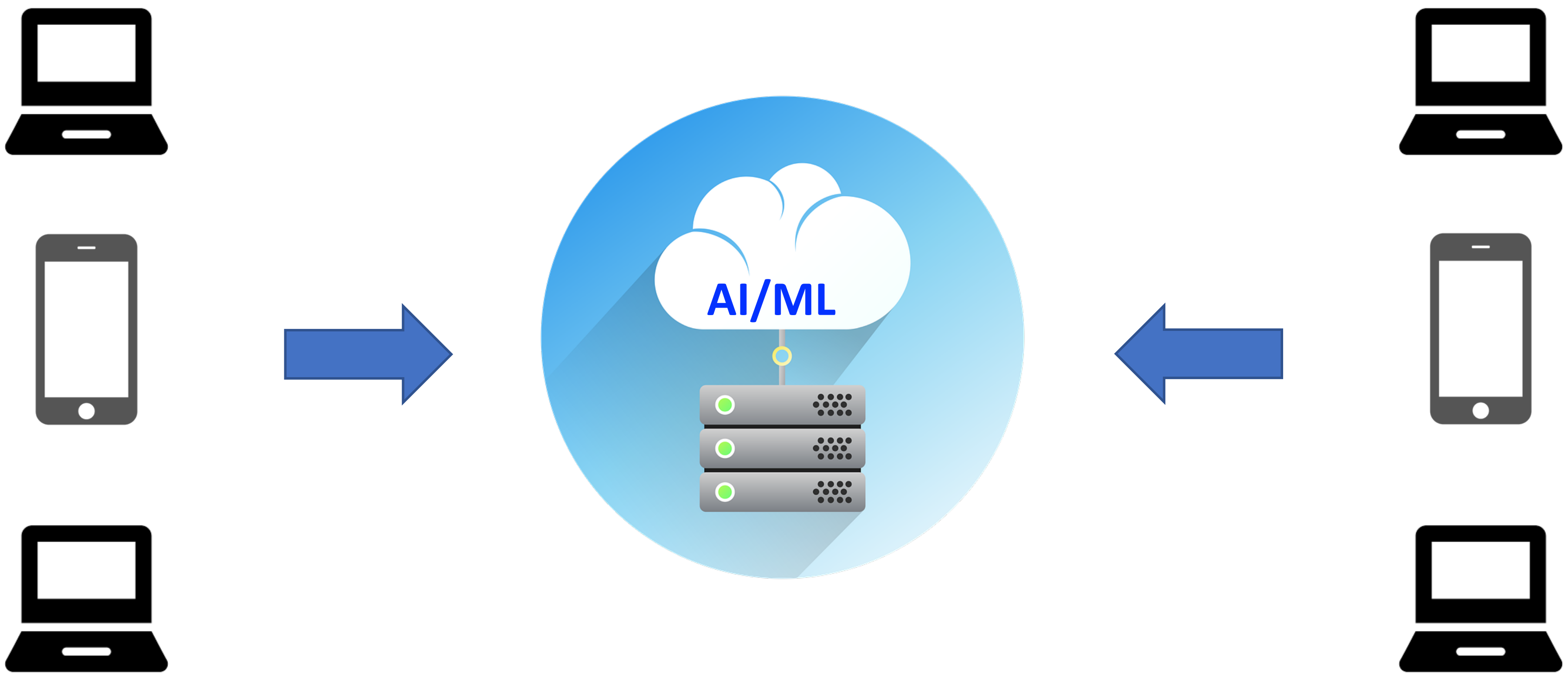
SENSORS' NETWORK LEVEL DATA







The current standard practice is to build machine learning models on Centralized data



But connected devices present a number of novel challenges

INTERMITTENT
INTERNET CONNECTION

But connected devices present a number of novel challenges

INTERMITTENT
INTERNET CONNECTION

HIGH DATA VOLUME AND
VELOCITY

But connected devices present a number of novel challenges

INTERMITTENT
INTERNET CONNECTION

LIMITED BATTERY

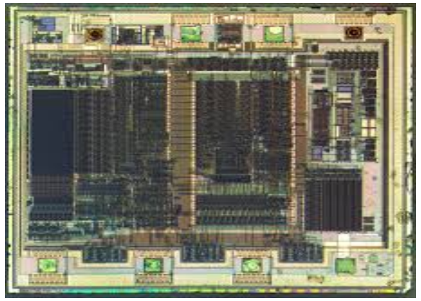
HIGH DATA VOLUME AND
VELOCITY

But connected devices present a number of novel challenges

INTERMITTENT
INTERNET CONNECTION

LIMITED BATTERY

HIGH DATA VOLUME AND
VELOCITY



LIMITED MEMORY AND
PROCESSING POWER

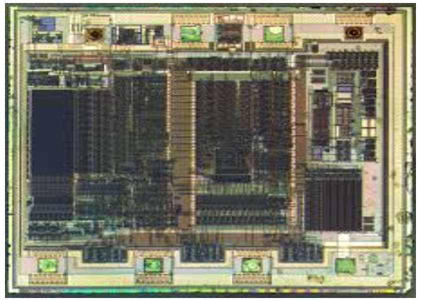
But connected devices present a number of novel challenges

INTERMITTENT
INTERNET CONNECTION

HIGH DATA VOLUME AND
VELOCITY

DATA PRIVACY

LIMITED BATTERY



LIMITED MEMORY AND
PROCESSING POWER

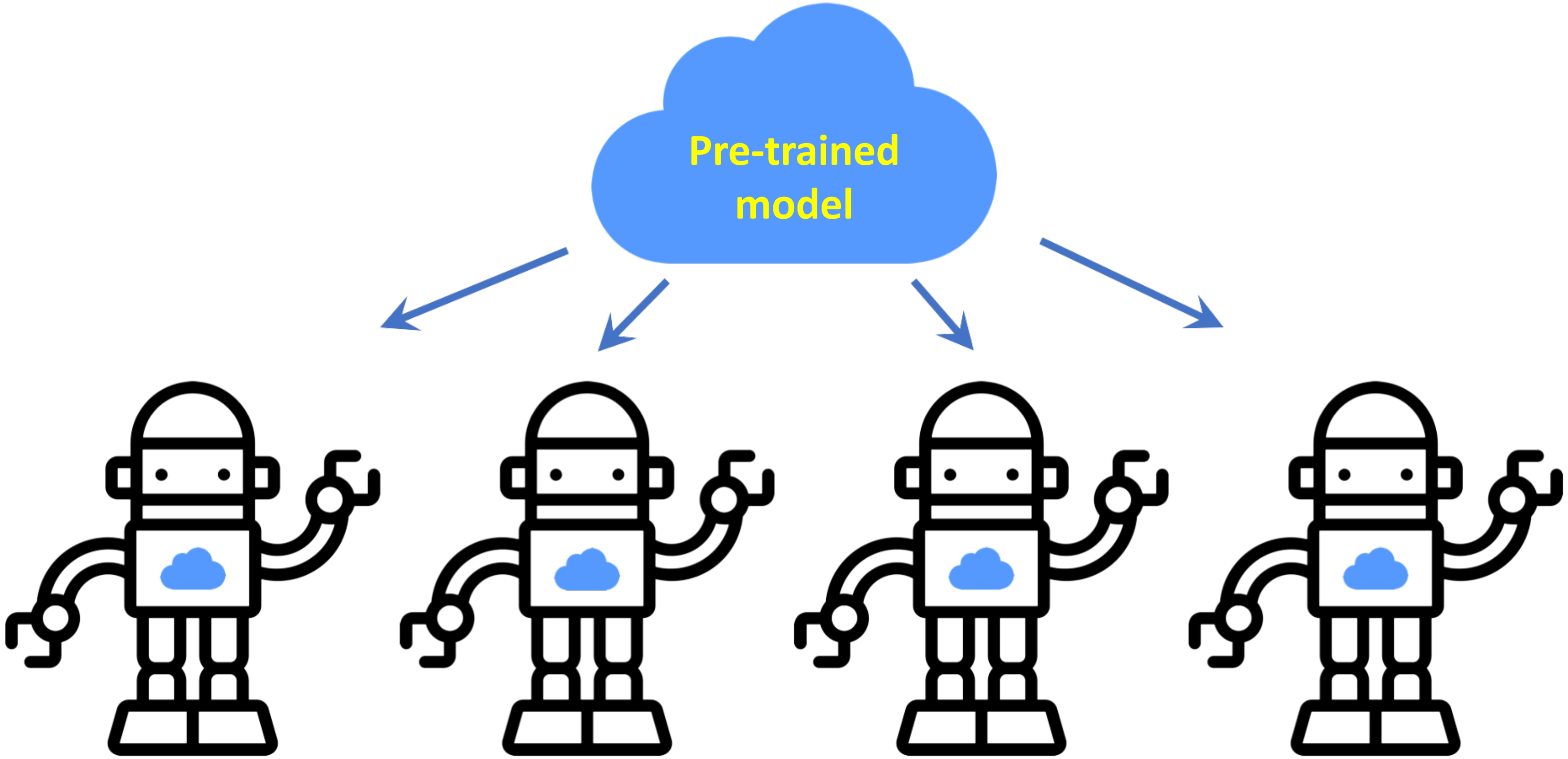
Federated Learning is here to rescue!!

- Decentralized learning
- Secure computing
- Preserve privacy

Federated Learning is here to rescue!!

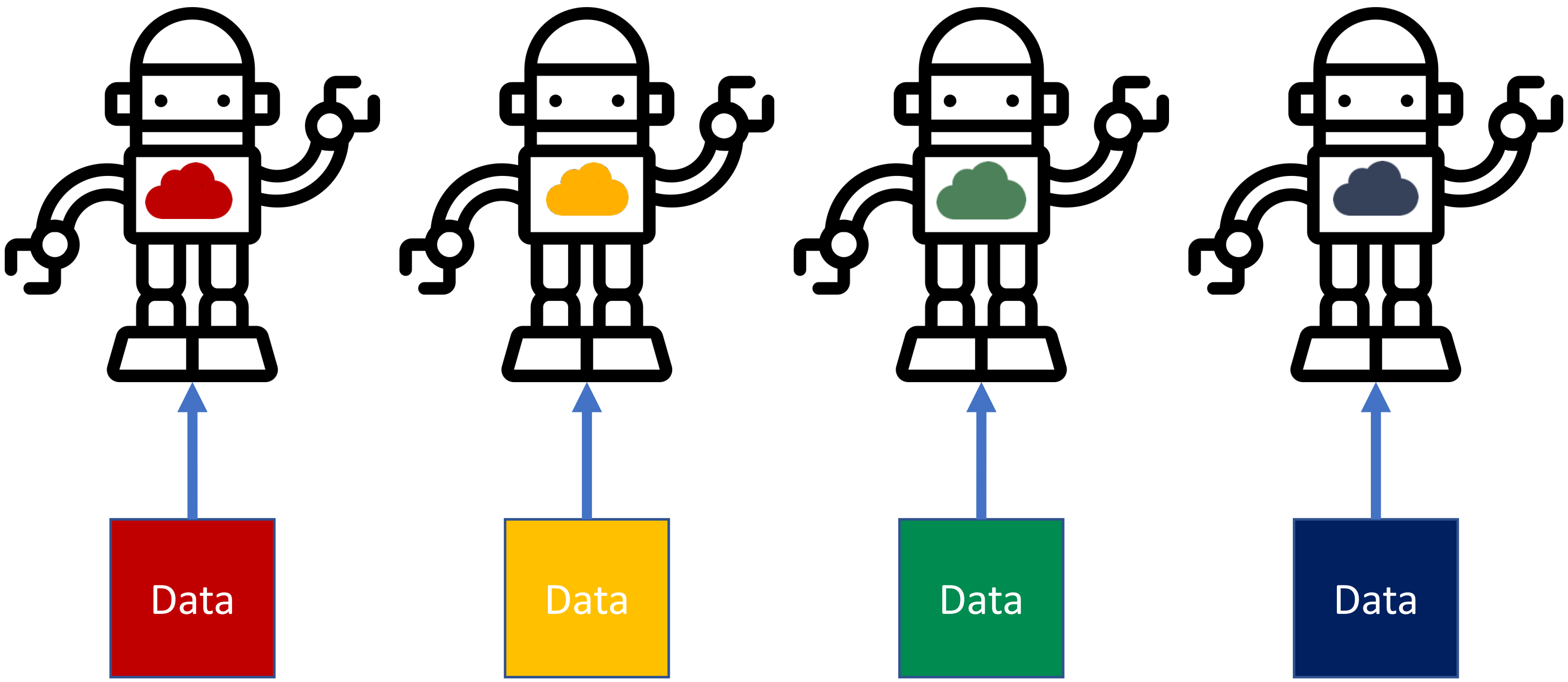
- Federation Construction.
- Decentralized Training.
- Model Accumulation.
- Model Aggregation (FedAvg).

(a) Federation Construction



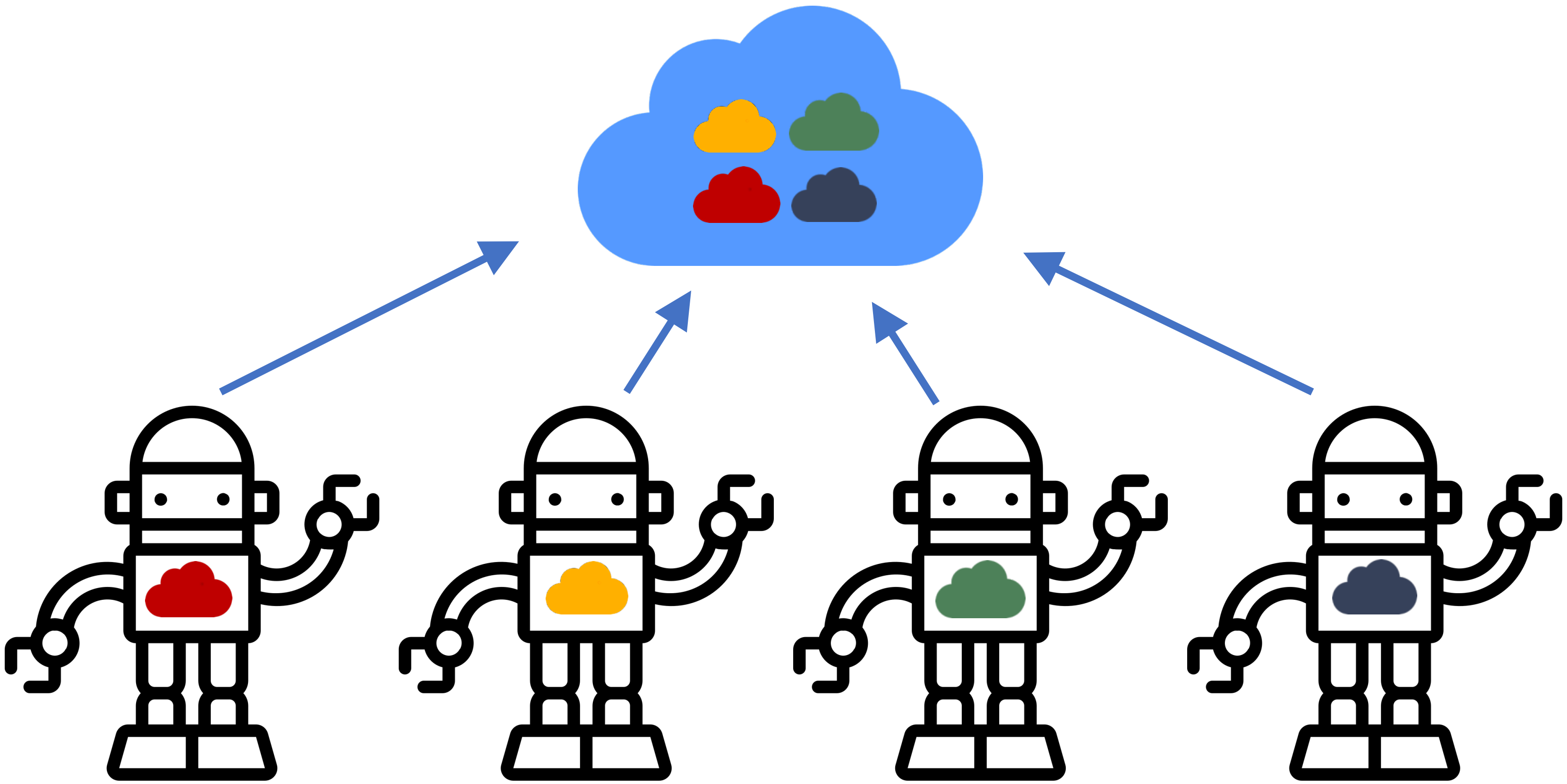
A random subset of members of the devices is selected to receive the global model synchronously from the server.

(b) Decentralized Training



Each selected device computes an updated model using its local data.

(c) Model Accumulation

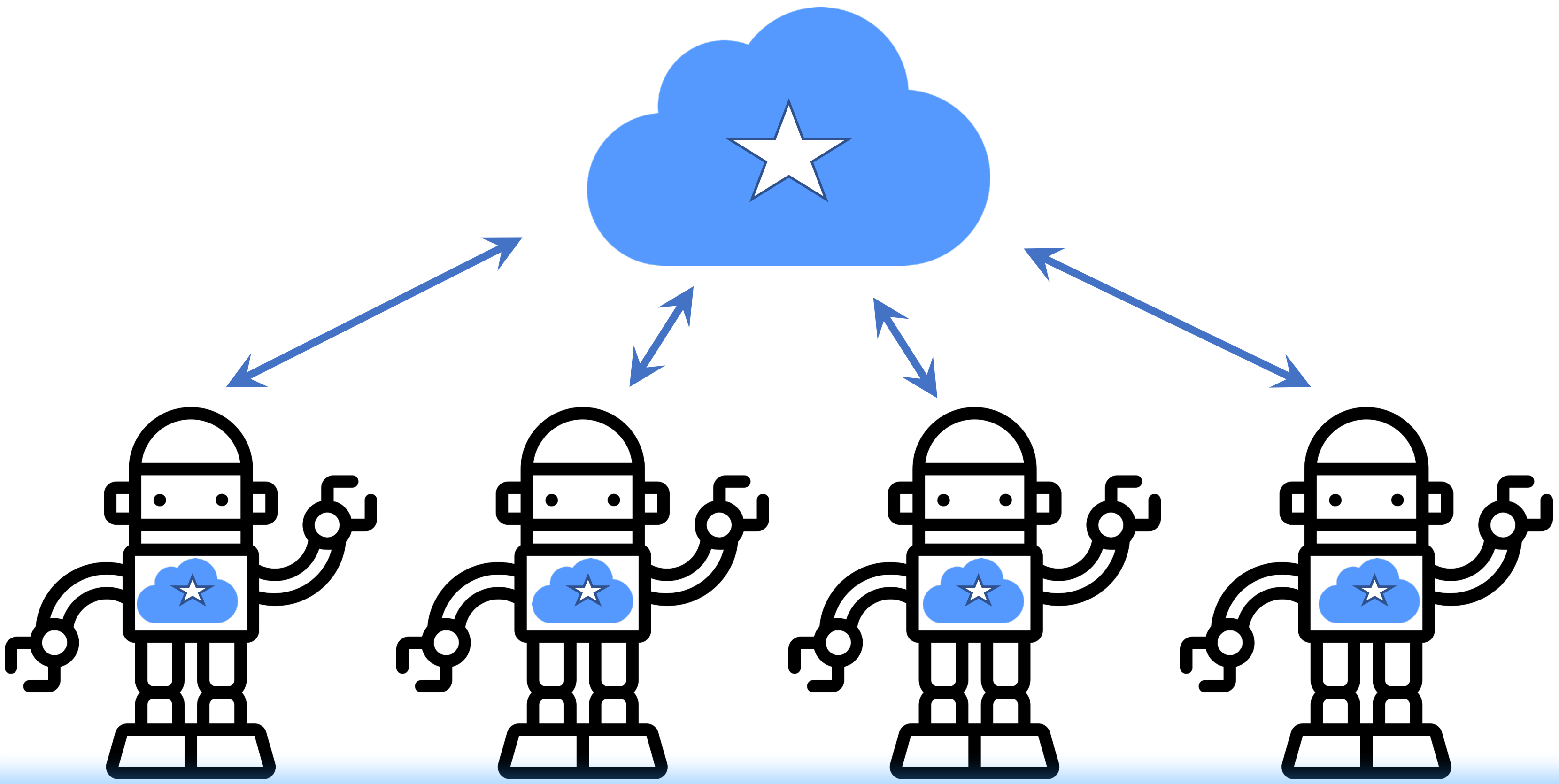


Only the model updates are sent from the federation to the server. Data is not moved.

(d) Model Aggregation

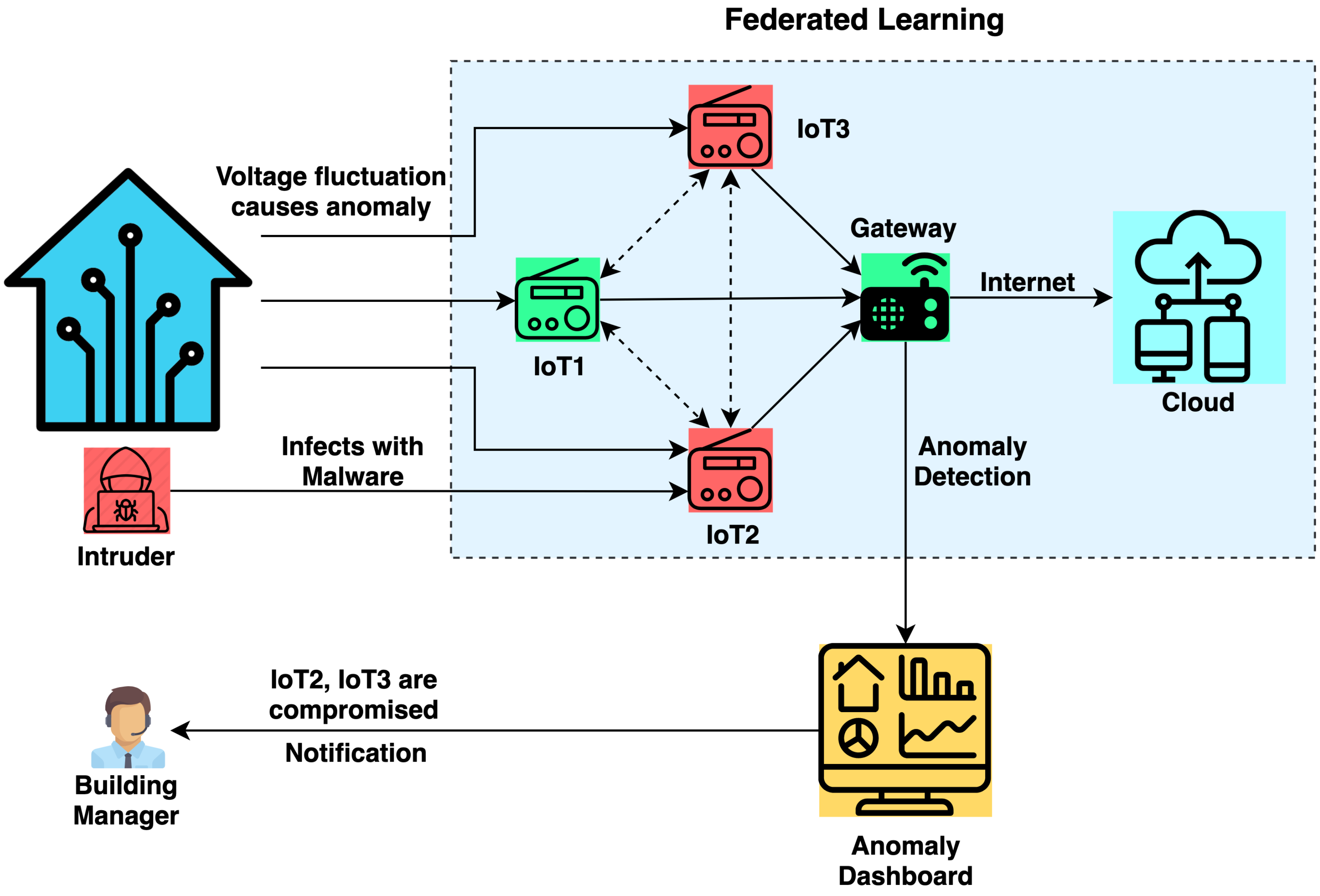
The server aggregates these model weights (typically by averaging) to construct an improved global model.

Federated Learning (Rinse, Repeat)



The devices receive the updated model.

Use Case



Federated Learning
IoT3
Voltage fluctuation causes anomaly
Gateway
Internet
IoT1
Cloud
Infects with Malware
Anomaly Detection
Intruder
IoT2
IoT2, IoT3 are compromised
Notification
Building Manager
Anomaly Dashboard

Tools- Choices

Google

OpenMined

PySyft

WeBank 微众银行

FATE

We use PySyft

Our journey

Rules + Z-score

K-Means + Isolation Forest + Oneclass SVM

Deep Auto-Encoder

Federated Learning

Unsupervised

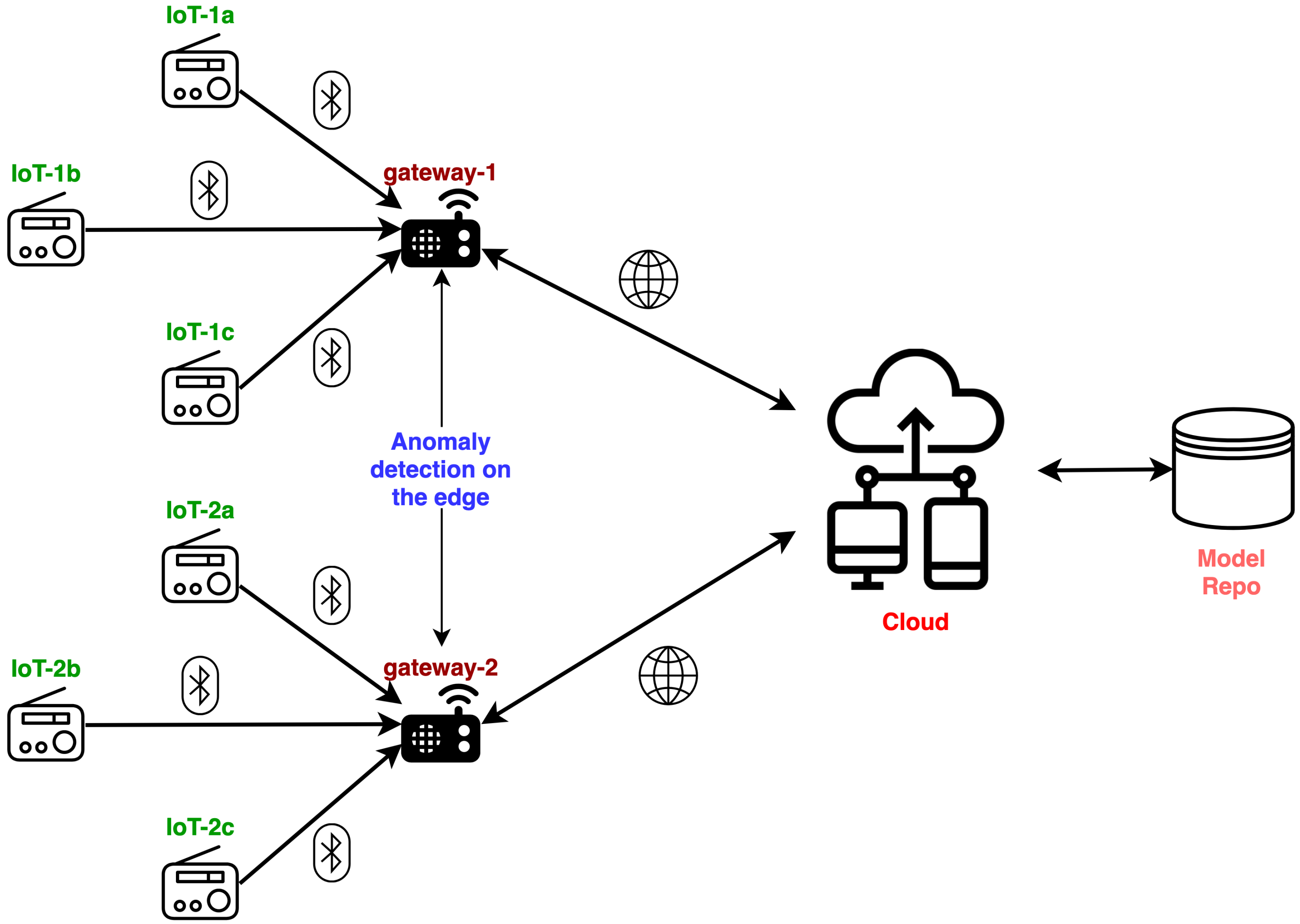
Unsupervised + Supervised

Demo

The notebook can be found here:-

https://github.com/tuhinsharma121/federated-ml/blob/master/notebooks/network-threat-detection-using-federated-learning.ipynb

Demo use case



1. Capture data.
2. Construct feature matrix
3. Train/Test Split
4. Setup environment
5. Prepare federated data.
6. Train model in federated way.
7. Save, Load, Predict.

Capture data

```python
import pandas as pd
pd.set_option("display.max_columns", 10)
# We use the KDD CUP 1999 data (https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html)
# 41 column names can be found at https://kdd.ics.uci.edu/databases/kddcup99/kddcup.names
colnames = ['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land',
            'wrong_fragment', 'urgent', 'hot', 'num_failed_logins', 'logged_in', 'num_compromised',
            'root_shell', 'su_attempted', 'num_root', 'num_file_creations', 'num_shells', 'num_access_files',
            'num_outbound_cmds', 'is_host_login', 'is_guest_login', 'count', 'srv_count', 'serror_rate',
            'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
            'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count', 'dst_host_same_srv_rate',
            'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
            'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
            'dst_host_srv_rerror_rate']

# We take 10% of the original data which can be found at
# http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz
df = pd.read_csv("http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz",
                 names=colnames+["threat_type"])

df.head(3)
```

	duration	protocol_type	service	flag	src_bytes	...	dst_host_serror_rate	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	threat_type
0	0	tcp	http	SF	181	...	0.0	0.0	0.0	0.0	normal.
1	0	tcp	http	SF	239	...	0.0	0.0	0.0	0.0	normal.
2	0	tcp	http	SF	235	...	0.0	0.0	0.0	0.0	normal.

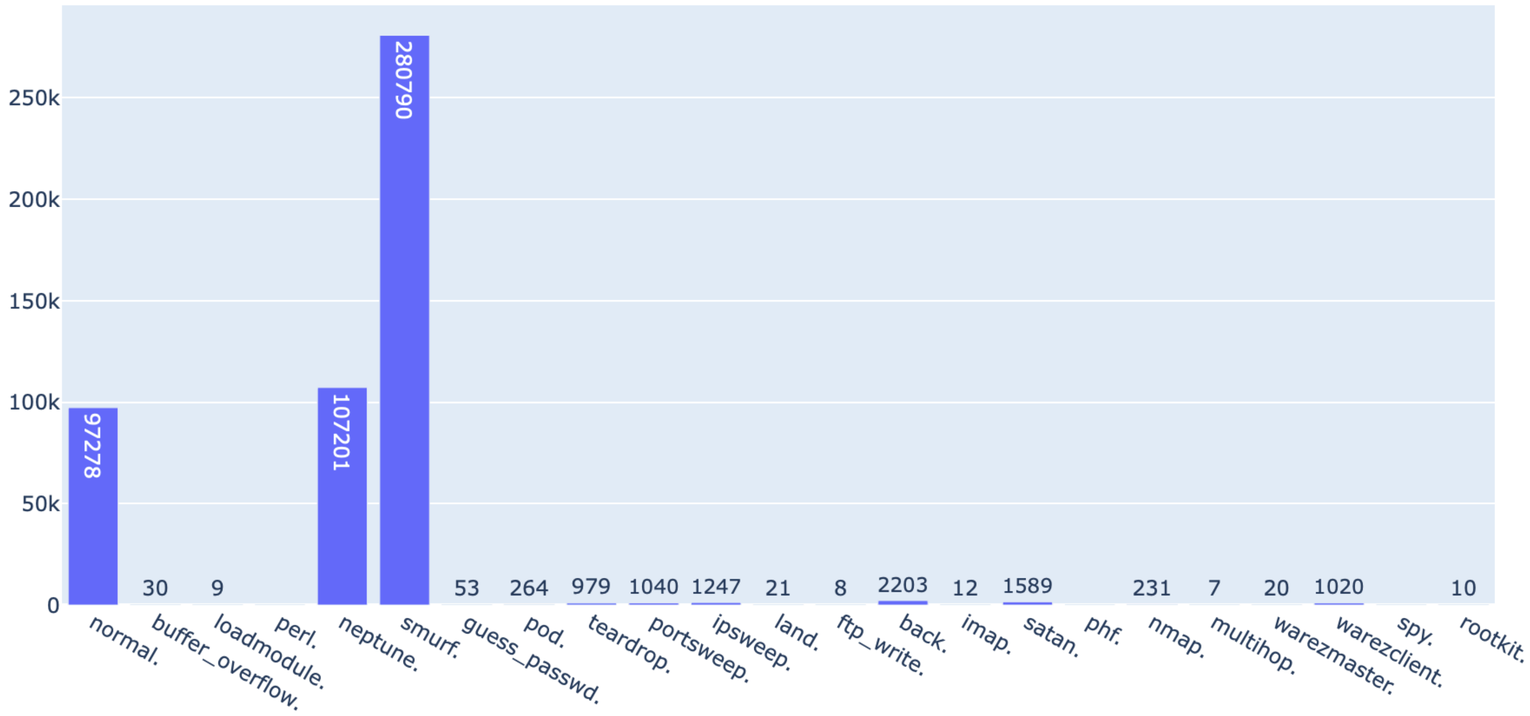
3 rows × 42 columns

Threat type distribution

```python
import plotly.graph_objects as go
from collections import Counter

threat_count_dict = Counter(df["threat_type"])
threat_types = list(threat_count_dict.keys())
threat_counts = [threat_count_dict[threat_type] for threat_type in threat_types]
print("Total distinct number of threat types : ",len(threat_types))
fig = go.Figure([go.Bar(x=threat_types, y=threat_counts,text=threat_counts,textposition='auto')])
fig.show()
```

```
Total distinct number of threat types :  23
```



Construct feature matrix and target vector

```
# 34 numerical columns are considered for training
numerical_colmanes = ['duration', 'src_bytes', 'dst_bytes', 'wrong_fragment', 'urgent', 'hot',
                      'num_failed_logins', 'num_compromised', 'root_shell', 'su_attempted', 'num_root',
                      'num_file_creations', 'num_shells', 'num_access_files', 'num_outbound_cmds', 'count',
                      'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate',
                      'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
                      'dst_host_srv_count', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
                      'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
                      'dst_host_srv_serror_rate', 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate']
numerical_df = df[numerical_colmanes].copy()
# Lets remove the numerical columns with constant value
numerical_df = numerical_df.loc[:, (numerical_df != numerical_df.iloc[0]).any()]
# lets scale the values for each column from [0,1]
# N.B. we dont have any negative values]
final_df = numerical_df/numerical_df.max()
X = final_df.values
# final dataframe has 33 features
print("Shape of feature matrix : ",X.shape)
```

```
Shape of feature matrix :  (494021, 33)
```

```
from sklearn.preprocessing import LabelEncoder

threat_types = df["threat_type"].values
encoder = LabelEncoder()
# use LabelEncoder to encode the threat types in numeric values
y = encoder.fit_transform(threat_types)
print("Shape of target vector : ",y.shape)
```

```
Shape of target vector :  (494021,)
```

Train/Test split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.4, random_state=42, stratify=y)
print("Number of records in training data : ", X_train.shape[0])
print("Number of records in test data : ", X_test.shape[0])
print("Total distinct number of threat types in training data : ",len(set(y_train)))
print("Total distinct number of threat types in test data : ",len(set(y_test)))
```

```
Number of records in training data :  296412
Number of records in test data :  197609
Total distinct number of threat types in training data :  23
Total distinct number of threat types in test data :  23
```

Stratified sampling preserves the class distribution after the split

Lets set up the environment for federated learning

```
%%capture
import torch
import syft as sy

# Hook PyTorch ie add extra functionalities to support Federated Learning
hook = sy.TorchHook(torch)
# Sets the seed for generating random numbers.
torch.manual_seed(1)
# Select CPU computation, in case you want GPU use "gpu" instead
device = torch.device("cpu")
# Data will be distributed among these VirtualWorkers.
# Remote training of the model will happen here.
gatway1 = sy.VirtualWorker(hook, id="gatway1")
gatway2 = sy.VirtualWorker(hook, id="gatway2")
```

In these 2 gateways data will reside and models will be trained

Lets set the training parameters

```python
import numpy as np

# Number of times we want to iterate over whole training data
BATCH_SIZE = 1000
EPOCHS = 5
LOG_INTERVAL = 120
lr = 0.01

n_feature = X_train.shape[1]
n_class = np.unique(y_train).shape[0]

print("Number of training features : ",n_feature)
print("Number of training classes : ",n_class)
```

```
Number of training features :  33
Number of training classes :  23
```

Prepare federated data and distribute across the gateways

```
# Create pytorch tensor from X_train,y_train,X_test,y_test
train_inputs = torch.tensor(X_train,dtype=torch.float).tag("#iot", "#network","#data","#train")
train_labels = torch.tensor(y_train).tag("#iot", "#network","#target","#train")
test_inputs = torch.tensor(X_test,dtype=torch.float).tag("#iot", "#network","#data","#test")
test_labels = torch.tensor(y_test).tag("#iot", "#network","#target","#test")

# Send the training and test data to the gatways in equal proportion.
train_idx = int(len(train_labels)/2)
test_idx = int(len(test_labels)/2)
gatway1_train_dataset = sy.BaseDataset(train_inputs[:train_idx], train_labels[:train_idx]).send(gatway1)
gatway2_train_dataset = sy.BaseDataset(train_inputs[train_idx:], train_labels[train_idx:]).send(gatway2)
gatway1_test_dataset = sy.BaseDataset(test_inputs[:test_idx], test_labels[:test_idx]).send(gatway1)
gatway2_test_dataset = sy.BaseDataset(test_inputs[test_idx:], test_labels[test_idx:]).send(gatway2)

# Create federated datasets, an extension of Pytorch TensorDataset class
federated_train_dataset = sy.FederatedDataset([gatway1_train_dataset, gatway2_train_dataset])
federated_test_dataset = sy.FederatedDataset([gatway1_test_dataset, gatway2_test_dataset])

# Create federated dataloaders, an extension of Pytorch DataLoader class
federated_train_loader = sy.FederatedDataLoader(federated_train_dataset, shuffle=True, batch_size=BATCH_SIZE)
federated_test_loader = sy.FederatedDataLoader(federated_test_dataset, shuffle=False, batch_size=BATCH_SIZE)
```

Lets define a simple logistic regression model

```python
import torch.nn as nn
class Net(nn.Module):
    def __init__(self, input_dim, output_dim):
        """
        input_dim: number of input features.
        output_dim: number of labels.
        """
        super(Net, self).__init__()
        self.linear = torch.nn.Linear(input_dim, output_dim)
    def forward(self, x):
        outputs = self.linear(x)
        return outputs
```

It can be any PyTorch DL model

Lets define the training process

```
import torch.nn.functional as F

def train(model, device, federated_train_loader, optimizer, epoch):
    model.train()
    # Iterate through each gateway's dataset
    for batch_idx, (data, target) in enumerate(federated_train_loader):
        # Send the model to the right gateway
        model.send(data.location)
        # Move the data and target labels to the device (cpu/gpu) for computation
        data, target = data.to(device), target.to(device)
        # Clear previous gradients (if they exist)
        optimizer.zero_grad()
        # Make a prediction
        output = model(data)
        # Calculate the cross entropy loss [We are doing classification]
        loss = F.cross_entropy(output, target)
        # Calculate the gradients
        loss.backward()
        # Update the model weights
        optimizer.step()
        # Get the model back from the gateway
        model.get()
        if batch_idx!=0 and batch_idx % LOG_INTERVAL == 0:
            # get the loss back
            loss = loss.get()
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * BATCH_SIZE, len(federated_train_loader) * BATCH_SIZE,
                100. * batch_idx / len(federated_train_loader), loss.item()))
```

Lets define the validation process

```
import torch.nn.functional as F

def test(model, device, federated_test_loader):
    model.eval()
    correct = 0
    with torch.no_grad():
        for batch_idx, (data, target) in enumerate(federated_test_loader):
            # Send the model to the right gateway
            model.send(data.location)
            # Move the data and target labels to the device (cpu/gpu) for computation
            data, target = data.to(device), target.to(device)
            # Make a prediction
            output = model(data)
            # Get the model back from the gateway
            model.get()
            # Calculate the cross entropy loss
            loss = F.cross_entropy(output, target)
            # Get the index of the max log-probability
            pred = output.argmax(1, keepdim=True)
            # Get the number of instances correctly predicted
            correct += pred.eq(target.view_as(pred)).sum().get()

    # get the loss back
    loss = loss.get()
    print('Test set: Loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        loss.item(), correct, len(federated_test_loader.federated_dataset),
        100. * correct / len(federated_test_loader.federated_dataset)))
```

Lets train the model in federated way

```
%%time
import torch.optim as optim

# Initialize the model
model = Net(n_feature,n_class)

#Initialize the SGD optimizer
optimizer = optim.SGD(model.parameters(), lr=lr)

for epoch in range(1, EPOCHS + 1):
    # Train on the training data in a federated way
    train(model, device, federated_train_loader, optimizer, epoch)
    # Check the test accuracy on unseen test data in a federated way
    test(model, device, federated_test_loader)
```

```
Train Epoch: 1 [120000/297000 (40%)]    Loss: 1.401350
Train Epoch: 1 [240000/297000 (81%)]    Loss: 0.981308
Test set: Loss: 0.8373, Accuracy: 146833/197609 (74%)

Train Epoch: 2 [120000/297000 (40%)]    Loss: 0.692904
Train Epoch: 2 [240000/297000 (81%)]    Loss: 0.475993
Test set: Loss: 0.4931, Accuracy: 182811/197609 (92%)

Train Epoch: 3 [120000/297000 (40%)]    Loss: 0.426403
Train Epoch: 3 [240000/297000 (81%)]    Loss: 0.389922
Test set: Loss: 0.3548, Accuracy: 190944/197609 (96%)

Train Epoch: 4 [120000/297000 (40%)]    Loss: 0.313678
Train Epoch: 4 [240000/297000 (81%)]    Loss: 0.316881
Test set: Loss: 0.2834, Accuracy: 191805/197609 (97%)

Train Epoch: 5 [120000/297000 (40%)]    Loss: 0.294926
Train Epoch: 5 [240000/297000 (81%)]    Loss: 0.299990
Test set: Loss: 0.2410, Accuracy: 192059/197609 (97%)
```

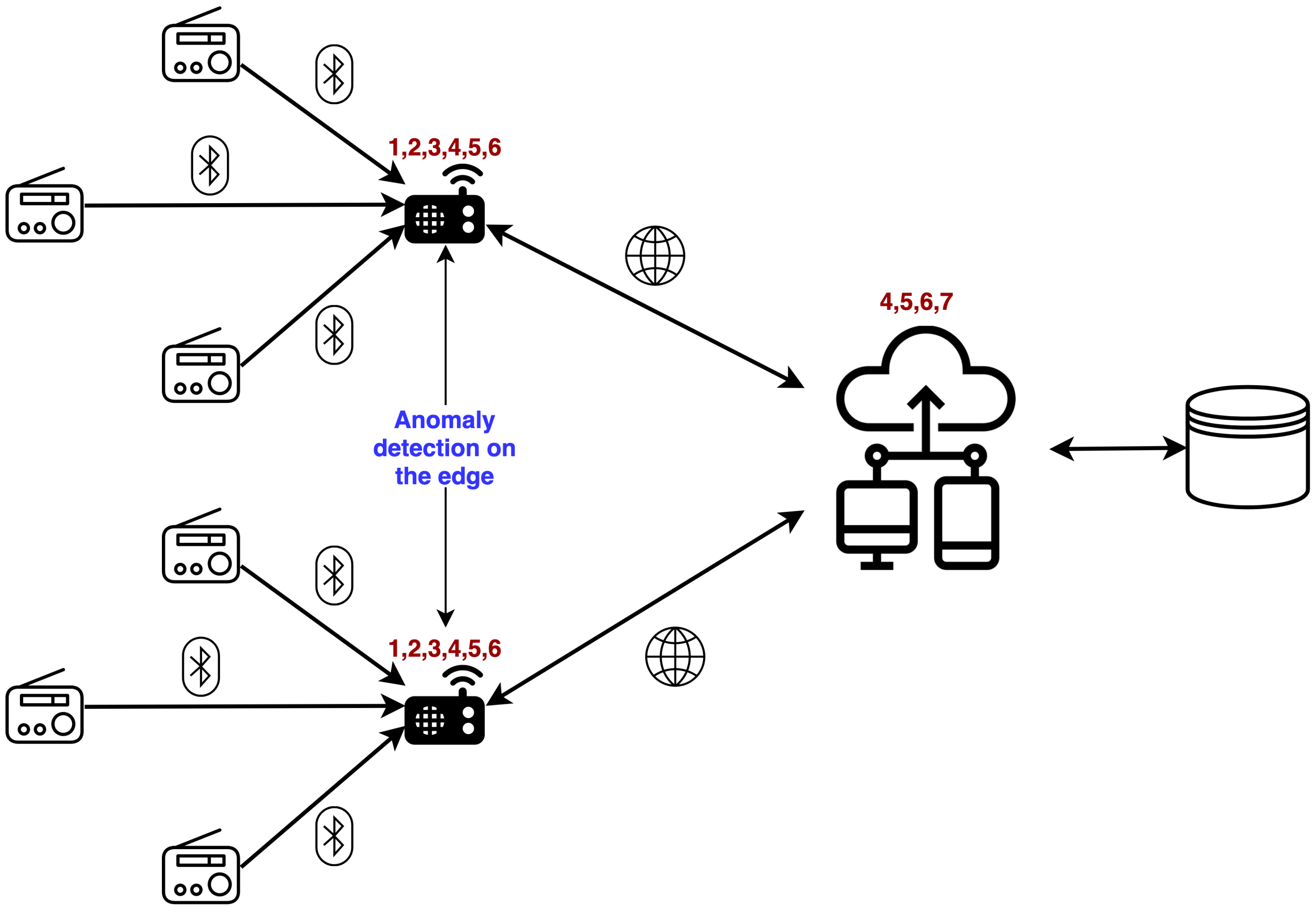
Save, Reload and Use the model to predict one network traffic data

```
# Save the model
torch.save(model.state_dict(), "binaize-threat-model.pt")
# Reload the model in a new model object
model_new = Net(n_feature,n_class)
model_new.load_state_dict(torch.load("binaize-threat-model.pt"))
model_new.eval()

# Take the 100th record from the test data
idx = 100
data = test_inputs[idx]
pred = model_new(test_inputs[0])
pred_label = int(pred.argmax().data.cpu().numpy())
pred_threat = encoder.inverse_transform([pred_label])[0]
print("Predicted threat type : ", pred_threat)
actual_label = int(test_labels[idx].data.cpu().numpy())
actual_threat = encoder.inverse_transform([actual_label])[0]
print("Actual threat type : ", actual_threat)
```

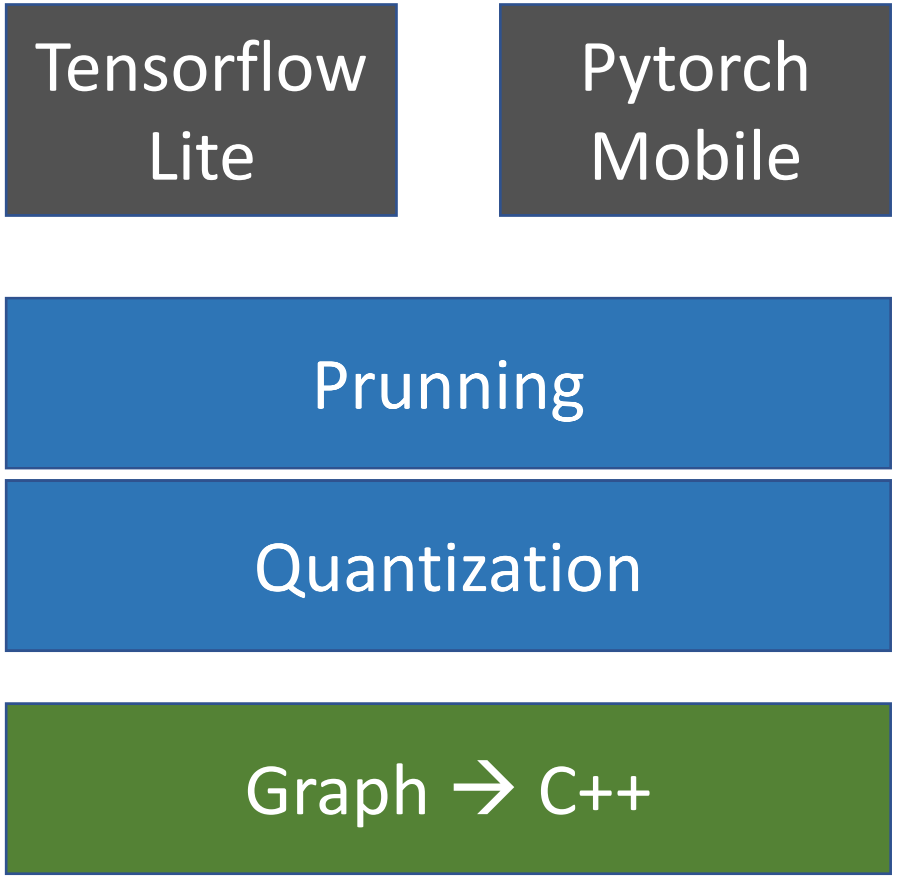
```
Predicted threat type :  smurf.
Actual threat type :  smurf.
```

Demo use case



1. Capture data.
2. Construct feature matrix
3. Train/Test Split
4. Setup environment
5. Prepare federated data.
6. Train model in federated way.
7. Save, Load, Predict.

Some of our design choices

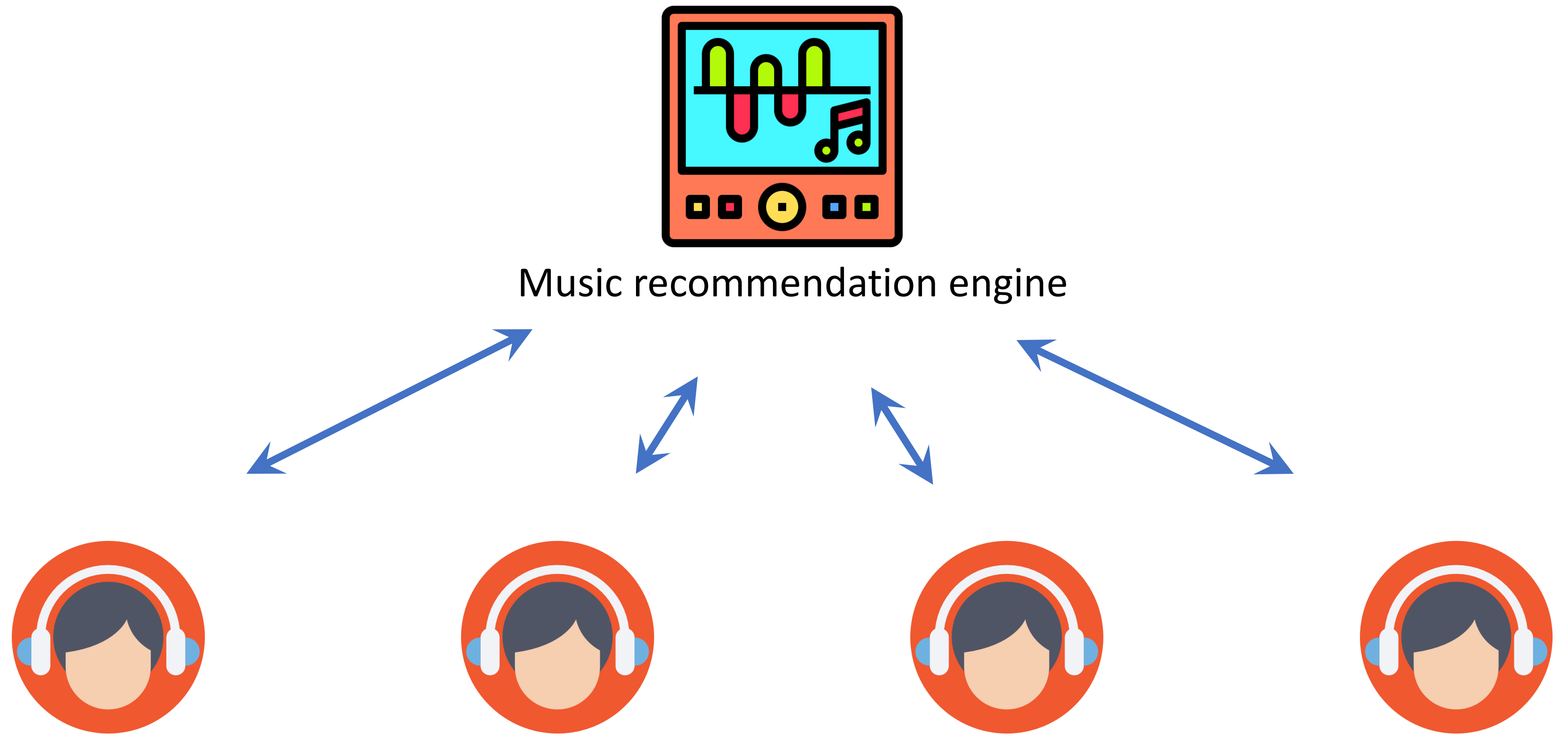


Types of Federated Learning

Single Party Federated Learning.

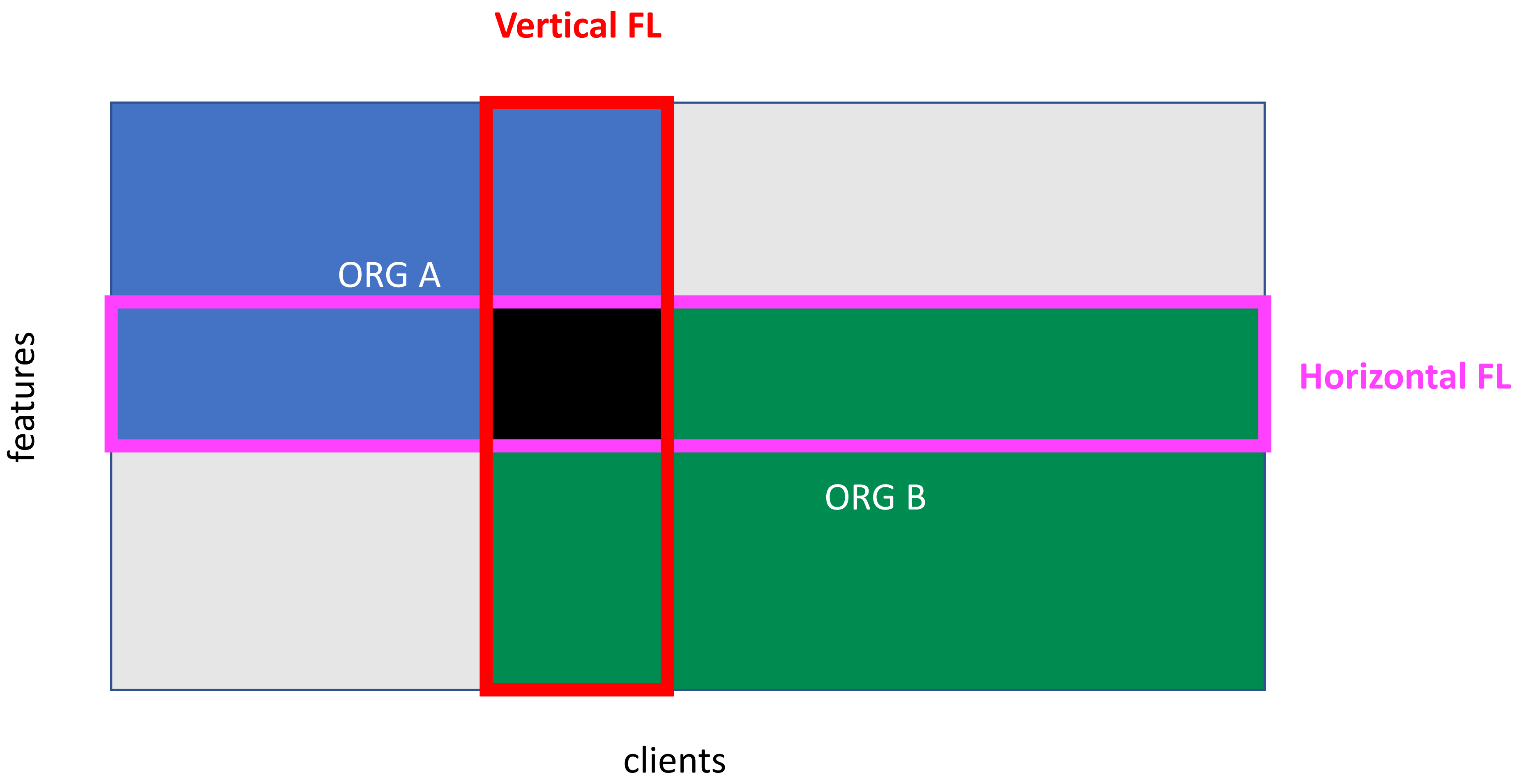
Multi Party Federated Learning.

Single Party Federated Learning



only one entity is involved in governance of the distributed data capture and flow system

Multi Party Federated Learning

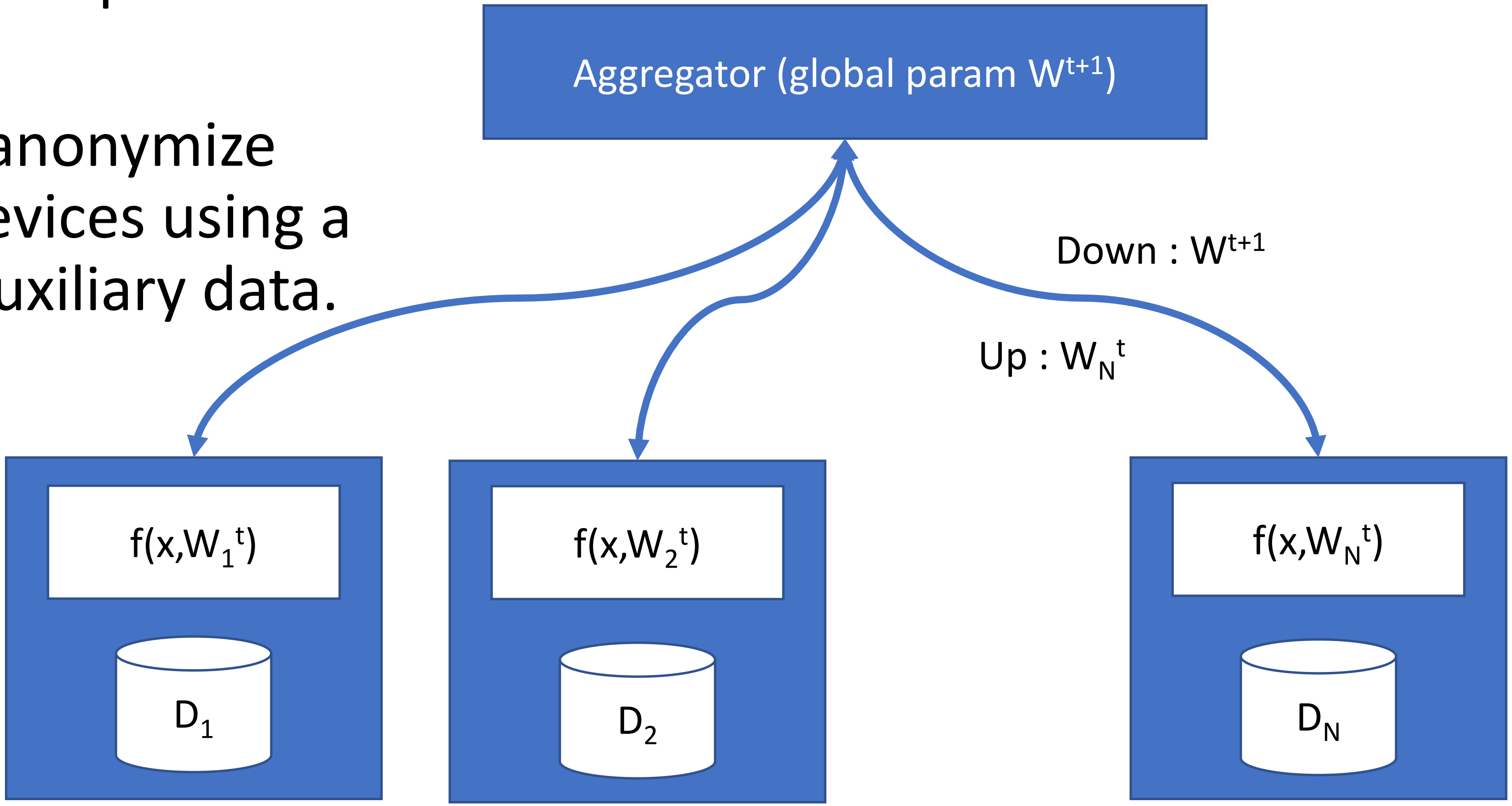


Challenges in Federated Learning

- Inference Attack.
- Model Poisoning.

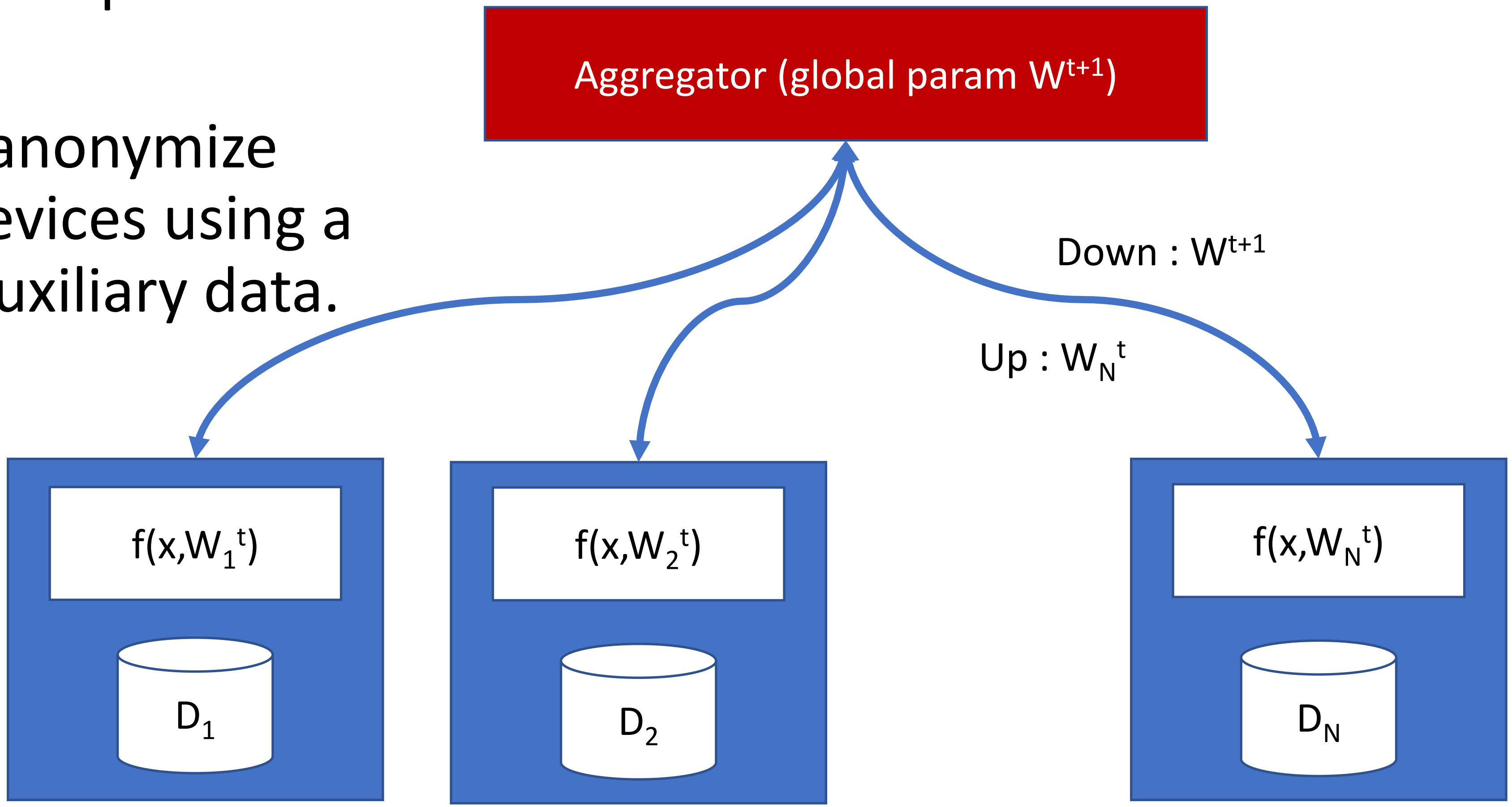
Inference Attack

- Model deltas encode subtle variations of user specific information.
- Possible to de-anonymize participating devices using a limited set of auxiliary data.



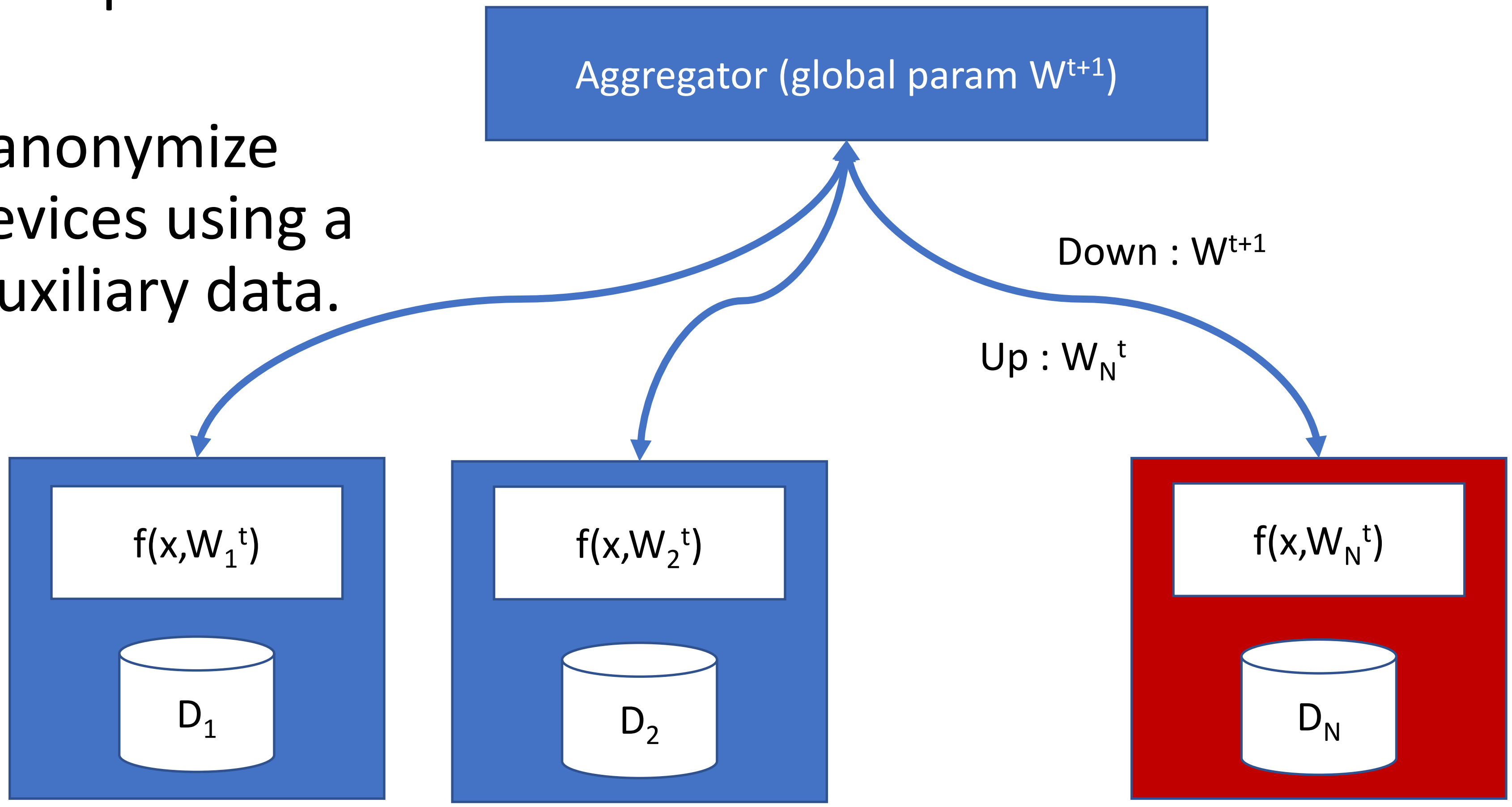
Inference Attack

- Model deltas encode subtle variations of user specific information.
- Possible to de-anonymize participating devices using a limited set of auxiliary data.

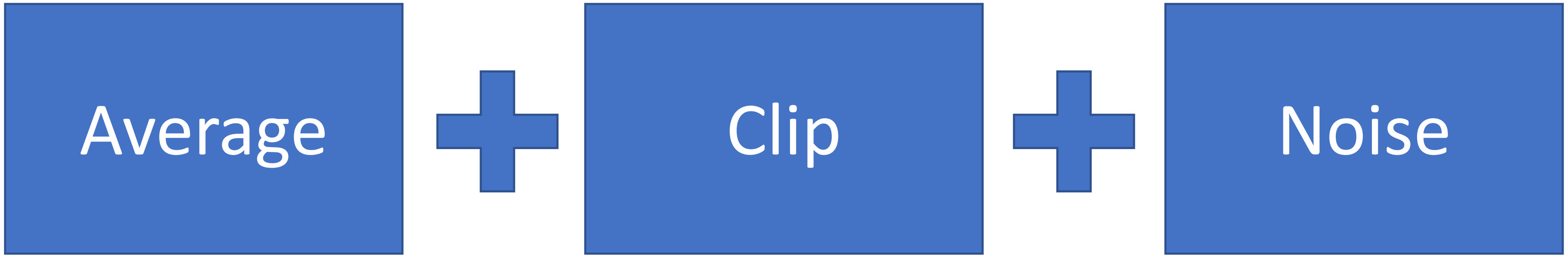


Inference Attack

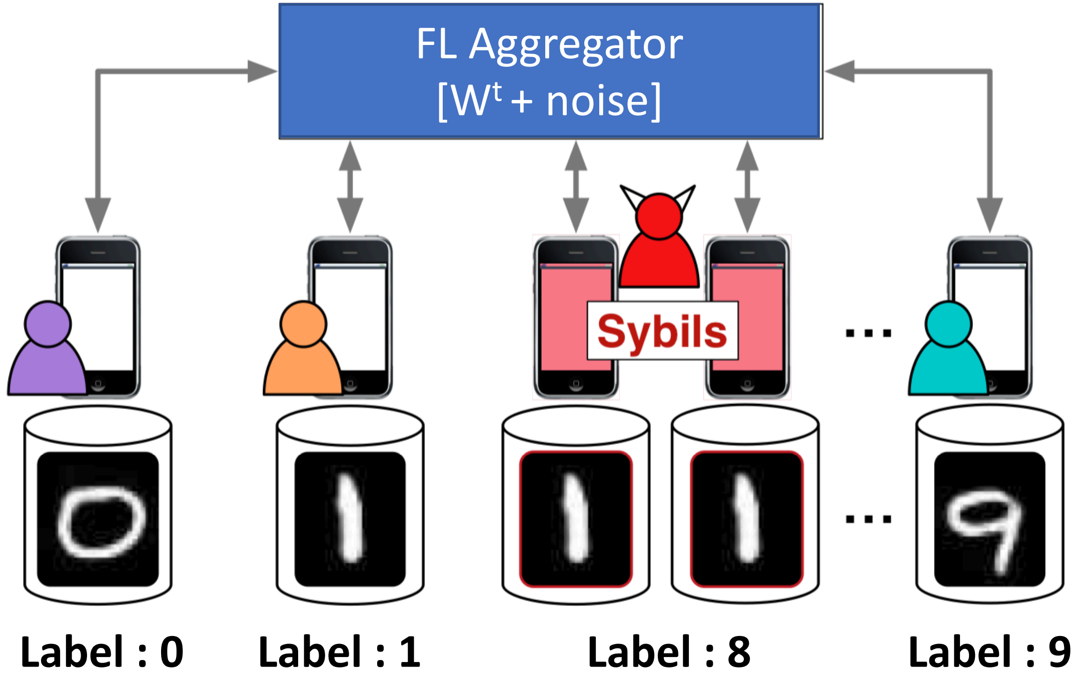
- Model deltas encode subtle variations of user specific information.
- Possible to de-anonymize participating devices using a limited set of auxiliary data.

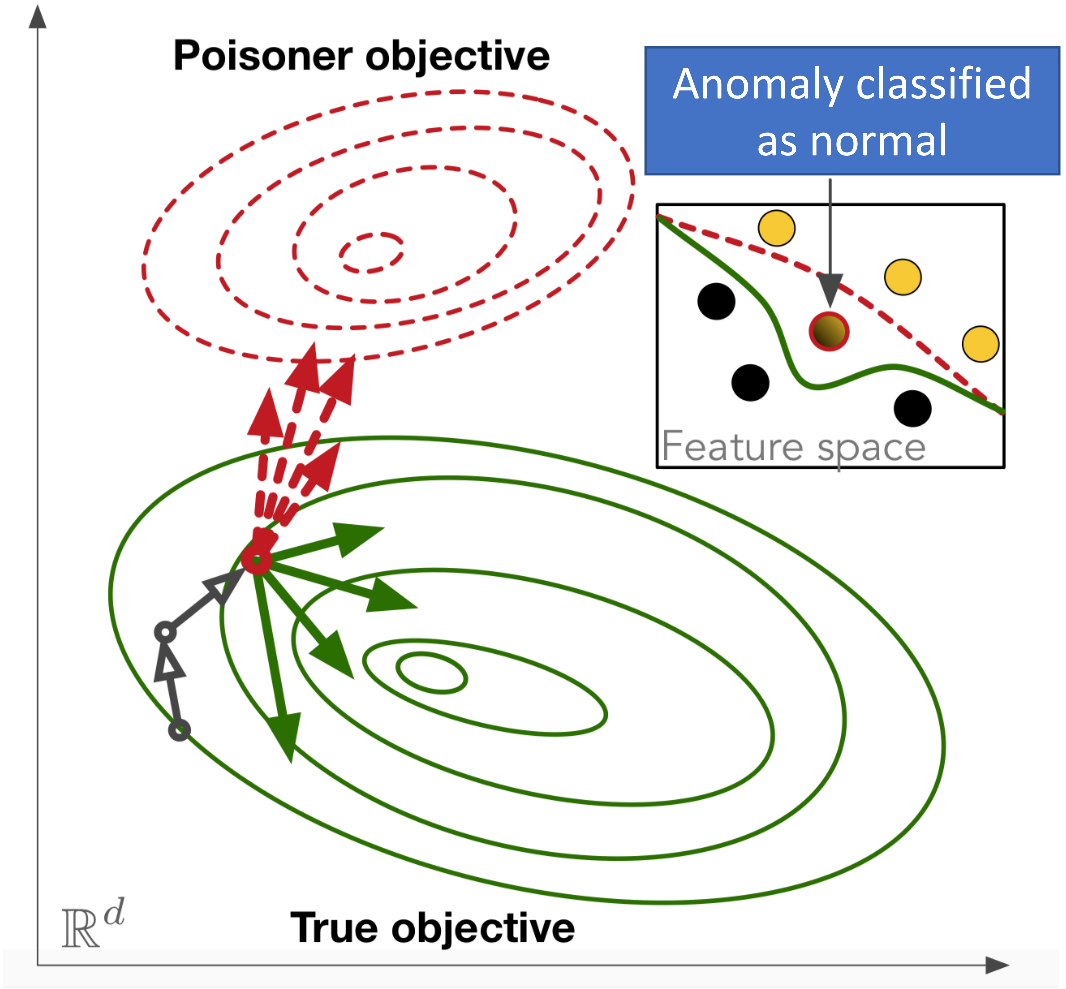


Solution: Differential Privacy

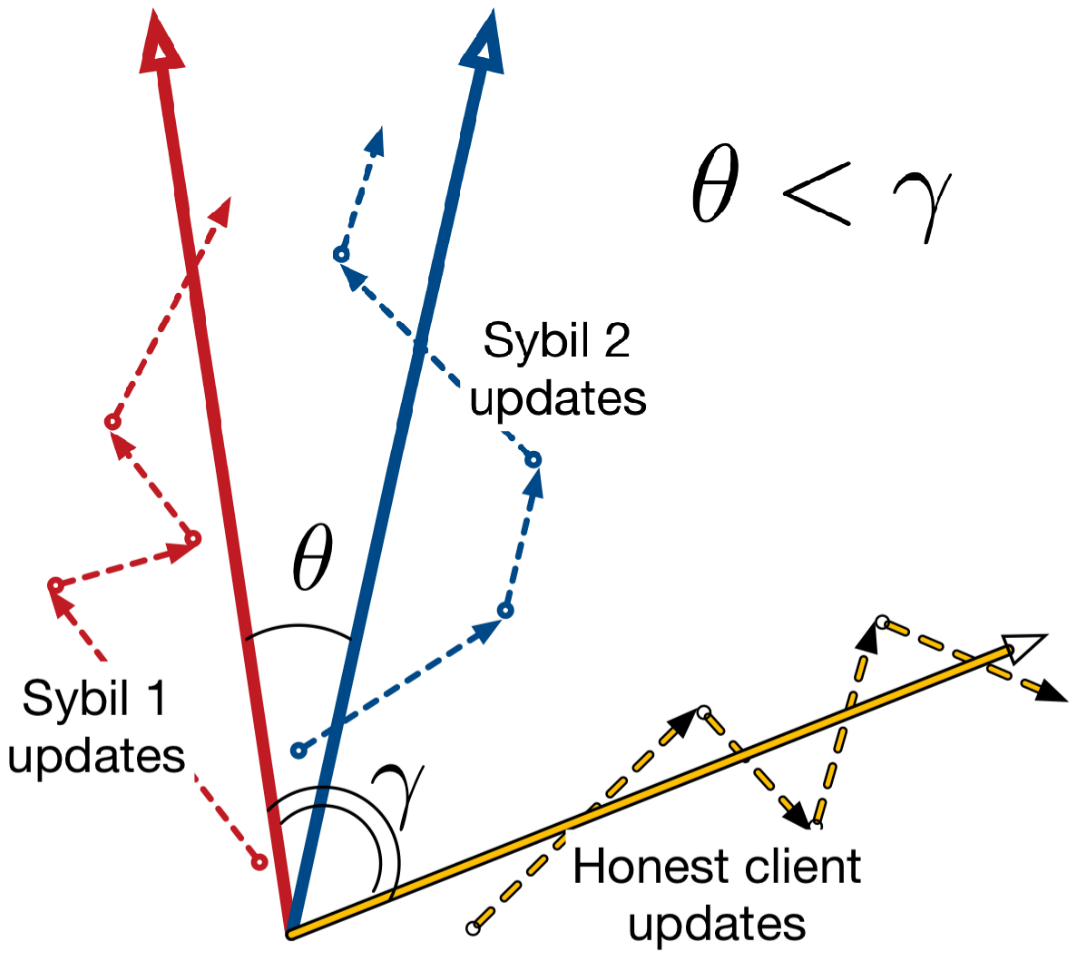


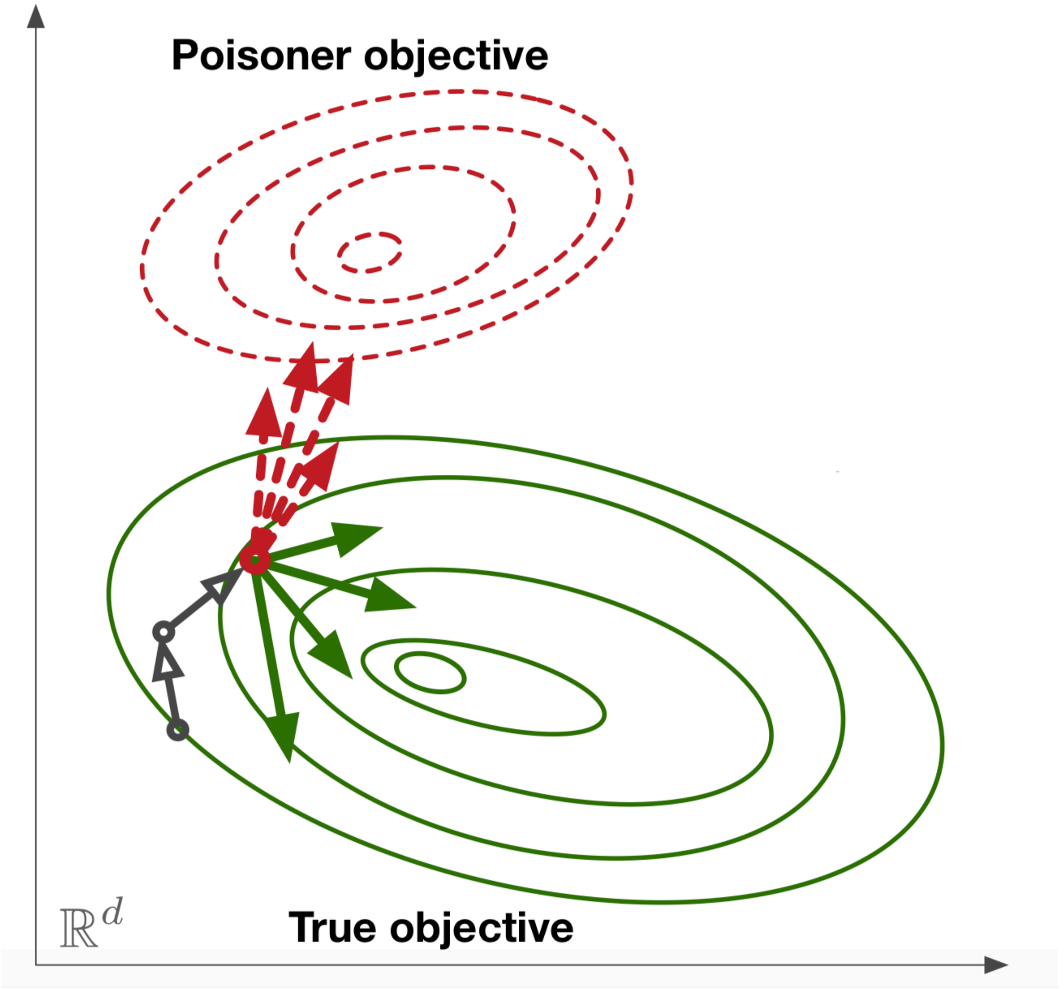
Model Poisoning





Solution: Sybil Detection



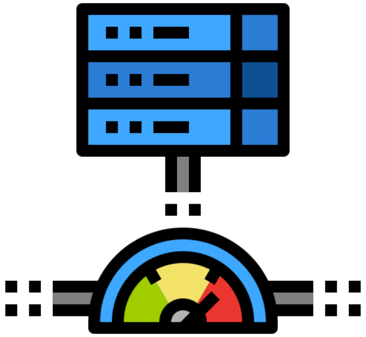


Benefits

LOWER LATENCY

Benefits

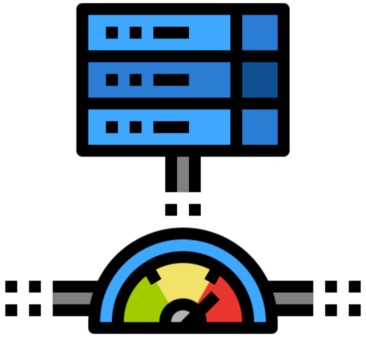
LOWER LATENCY



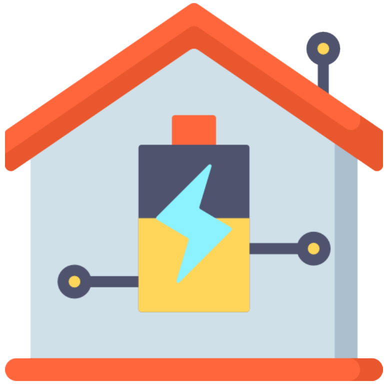
LESS NETWORK LOAD

Benefits

LOWER LATENCY



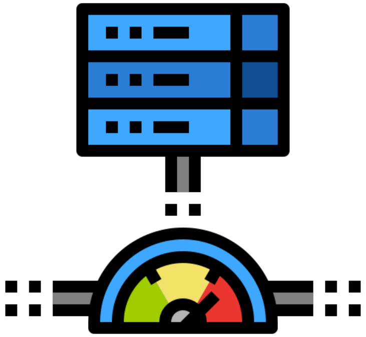
LESS NETWORK LOAD



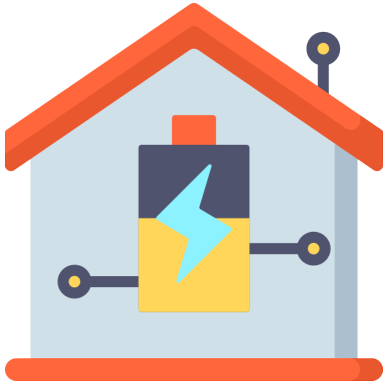
LESS POWER CONSUMPTION

Benefits

LOWER LATENCY



LESS NETWORK LOAD

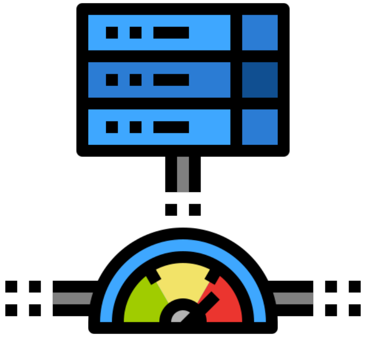


LESS POWER CONSUMPTION

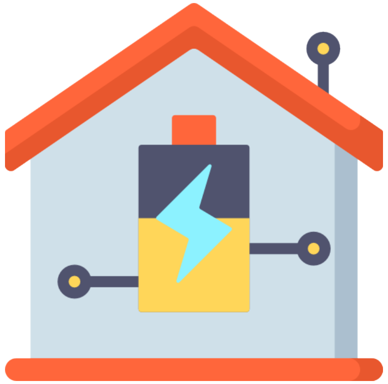
PRIVACY

Benefits

LOWER LATENCY



LESS NETWORK LOAD



LESS POWER CONSUMPTION

PRIVACY

ACROSS ORGANIZATIONS

Acknowledgements

- https://github.com/OpenMined/PySyft
- "Federated Learning: Strategies for Improving Communication Efficiency" by Jakub Konečný,H. Brendan McMahan,Felix X. Yu,Peter Richtarik,Ananda Theertha Suresh,Dave Bacon
- "Gradient-Leaks: Understanding and Controlling Deanonymization in Federated Learning" by Tribhuvanesh Orekondy, Seong Joon Oh, Yang Zhang, Bernt Schiele, Mario Fritz
- "Comprehensive Privacy Analysis of Deep Learning: Stand-alone and Federated Learning under Passive and Active White-box Inference Attacks" by "Milad Nasr, Reza Shokri, Amir Houmansadr
- https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf
- "Mitigating Sybils in Federated Learning Poisoning" by Clement Fung, Chris J.M. Yoon, Ivan Beschastnikh

THANK YOU

Tuhin Sharma | Binaize Labs

@tuhinsharma121