

O'REILLY®

Artificial Intelligence Conference



Anomaly Detection in Smart Buildings using Federated Learning

oreillyaicon.com
#OReillyAI

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.

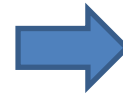
- Forbes 2019



ARTIFICIAL INTELLIGENCE



SMART BUILDING



SMARTER BUILDING

How AI is helping buildings become smarter



WATER MANAGEMENT



BUILDING MAINTENANCE



PARKING ASSISTANCE



SMART BULBS MANAGEMENT

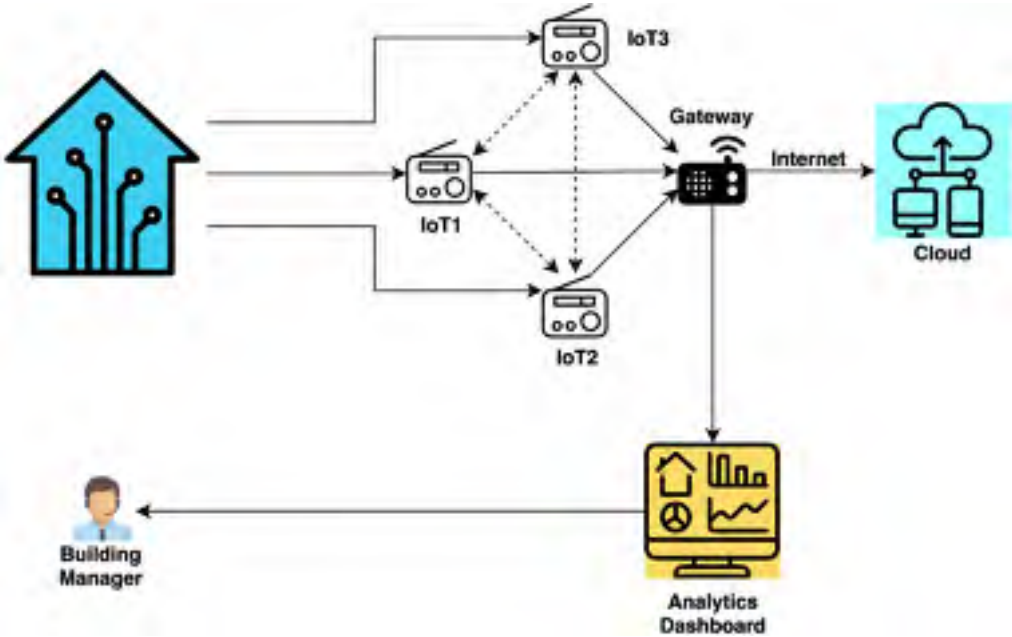


HVAC MANAGEMENT

Smart HVAC Management



HVAC MANAGEMENT



Challenges in Smart Building



DATA CORRUPTION



CYBER BREACH



Anomaly detection is critical

© 2017 REDISSA SOFTWARES U.C. ALL RIGHTS RESERVED.

O'REILLY[®]
Artificial Intelligence

PRESENTED WITH



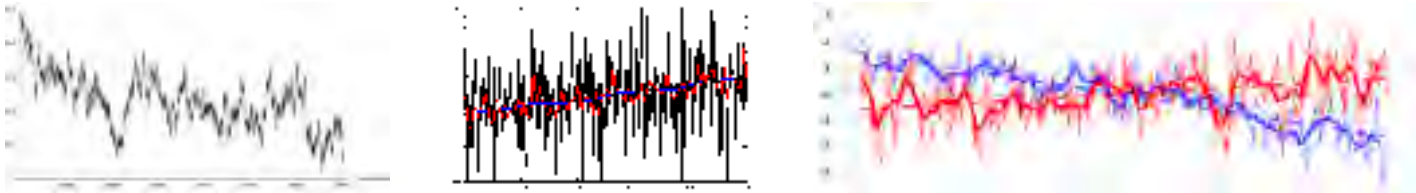
oreillyaicon.com
#OReillyAI

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 ML 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**



**HIGH DATA VOLUME
AND VELOCITY**



LIMITED BATTERY

But connected devices present a number of novel challenges



**INTERMITTENT
INTERNET CONNECTION**



**HIGH DATA VOLUME
AND VELOCITY**



LIMITED BATTERY



**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

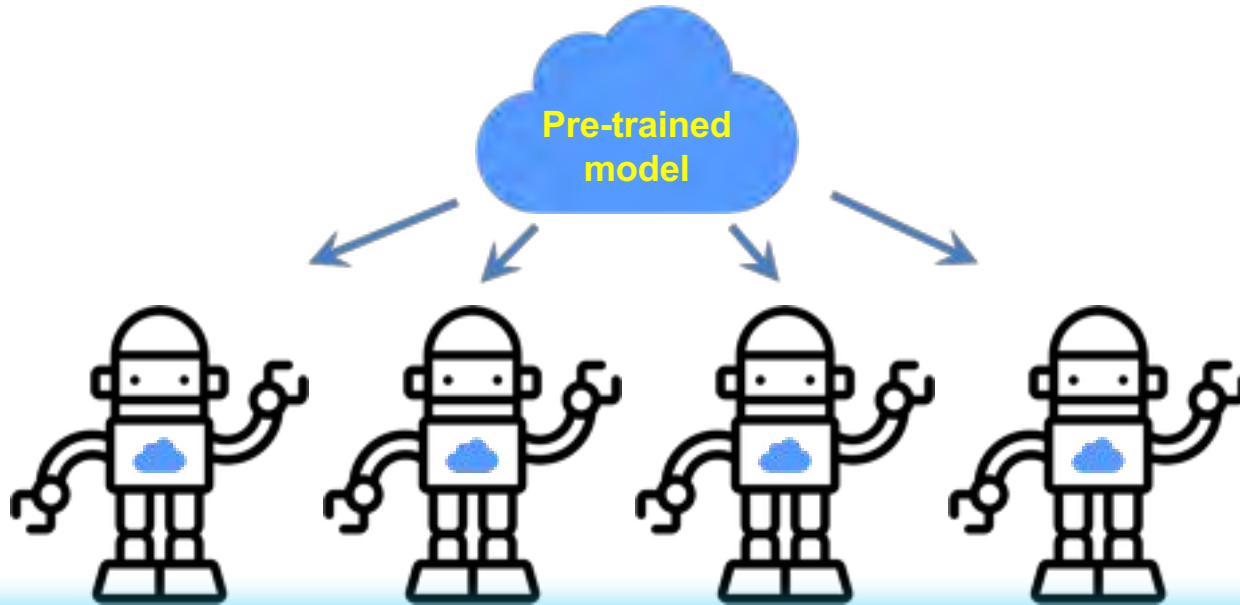


Steps for Federated Learning

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

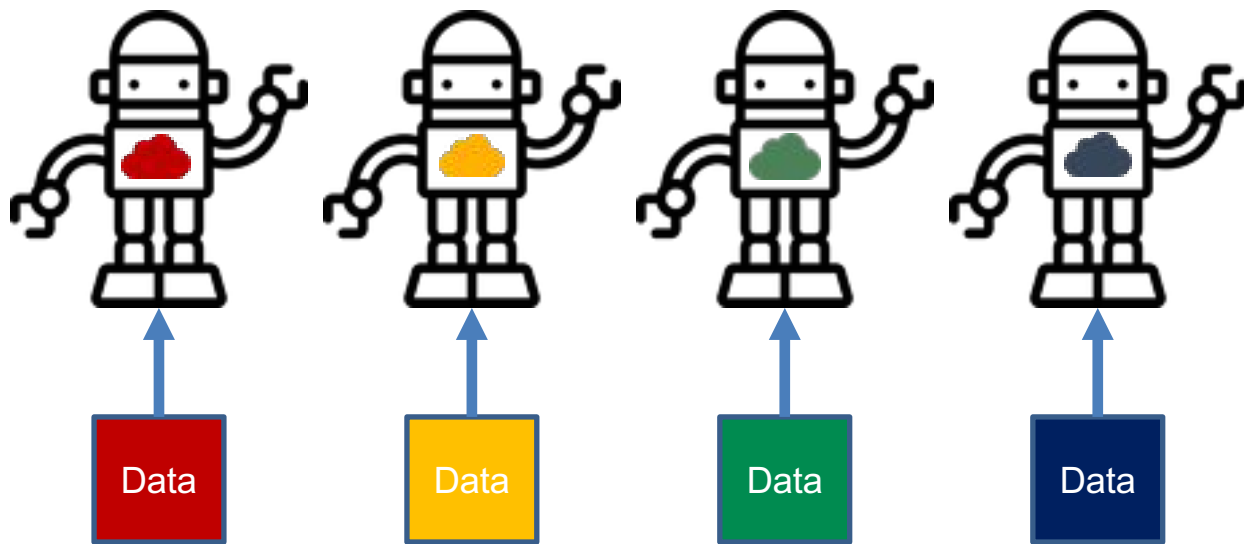


(a) Federation Construction



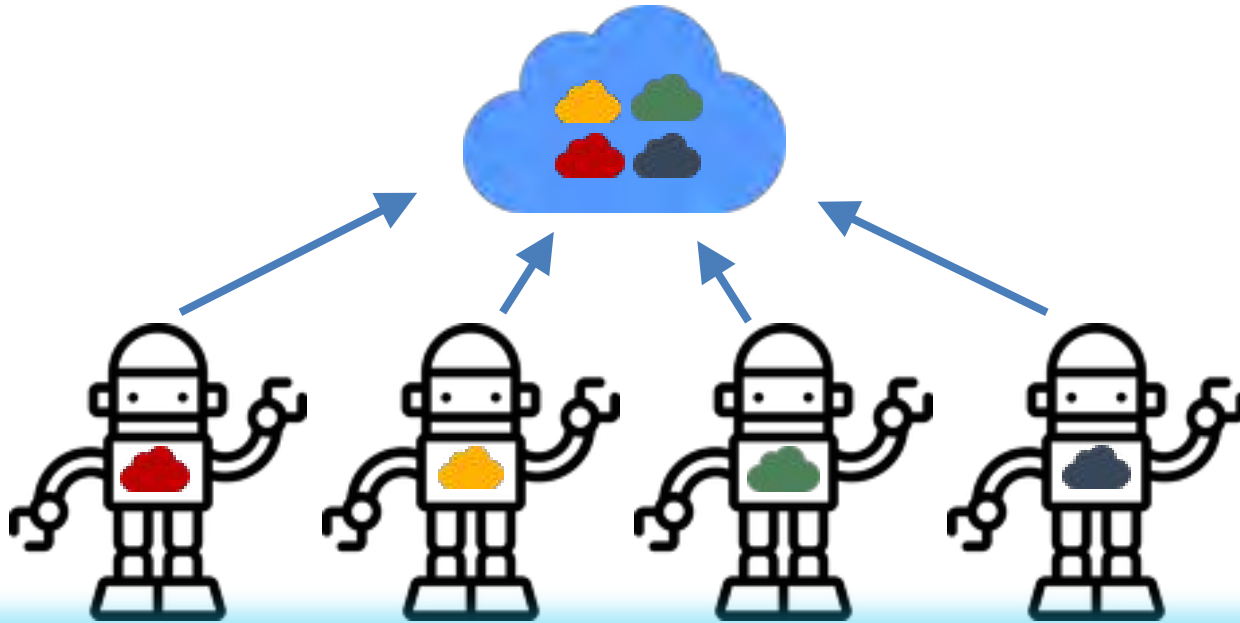
A random subset of members of the devices is selected to receive the global model synchronously from the server.

(b) Decentralized Training



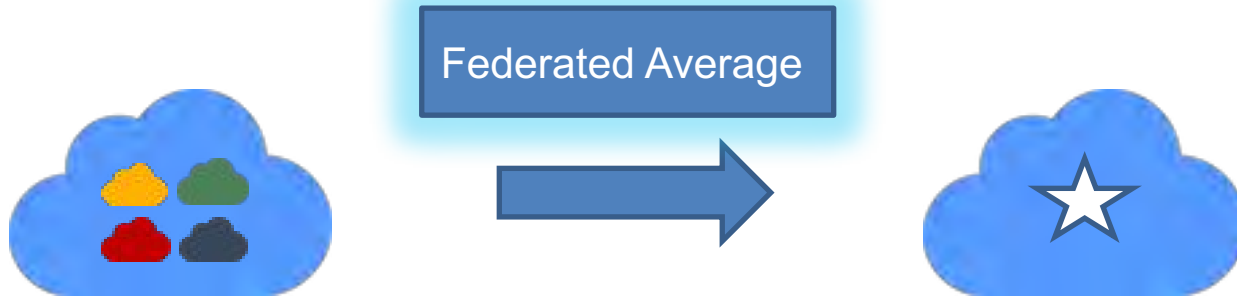
Each selected device computes an updated model using its local data.

(c) Model Accumulation



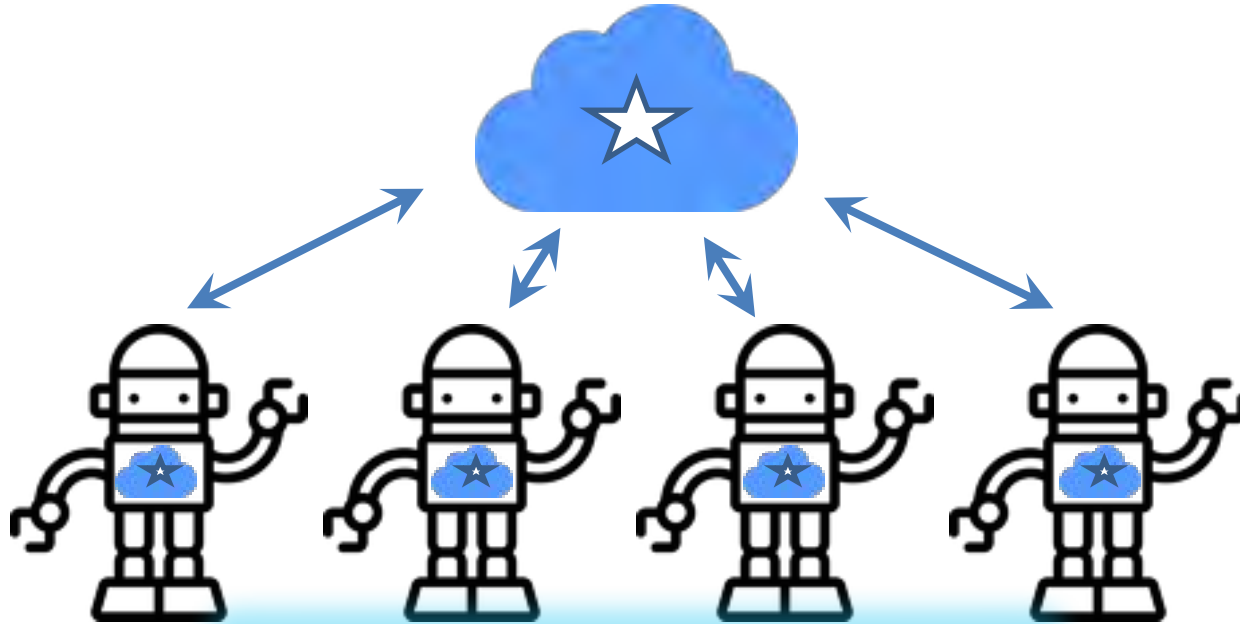
Only the model updates are sent from the federation to the server. Data is not moved.

(d) Model Aggregation



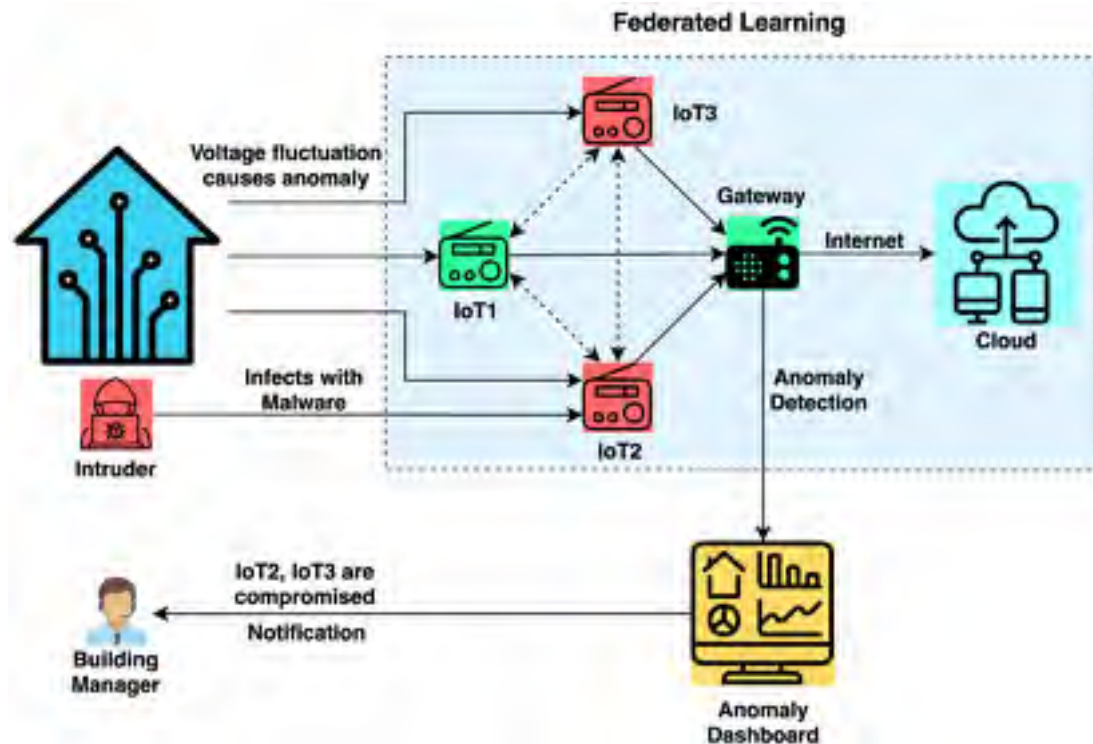
The server aggregates these model weights (typically by averaging) to construct an improved global model.

Federated Learning (Rinse, Repeat)



The devices receive the updated model.

Use Case



Tools- Choices



WeBank 微众银行

FATE

We use PySyft

Our journey

Rules + Z-score

K-Means +
Isolation
Forest +
OneClass SVM

Unsupervised

Deep Auto-Encoder

Federated Learning

Unsupervised + Supervised

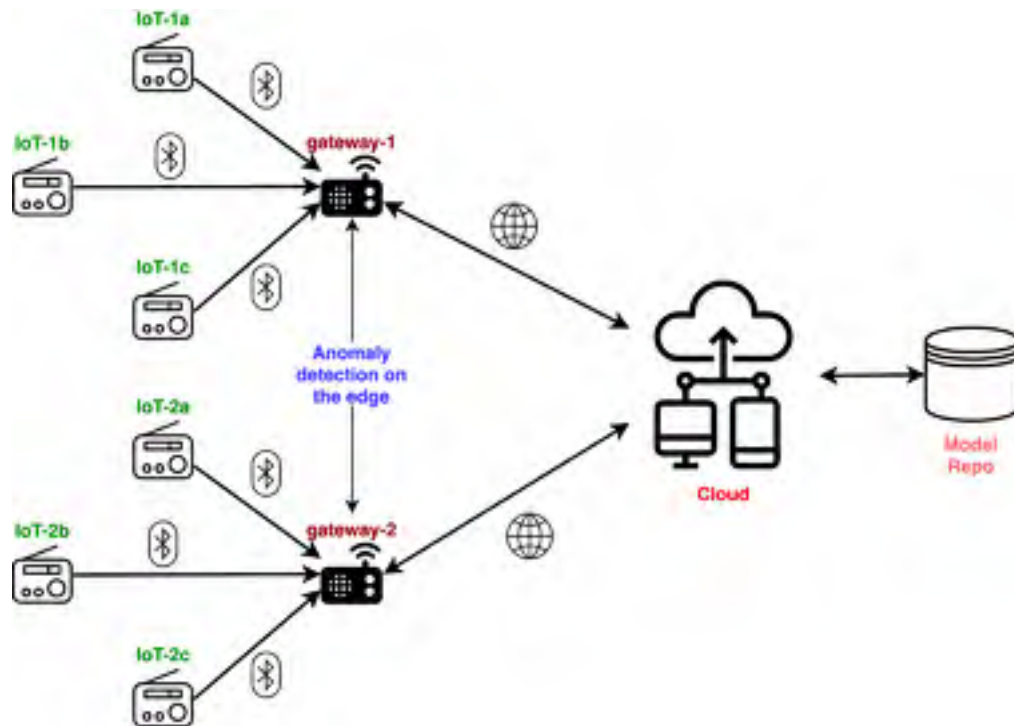


Demo

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

```
1 import pandas as pd
2 pd.set_option("display.max_columns", 10)
3 # We use the KDD CUP 1999 data (https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html)
4 # 41 column names can be found at https://kdd.ics.uci.edu/databases/kddcup99/kddcup.names
5 colnames = ['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land',
6             'wrong_fragment', 'urgent', 'hot', 'num_failed_logins', 'logged_in', 'num_compromised',
7             'root_shell', 'su_attempted', 'num_root', 'num_file_creations', 'num_shells', 'num_access_files',
8             'num_outbound_cmds', 'is_host_login', 'is_guest_login', 'count', 'srv_count', 'error_rate',
9             'srv_error_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
10            'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count', 'dst_host_same_srv_rate',
11            'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
12            'dst_host_error_rate', 'dst_host_srv_error_rate', 'dst_host_rerror_rate',
13            'dst_host_srv_rerror_rate']
14
15 # We take 10% of the original data which can be found at
16 # http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz
17 df = pd.read_csv("http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz",
18                 names=colnames+["threat_type"])
19
20 df.head(3)
```

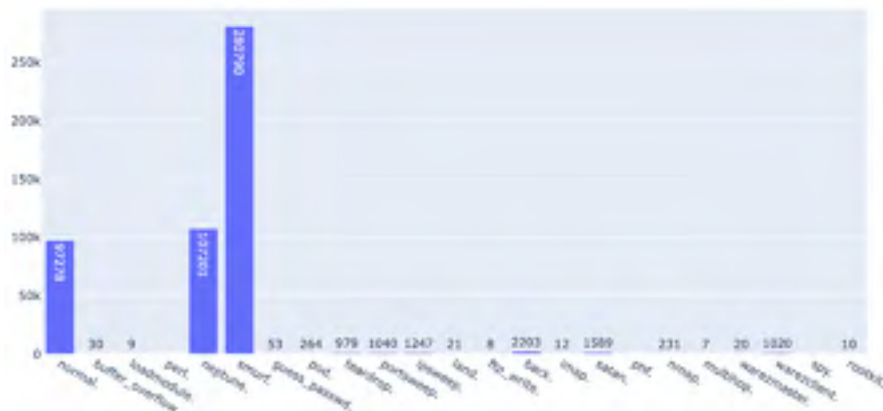
	duration	protocol_type	service	flag	src_bytes	...	dst_host_error_rate	dst_host_srv_error_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	threat_type
0	0	tcp	http	SF	181	...	0.0	0.0	0.0	0.0	normal.
1	0	tcp	http	SF	239	...	0.0	0.0	0.0	0.0	normal.
2	0	tcp	http	SF	235	...	0.0	0.0	0.0	0.0	normal.

3 rows x 42 columns

Threat type distribution

```
1 import plotly.graph_objects as go
2 from collections import Counter
3
4 threat_count_dict = Counter(df['threat_type'])
5 threat_types = list(threat_count_dict.keys())
6 threat_counts = [threat_count_dict[threat_type] for threat_type in threat_types]
7 print("Total distinct number of threat types : ", len(threat_types))
8 fig = go.Figure([go.Bar(x=threat_types, y=threat_counts, text=threat_counts, textposition='auto')])
9 fig.show()
```

Total distinct number of threat types : 23



Construct feature matrix and target vector

```
1 # 34 numerical columns are considered for training
2 numerical_colnames = ['duration', 'src_bytes', 'dst_bytes', 'wrong_fragment', 'urgent', 'hot',
3                       'num_failed_logins', 'num_compromised', 'root_shell', 'su_attempted', 'num_root',
4                       'num_file_creations', 'num_shells', 'num_access_files', 'num_outbound_cmds', 'count',
5                       'srv_count', 'serror_rate', 'srv_error_rate', 'rerror_rate', 'srv_rerror_rate',
6                       'same_srv_rate', 'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_count',
7                       'dst_host_srv_count', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
8                       'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate', 'dst_host_rerror_rate',
9                       'dst_host_srv_rerror_rate', 'dst_host_rerror_rate', 'dst_host_srv_rerror_rate']
10 numerical_df = df[numerical_colnames].copy()
11 # Lets remove the numerical columns with constant value
12 numerical_df = numerical_df.loc[:, (numerical_df != numerical_df.iloc[0]).any()]
13 # Lets scale the values for each column from [0,1]
14 # N.B. we dont have any negative values]
15 final_df = numerical_df/numerical_df.max()
16 X = final_df.values
17 # final dataframe has 33 features
18 print("Shape of feature matrix : ",X.shape)
```

Shape of feature matrix : (494021, 33)

```
1 from sklearn.preprocessing import LabelEncoder
2
3 threat_types = df["threat_type"].values
4 encoder = LabelEncoder()
5 # use LabelEncoder to encode the threat types in numeric values
6 y = encoder.fit_transform(threat_types)
7 print("Shape of target vector : ",y.shape)
```

Shape of target vector : (494021,)

Train/Test split

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=42, stratify=y)
3 print("Number of records in training data : ", X_train.shape[0])
4 print("Number of records in test data : ", X_test.shape[0])
5 print("Total distinct number of threat types in training data : ", len(set(y_train)))
6 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

```
1 %%capture
2 import torch
3 import syft as sy
4
5 # Hook PyTorch ie add extra functionalities to support Federated Learning
6 hook = sy.TorchHook(torch)
7 # Sets the seed for generating random numbers.
8 torch.manual_seed(1)
9 # Select CPU computation, in case you want GPU use "gpu" instead
10 device = torch.device("cpu")
11 # Data will be distributed among these VirtualWorkers.
12 # Remote training of the model will happen here.
13 gateway1 = sy.VirtualWorker(hook, id="gateway1")
14 gateway2 = sy.VirtualWorker(hook, id="gateway2")
```

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

Lets set the training parameters

```
1 import numpy as np
2
3 # Number of times we want to iterate over whole training data
4 BATCH_SIZE = 1000
5 EPOCHS = 5
6 LOG_INTERVAL = 120
7 lr = 0.01
8
9 n_feature = X_train.shape[1]
10 n_class = np.unique(y_train).shape[0]
11
12 print("Number of training features : ",n_feature)
13 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

```
14 # Create pytorch tensor from X_train,y_train,X_test,y_test
15 train_inputs = torch.tensor(X_train,dtype=torch.float).tag("#iot", "#network", "#data", "#train")
16 train_labels = torch.tensor(y_train).tag("#iot", "#network", "#target", "#train")
17 test_inputs = torch.tensor(X_test,dtype=torch.float).tag("#iot", "#network", "#data", "#test")
18 test_labels = torch.tensor(y_test).tag("#iot", "#network", "#target", "#test")
19
20 # Send the training and test data to the gateways in equal proportion.
21 train_idx = int(len(train_labels)/2)
22 test_idx = int(len(test_labels)/2)
23 gateway1_train_dataset = sy.BaseDataset(train_inputs[:train_idx], train_labels[:train_idx]).send(gateway1)
24 gateway2_train_dataset = sy.BaseDataset(train_inputs[train_idx:], train_labels[train_idx:]).send(gateway2)
25 gateway1_test_dataset = sy.BaseDataset(test_inputs[:test_idx], test_labels[:test_idx]).send(gateway1)
26 gateway2_test_dataset = sy.BaseDataset(test_inputs[test_idx:], test_labels[test_idx:]).send(gateway2)
27
28 # Create federated datasets, an extension of Pytorch TensorDataset class
29 federated_train_dataset = sy.FederatedDataset([gateway1_train_dataset, gateway2_train_dataset])
30 federated_test_dataset = sy.FederatedDataset([gateway1_test_dataset, gateway2_test_dataset])
31
32 # Create Federated dataloaders, an extension of Pytorch DataLoader class
33 federated_train_loader = sy.FederatedDataLoader(federated_train_dataset, shuffle=True, batch_size=BATCH_SIZE)
34 federated_test_loader = sy.FederatedDataLoader(federated_test_dataset, shuffle=False, batch_size=BATCH_SIZE)
```

Lets define a simple logistic regression model

```
1 import torch.nn as nn
2 class Net(nn.Module):
3     def __init__(self, input_dim, output_dim):
4         """
5         input_dim: number of input features.
6         output_dim: number of labels.
7         """
8         super(Net, self).__init__()
9         self.linear = torch.nn.Linear(input_dim, output_dim)
10    def forward(self, x):
11        outputs = self.linear(x)
12        return outputs
```

It can be any PyTorch DL model

Lets define the training process

```
1 import torch.nn.functional as F
2
3 def train(model, device, federated_train_loader, optimizer, epoch):
4     model.train()
5     # Iterate through each gateway's dataset
6     for batch_idx, (data, target) in enumerate(federated_train_loader):
7         # Send the model to the right gateway
8         model.send(data.location)
9         # Move the data and target labels to the device (cpu/gpu) for computation
10        data, target = data.to(device), target.to(device)
11        # Clear previous gradients (if they exist)
12        optimizer.zero_grad()
13        # Make a prediction
14        output = model(data)
15        # Calculate the cross entropy loss (We are doing classification)
16        loss = F.cross_entropy(output, target)
17        # Calculate the gradients
18        loss.backward()
19        # Update the model weights
20        optimizer.step()
21        # Get the model back from the gateway
22        model.get()
23        if batch_idx!=0 and batch_idx % LOG_INTERVAL == 0:
24            # get the loss back
25            loss = loss.get()
26            print('Train Epoch: {} [{} / {}] ( {:.0f}%) \t Loss: {:.6f}'.format(
27                epoch, batch_idx * BATCH_SIZE, len(federated_train_loader) * BATCH_SIZE,
28                100. * batch_idx / len(federated_train_loader), loss.item()))
```

Lets define the validation process

```
1 import torch.nn.functional as F
2
3 def test(model, device, federated_test_loader):
4     model.eval()
5     correct = 0
6     with torch.no_grad():
7         for batch_idx, (data, target) in enumerate(federated_test_loader):
8             # Send the model to the right gateway
9             model.send(data.location)
10            # Move the data and target labels to the device (cpu/gpu) for computation
11            data, target = data.to(device), target.to(device)
12            # Make a prediction
13            output = model(data)
14            # Get the model back from the gateway
15            model.get()
16            # Calculate the cross entropy loss
17            loss = F.cross_entropy(output, target)
18            # Get the index of the max log-probability
19            pred = output.argmax(1, keepdim=True)
20            # Get the number of instances correctly predicted
21            correct += pred.eq(target.view_as(pred)).sum().get()
22
23    # get the loss back
24    loss = loss.get()
25    print('Test set: Loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
26        loss.item(), correct, len(federated_test_loader.federated_dataset),
27        100. * correct / len(federated_test_loader.federated_dataset)))
```

Lets train the model in federated way

```
1 time
2 import torch.optim as optim
3
4
5 # Initialize the model
6 model = Net(n_feature,n_class)
7
8 #Initialize the SGD optimizer
9 optimizer = optim.SGD(model.parameters(), lr=lr)
10
11 for epoch in range(1, EPOCHS + 1):
12     # Train on the training data in a federated way
13     train(model, device, federated_train_loader, optimizer, epoch)
14     # Check the test accuracy on unseen test data in a federated way
15     test(model, device, federated_test_loader)
```

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

```
Train Epochs: 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 Epochs: 4 [240000/297000 (81%)] Loss: 0.316881
Test set: Loss: 0.2834, Accuracy: 191805/197609 (97%)
```

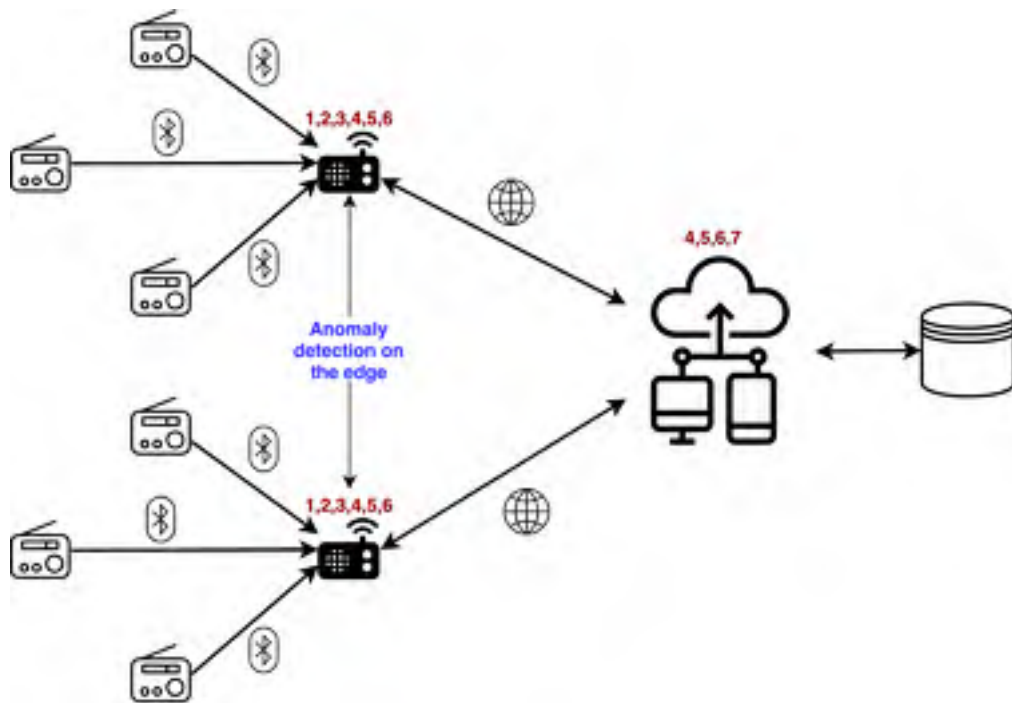
```
Train Epoch: 5 [120000/297000 (40%)] Loss: 0.294926
Train Epochs: 5 [240000/297000 (81%)] Loss: 0.299990
Test set: Loss: 0.2410, Accuracy: 192059/197609 (97%)
```

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

```
1 # Save the model
2 torch.save(model.state_dict(), "binaize-threat-model.pt")
3 # Reload the model in a new model object
4 model_new = Net(n_feature,n_class)
5 model_new.load_state_dict(torch.load("binaize-threat-model.pt"))
6 model_new.eval()
7
8 # Take the 100th record from the test data
9 idx = 100
10 data = test_inputs[idx]
11 pred = model_new(test_inputs[0])
12 pred_label = int(pred.argmax().data.cpu().numpy())
13 pred_threat = encoder.inverse_transform([pred_label])[0]
14 print("Predicted threat type : ", pred_threat)
15 actual_label = int(test_labels[idx].data.cpu().numpy())
16 actual_threat = encoder.inverse_transform([actual_label])[0]
17 print("Actual threat type : ", actual_threat)
```

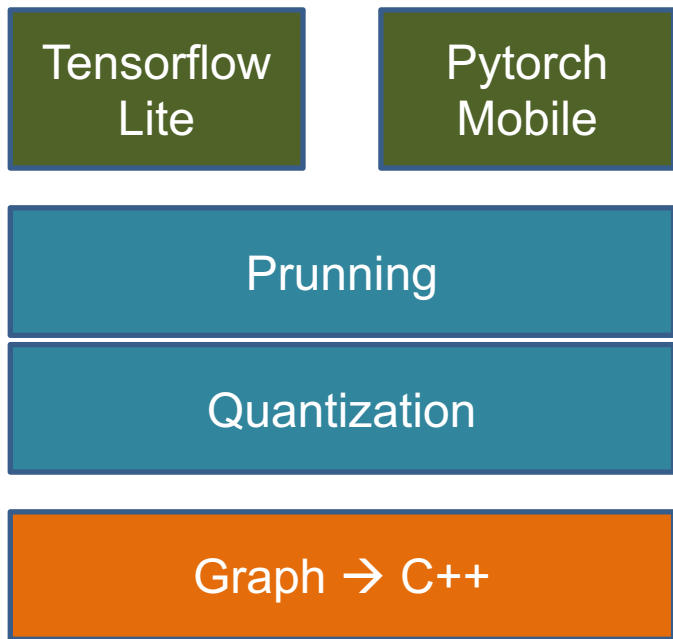
```
Predicted threat type : smurf.
Actual threat type : smurf.
```

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

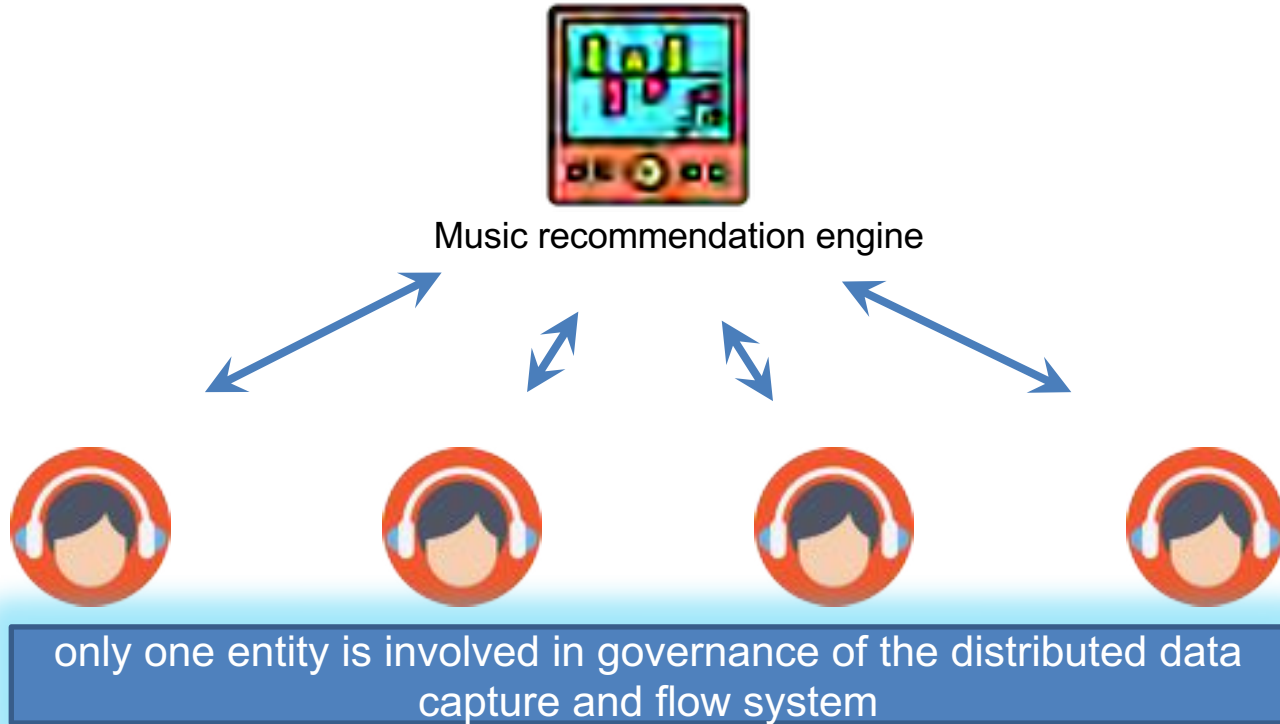


SINGLE PARTY FEDERATED LEARNING

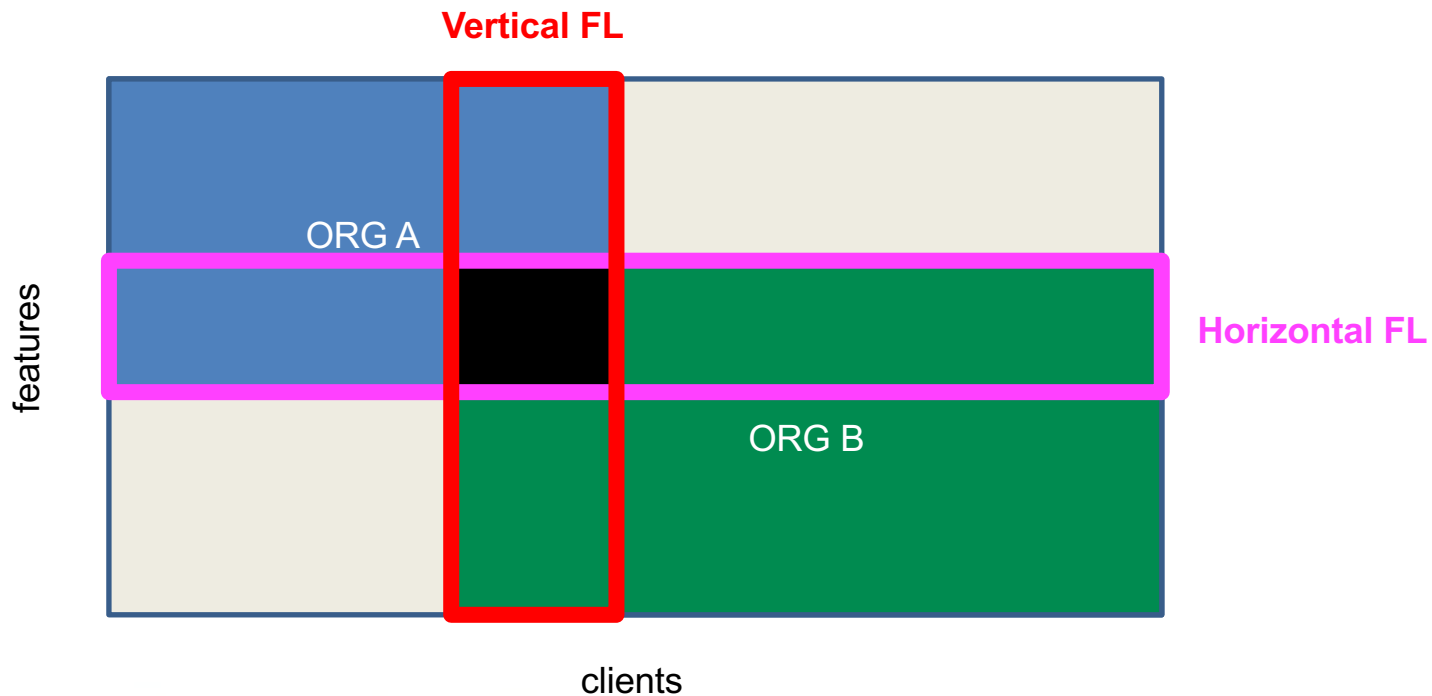


MULTI PARTY FEDERATED LEARNING.

Single Party Federated Learning



Multi Party Federated Learning



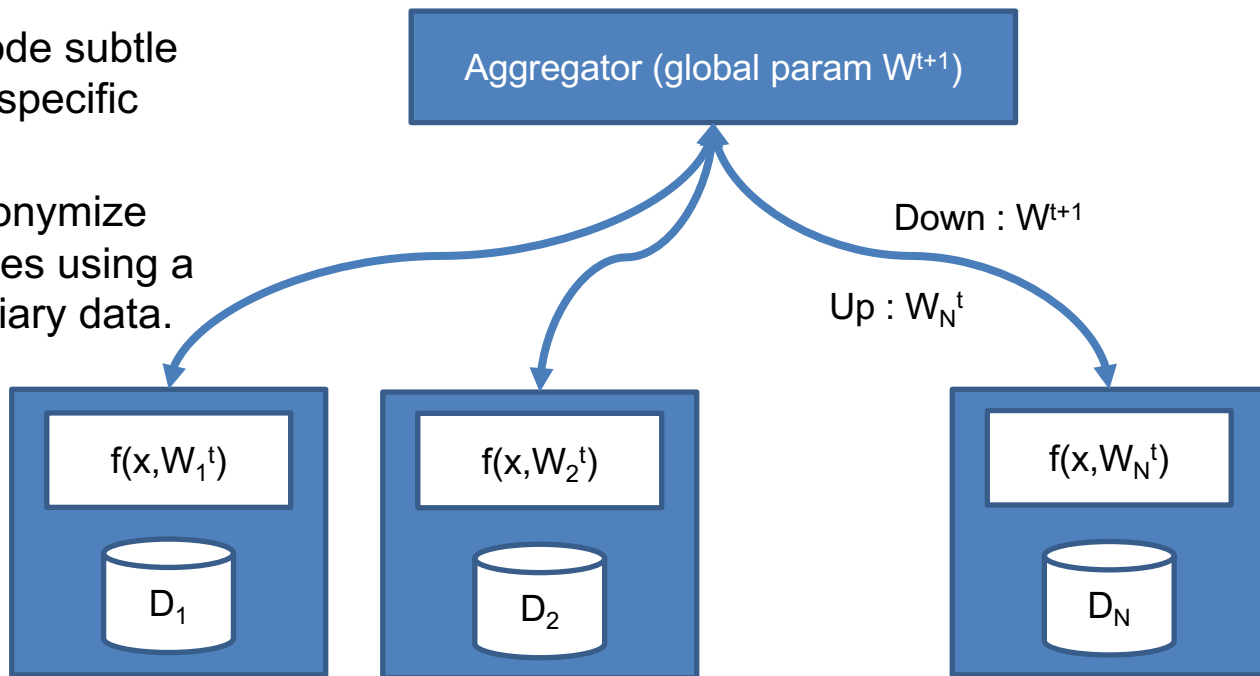
Challenges in Federated Learning

- Inference Attack.
- Model Poisoning.



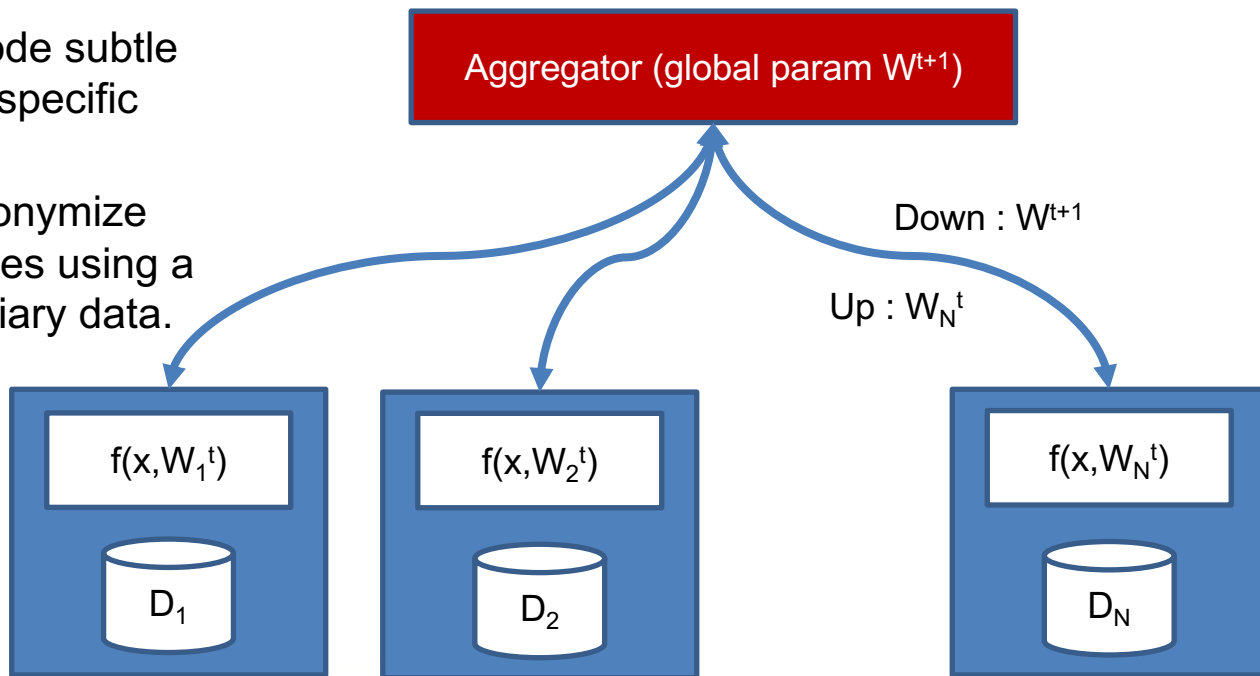
Inference Attack

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



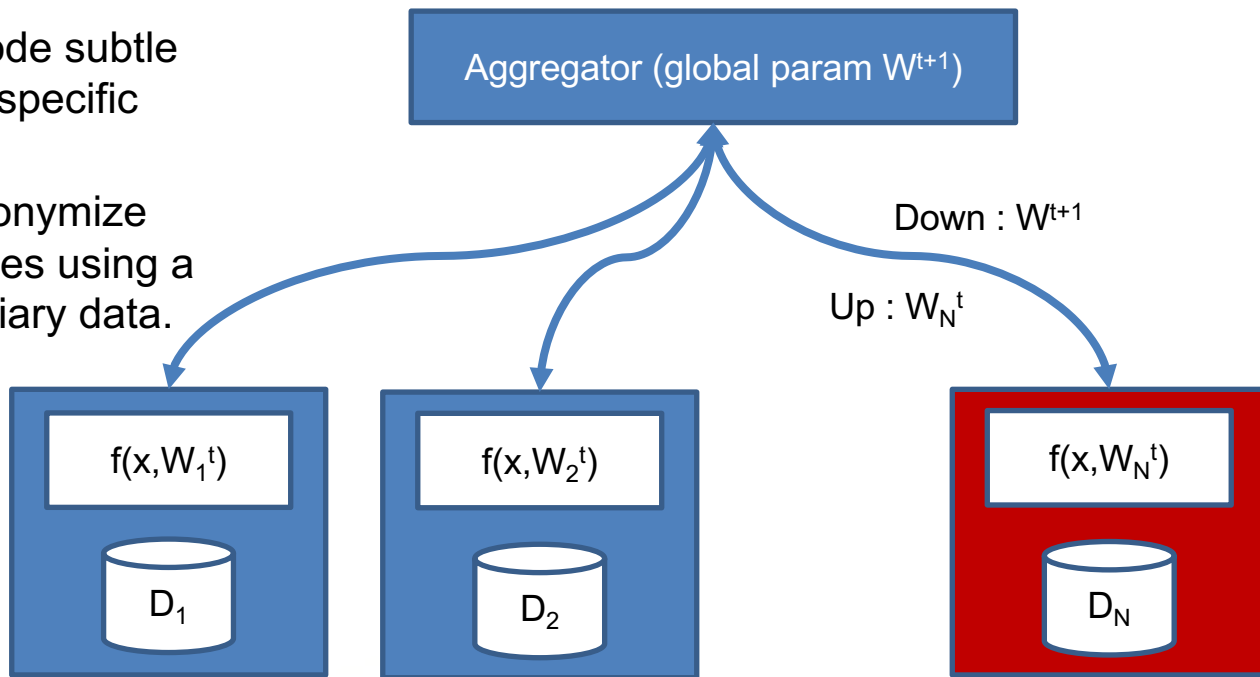
Inference Attack

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



Inference Attack

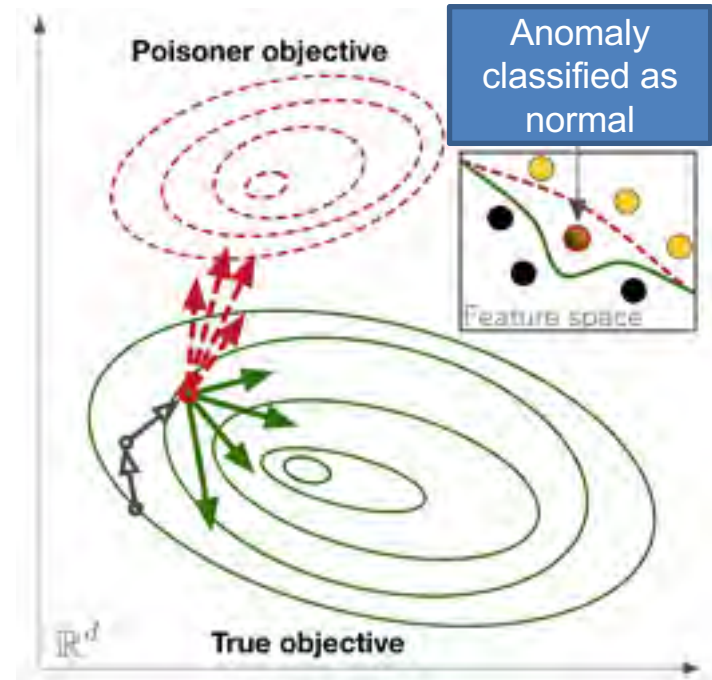
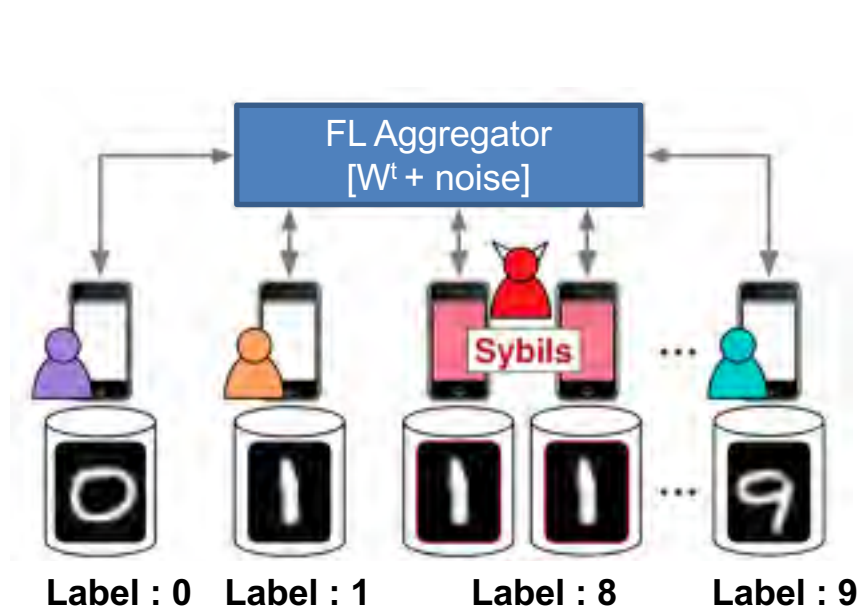
- Model deltas encode subtle variations of user specific information.
- Possible to de-anonymize participating devices using a limited set of auxiliary data.



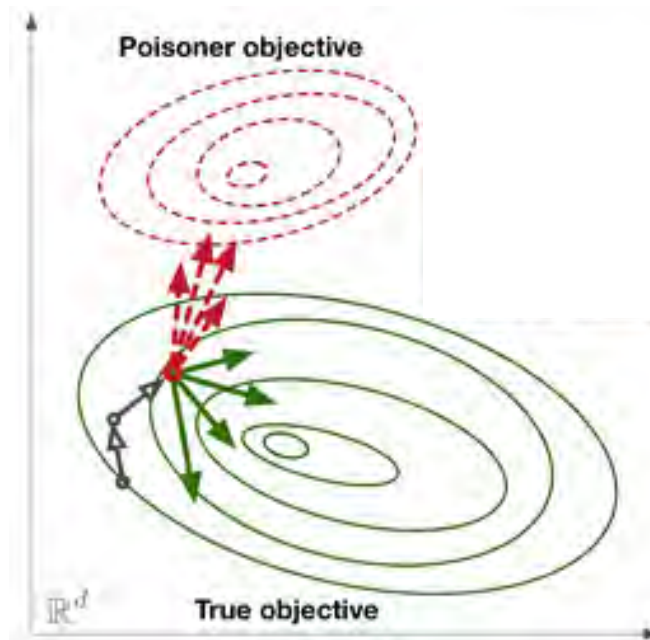
Solution: Differential Privacy



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 Deanonimization 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

Life is Beautiful!!

Tuhin Sharma | Binaize Labs



@tuhinsharma121

Rate today's session

Anomaly detection in smart buildings using federated learning

Tushin Sharma (Birkbeck Labs), Barqawa Subramanian (Birkbeck Labs)
16:00-16:40 Thursday, 17 October 2019
Location: Blenheim Room - Palace Suite
Models and Methods
Secondary topics: Data, Data Networks, Data Quality, Ethics, Security, and Privacy, Machine Learning, Machine Learning tools, Mobile Computers, IoT, Edge

RATE THIS SESSION

Who is this presentation for?

- Data scientists, machine learning engineers, and product managers

Session page on conference website

Anomaly detection in smart buildings using federated learning

Speakers

Tushin Sharma
Chairman & CEO
Birkbeck

Barqawa Subramanian
Operations Director, London
Birkbeck

Blenheim Room - Palace Suite

There's an exponential growth in the number of internet-enabled devices on modern smart buildings. IoT sensors measure temperature, lighting, IP cameras, and more. Tushin Sharma and Barqawa Subramanian explain how they build anomaly-detection models using federated learning – which is privacy preserving and doesn't require data to be moved to the cloud – for data quality and cybersecurity.

RATE THIS SESSION

O'Reilly Events App