FEDERATED LEARNING TUTORIAL

APPFL

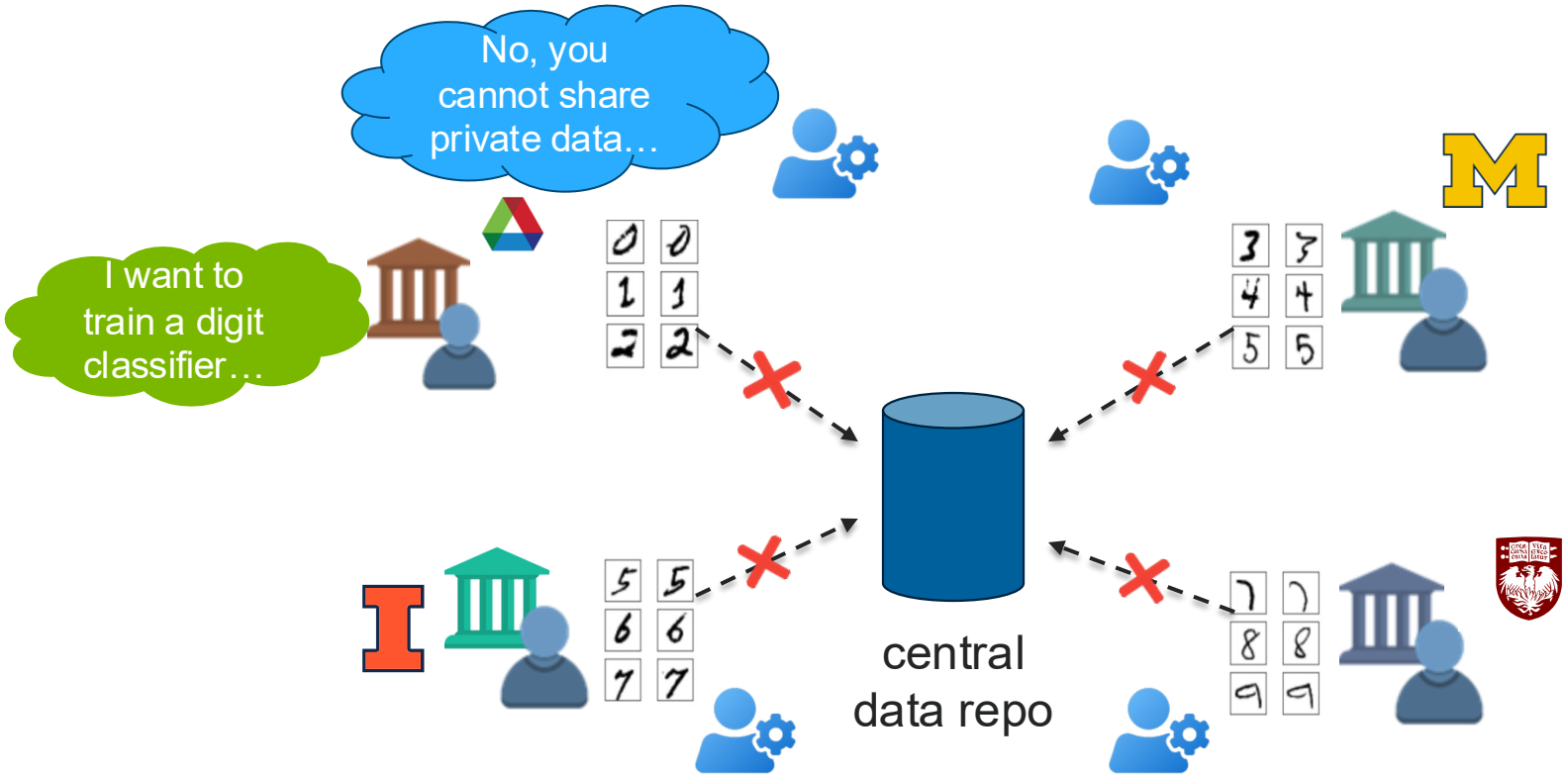
ZILINGHAN LI

Machine Learning Engineer
Data Science and Learning Division,
Argonne National Laboratory
zilinghan.li@anl.gov

U.S. DEPARTMENT OF ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

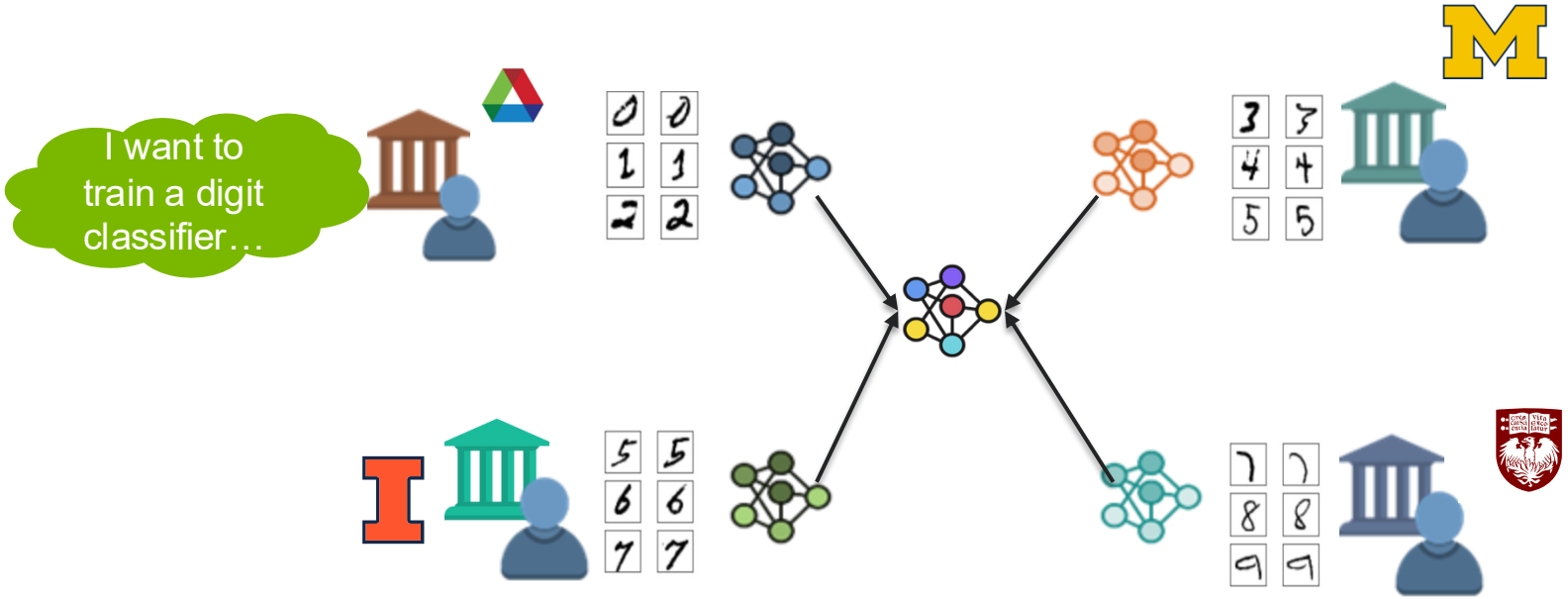
BRIDGE2AI

MOTIVATION FOR FL – AN "EXTREME" CASE



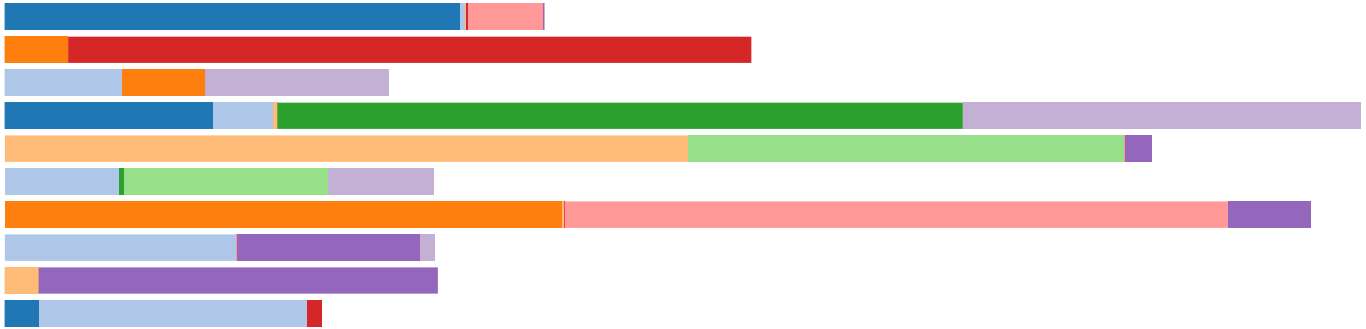
MOTIVATION FOR FL – AN "EXTREME" CASE

Share the model trained on local data instead of sharing the data directly



These four clients create a loose federation, and we call it federated learning (FL).

MOTIVATION FOR FL – AN "EXTREME" CASE



MNIST dataset partition: number of data samples for different labels in each client's local datasets.

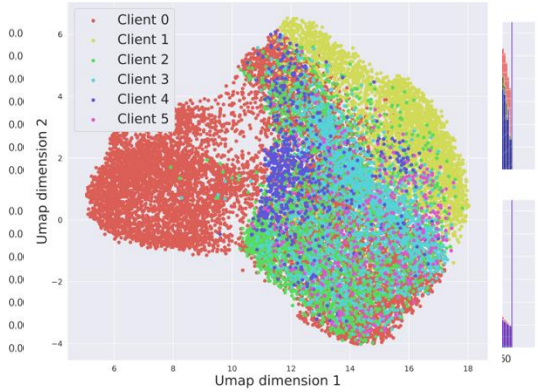
Training a CNN by sharing model parameters trained locally (i.e. using federated learning) can get **>95%** accuracy!

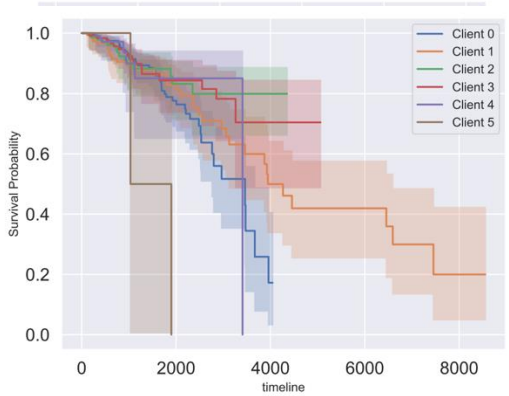
MOTIVATION – ANY MORE REAL REASONS?

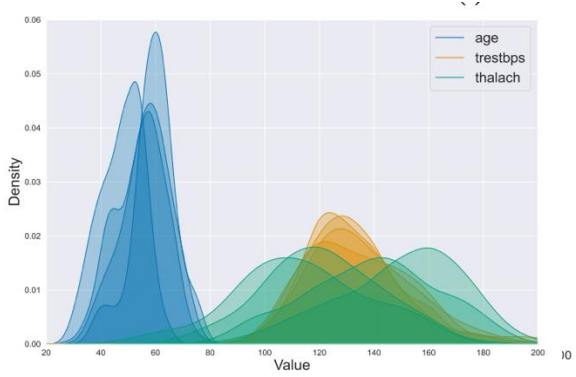
This example is trivial and does not sound like something will happen in real life…

So, again, why federated learning?

In real world, it is common that the data at different data silos have **different distributions**, especially in domains such as biomedicine where data privacy is important. The biomedicine data can be very heterogeneous due to **different population distributions**, **device variations**, **data collection protocol differences**, etc.







Ogier du Terrail, Jean, Samy-Safwan Ayed, Edwige Cyffers, Felix Grimberg, Chaoyang He, Regis Loeb, Paul Mangold et al. "Flamby: Datasets and benchmarks for cross-silo federated learning in realistic healthcare settings." *Advances in Neural Information Processing Systems* 35 (2022): 5315-5334.

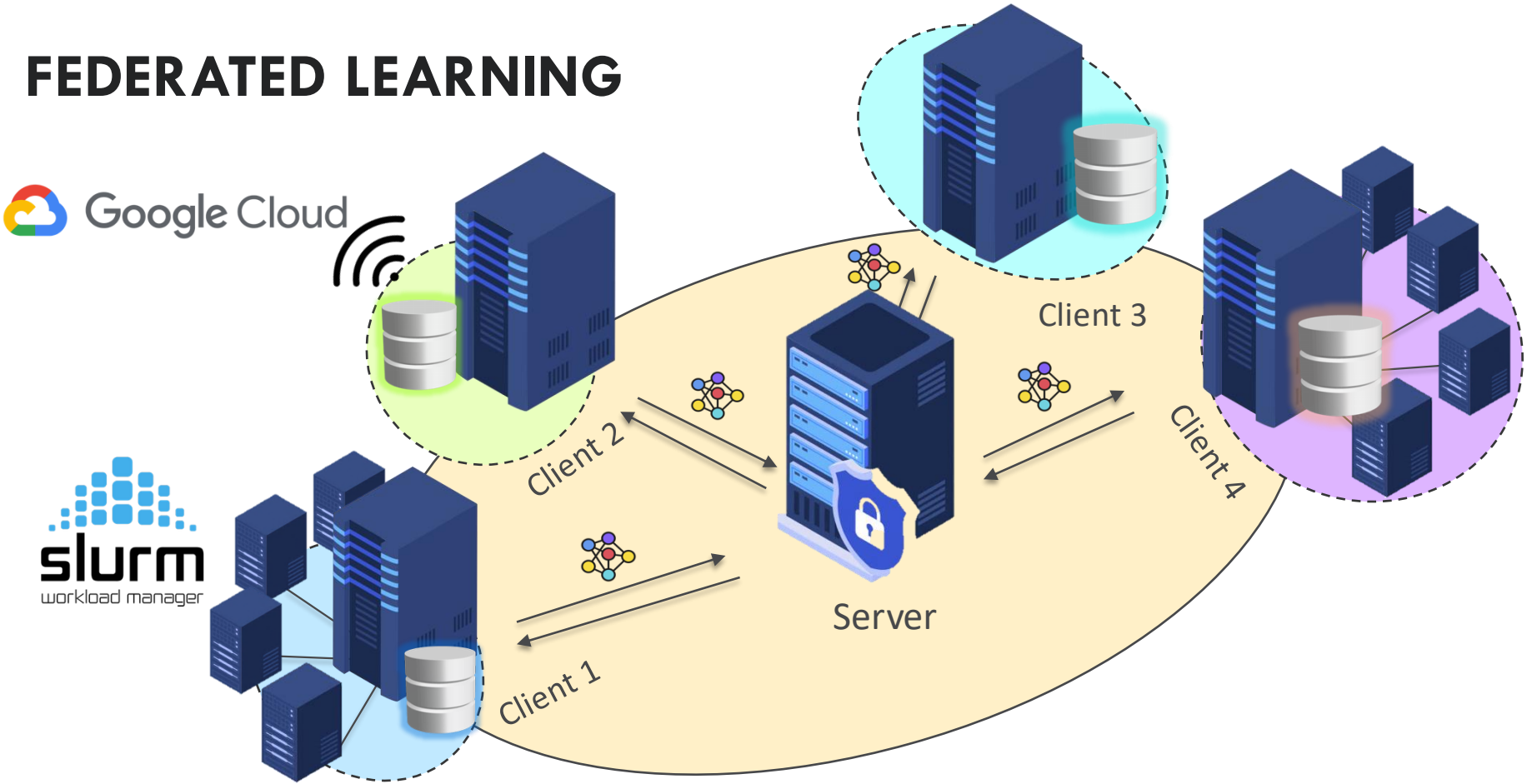
MOTIVATION – ANY MORE REAL REASONS?

Privacy Concerns in Biomedical Data

In biomedical research, access to many types of data, such as identifiable electronic health records (HER), medical images, and electrocardiogram (ECG) readings, is strictly regulated by law like HIPAA in the United States and GDPR in the European Union. Data access committees and institutional review boards (IRB) manually manage access controls to ensure that research is ethical and that the privacy of research subjects is safeguarded.

Historically, these necessary, but cumbersome, access restrictions have had the undesirable effect of siloing data at the host institution which in turn has stymied collaborative research.

FEDERATED LEARNING



HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

From the previous tutorial, we saw that the AI-READI dataset contains images from four different types of devices .

topcon_triton

optomed_aurora

topcon_maestro2

icare_eidon

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Now Let's consider we have 2 hospitals with data distribution as below:

Hospital 1 (Client 1) has only the following device types:

topcon_triton

optomed_aurora

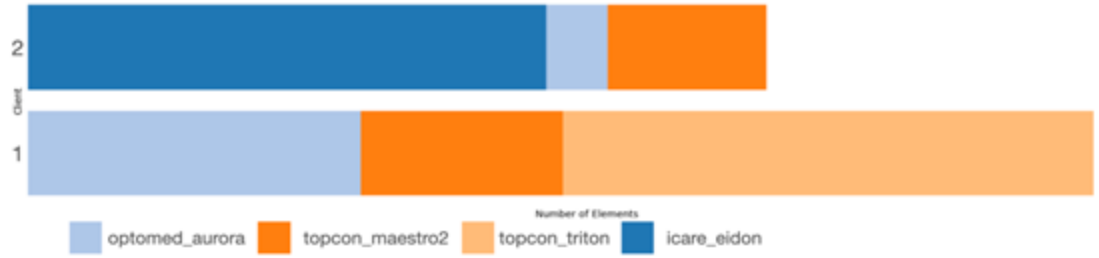
topcon_maestro2

Hospital 2 (Client 2) has only the following device types:

icare_eidon

optomed_aurora

topcon_maestro2



HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

If we train our model solely on data from one hospital, which includes only a subset of device types.

The model may not generalize well to unseen devices.

For example, if a hospital later adds a new device type that wasn't present in its training data (but exists in another hospital), the model may fail to properly recognize or classify its outputs.

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Now, Let's get everyone divided into groups of 3.

Each person in the group can open one of the following three notebooks.

APPFL_Server_AI_READI.ipynb

APPFL_Client1_AIREADI.ipynb

APPFL_Client2_AIREADI.ipynb

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Step 1: Train a model using each client's local data.

APPFL_Client1_AIREADI.ipynb

Train a model independently

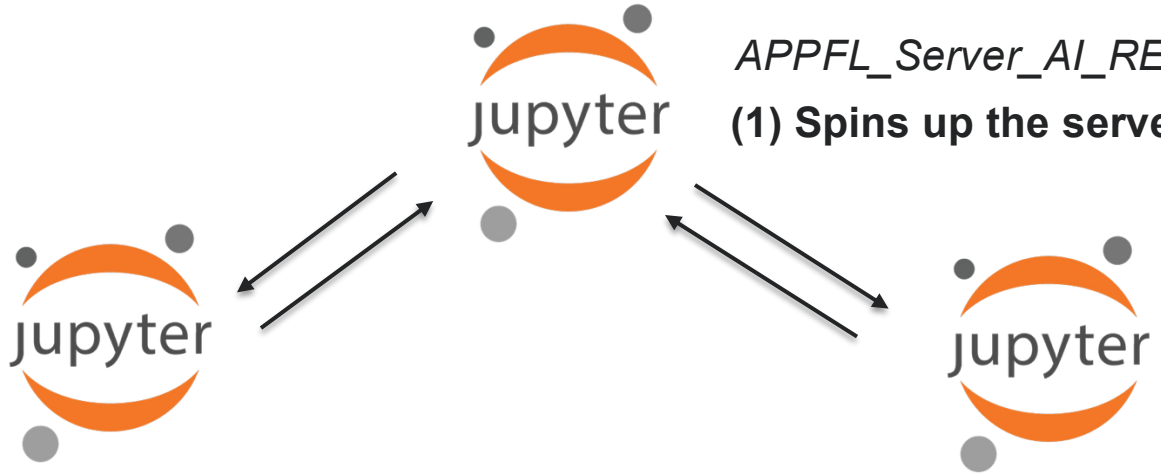
APPFL_Client2_AIREADI.ipynb

Train a model independently

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Step 2: Federated training of a model with two clients



APPFL_Server_AI_READI.ipynb

(1) Spins up the server

APPFL_Client1_AIREADI.ipynb

(2) Train model locally

(3) Send aggregation request

APPFL_Client2_AIREADI.ipynb

(2) Train model locally

(3) Send aggregation request

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

For the two clients, open your notebook, and let's first train a classification model using your **"local data" only**.

In client notebooks lets run the first cell.

Load training configuration to train model on client1's data

```
import os
import argparse
from omegaconf import OmegaConf
from appfl.agent import ClientAgent
import torchvision.models as models
import torch.nn as nn

os.chdir("/home/jupyter/lab/APPFL/examples")
client_agent_config = OmegaConf.load("./resources/configs/aireadi/client_1_ec2_single.yaml")
client_agent_config.data_configs.dataset_kwargs.visualization = False
print("==========Client Configuration==========")
print(OmegaConf.to_yaml(client_agent_config))
print("========================================")
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

The training configurations will be printed.

```
==========Client Configuration==========
client_id: Client1
train_configs:
  device: cuda
  logging_output_dirname: ./output
  logging_output_filename: result
  trainer: VanillaTrainer
  mode: epoch
  num_local_epochs: 1
  optim: Adam
  optim_args:
    lr: 7.0e-06
    weight_decay: 0.001
  loss_fn_path: ./resources/loss/celoss.py
  loss_fn_name: CELoss
  do_validation: false
  do_pre_validation: true
  metric_path: ./resources/metric/acc.py
  metric_name: accuracy
  use_dp: false
  train_batch_size: 64
  val_batch_size: 64
  train_data_shuffle: true
  val_data_shuffle: false
  num_workers: 4
model_configs:
  model_path: ./resources/model/resnet_aireadi.py
  model_name: ResNet50
  model_kwargs:
    num_classes: 4
data_configs:
  dataset_path: ./resources/dataset/ai_readi_dataset.py
  dataset_name: get_ai_readi
  dataset_kwargs:
    num_clients: 2
    client_id: 0
    partition_strategy: class_noniid
    sampling_factor: 0.5
    visualization: false
    output_dirname: ./output
    output_filename: visualization.pdf
    data_path: full/
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Run Cell 2 on both clients' notebook to launch local training for 10 epochs.

Train model on Client1's data

```
client_agent = ClientAgent(client_agent_config=client_agent_config)
model = models.resnet50(weights=None)
model.fc = nn.Linear(model.fc.in_features, 4)
client_agent.load_parameters(model.state_dict())
num_of_epoch = 10
for i in range(0, num_of_epoch):
    client_agent.train()
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

We can see that the accuracy stays around 70%, which is not good as we expect.

```
appfl: ✅[2025-05-02 23:05:37,285 Client1]: Logging to ./output/result_Client1_2025-05-02-23-05-37.txt
appfl: ✅[2025-05-02 23:05:38,622 Client1]:      Round      Epoch   Pre Val?       Time Train Loss Train Accuracy   Val Loss Val Accuracy
appfl: ✅[2025-05-02 23:05:41,219 Client1]:          0          0          Y                                          24.6032      22.0874
appfl: ✅[2025-05-02 23:05:56,509 Client1]:          0          0          N    15.2878     0.8924       68.7624    24.6032      22.0874
appfl: ✅[2025-05-02 23:05:58,932 Client1]:          1          0          Y                                           1.4531      50.5461
appfl: ✅[2025-05-02 23:06:14,109 Client1]:          1          0          N    15.1753     0.4828       82.8811     1.4531      50.5461
appfl: ✅[2025-05-02 23:06:16,551 Client1]:          2          0          Y                                           1.6614      56.7961
appfl: ✅[2025-05-02 23:06:31,746 Client1]:          2          0          N    15.1937     0.3170       87.8668     1.6614      56.7961
appfl: ✅[2025-05-02 23:06:34,181 Client1]:          3          0          Y                                           1.9630      59.0413
appfl: ✅[2025-05-02 23:06:49,452 Client1]:          3          0          N    15.2692     0.1947       93.6245     1.9630      59.0413
appfl: ✅[2025-05-02 23:06:51,888 Client1]:          4          0          Y                                           2.1677      65.7160
appfl: ✅[2025-05-02 23:07:07,151 Client1]:          4          0          N    15.2621     0.0786       98.3455     2.1677      65.7160
appfl: ✅[2025-05-02 23:07:09,533 Client1]:          5          0          Y                                           2.3266      68.5680
appfl: ✅[2025-05-02 23:07:24,758 Client1]:          5          0          N    15.2229     0.0316       99.5147     2.3266      68.5680
appfl: ✅[2025-05-02 23:07:27,162 Client1]:          6          0          Y                                           2.6801      69.5388
appfl: ✅[2025-05-02 23:07:42,471 Client1]:          6          0          N    15.3074     0.0161       99.7573     2.6801      69.5388
appfl: ✅[2025-05-02 23:07:44,869 Client1]:          7          0          Y                                           2.8296      69.4175
appfl: ✅[2025-05-02 23:08:00,120 Client1]:          7          0          N    15.2492     0.0096       99.8897     2.8296      69.4175
appfl: ✅[2025-05-02 23:08:02,529 Client1]:          8          0          Y                                           3.0046      70.1456
appfl: ✅[2025-05-02 23:08:17,772 Client1]:          8          0          N    15.2417     0.0056       99.9779     3.0046      70.1456
appfl: ✅[2025-05-02 23:08:20,246 Client1]:          9          0          Y                                           2.9618      68.2039
appfl: ✅[2025-05-02 23:08:35,492 Client1]:          9          0          N    15.2444     0.0039       99.9559     2.9618      68.2039
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Now, let's try to leverage federated learning to have two hospitals to collaboratively train a more robust and generalized model that can predict all four device types.

In this tutorial, we will leverage the Advanced Privacy-Preserving Federated Learning (APPFL) an open-source framework to launch a federated learning client for running a federated learning experiment with two clients and one central server.

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Let’s leave our client notebooks now and move on to the server notebook.

Now the member of the group who is designated as server will run their notebook.

Two clients need to keep their notebooks but don’t run it yet. – We need to first spin up the server.

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

We will run cells of server notebook

First, since we have pre-installed the APPFL module for you we don't have to run anything.

Second, We just set the working directory for our code.

2. Set working directory

```
import os
import warnings

warnings.filterwarnings('ignore')
# setting current working directory
target_directory = '/home/jupyter/lab/APPFL/examples'

os.chdir(target_directory)
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Now, let's launch the server agent with configurations.

3. Create federated learning server agent from configurations

Here we print the configurations needed to create a federated learning server. Specifically, it is composed of two main parts: `client_configs` and `server_configs`.

The `client_configs` mainly contains **general** configurations that should be the same among all cleints (e.g., training configurations, model architectures). Those configurations will be shared will all the clients at the beginning of experiments. This can avoid having each client to set the same configurations themselves separately. In this example, `client_configs` has two main components:

- `client_configs.train_configs`: This component contains configruations related to client's local training, such as the trainer to use, loss function, validation function, etc.
- `client_configs.model_configs`: This provides the path which defines the python function to load the architecture of the model to be trained.

The `server_configs` contains configurations that are specific to the federated learning server, such as the number of global epochs (total communication rounds between server and clients), and aggregators to use, etc.

If you are interested, you can find more detailed explanations for the meanings of different configuration fields at APPFL's official document.

```
from omegaconf import OmegaConf
from appfl.agent import ServerAgent

server_agent_config = OmegaConf.load("./resources/configs/aireadi/server_fedavg_ec2.yaml")
server_agent_config.client_configs.train_configs.train_batch_size = 64
server_agent_config.client_configs.train_configs.val_batch_size = 64
server_agent_config.client_configs.train_configs.num_workers = os.cpu_count()
# server_agent_config.server_configs.num_global_epochs = 8
print("==========Server Configuration==========")
print(OmegaConf.to_yaml(server_agent_config))
print("========================================")
server_agent = ServerAgent(server_agent_config=server_agent_config)
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Training settings

- Model: ResNet50
- Optimizer: Adam
- Learning Rate: 7e-6
- Weight Decay: 1e-3
- Loss Function: Cross-Entropy

Federated Learning settings

- Number of clients: 2
- Epoch per client: 25
- Federated Averaging Aggregator training with synchronous model update
- gRPC for communication between the client and server sending requests

```
==========Server Configuration==========
client_configs:
  train_configs:
    trainer: VanillaTrainer
    mode: epoch
    num_local_epochs: 1
    optim: Adam
    optim_args:
      lr: 7.0e-06
      weight_decay: 0.001
    loss_fn_path: ./resources/loss/celoss.py
    loss_fn_name: CELoss
    do_validation: false
    do_pre_validation: true
    metric_path: ./resources/metric/acc.py
    metric_name: accuracy
    use_dp: false
    train_batch_size: 64
    val_batch_size: 64
    train_data_shuffle: true
    val_data_shuffle: false
    num_workers: 4
  model_configs:
    model_path: ./resources/model/resnet_aireadi.py
    model_name: ResNet50
    model_kwargs:
      num_classes: 4
server_configs:
  num_clients: 2
  scheduler: SyncScheduler
  scheduler_kwargs:
    same_init_model: true
  aggregator: FedAvgAggregator
  aggregator_kwargs:
    client_weights_mode: equal
  device: cpu
  num_global_epochs: 25
  logging_output_dirname: ./output
  logging_output_filename: result
  comm_configs:
    grpc_configs:
      server_uri: 0.0.0.0:50051
      max_message_size: 1048576
      use_ssl: false

=========================================
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Now we need to define a server communicator to facilitate the communication between the server and two other clients.

In this step, we create a server communicator that facilitates communication between the server and its clients using gRPC. It provides a reliable medium for the server and clients to exchange information efficiently and asynchronously.

The following code block will print out a link that you need to copy to your two client notebooks.

```
import socket
from appfl.comm.grpc import GRPCServerCommunicator

# Start the gRPC server communicator
communicator = GRPCServerCommunicator(
    server_agent,
    logger=server_agent.logger,
    **server_agent_config.server_configs.comm_configs.grpc_configs,
)

# Getting the private IP address of the server
hostname = socket.gethostname()
private_ip = socket.gethostbyname(hostname)
print(f"\n\n✅ Replace server_uri with below in client notebook: {private_ip}:5005
```

Share the output URI from this cell to the other members of the group that have client notebooks open.

```
✅ Replace server_uri with below in client notebook: 172.31.66.136:50051
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Finally, run the last cell to start serving.

Finally, you can use the communicator to start serving the FL server, which will handle different requests from the two clients to finish the FL experiments.

The client training results are logged in real time as a dictionary containing metrics such as `pre_val_accuracy` and `pre_val_loss` (accuracy and loss before each client's local training round) and `val_accuracy` and `val_loss` (accuracy and loss afterward). You will observe an increase in the model accuracy as the training proceeds.

```
from appfl.comm.grpc import serve

serve(
    communicator,
    **server_agent_config.server_configs.comm_configs.grpc_configs,
)
```

Hooray! Your server is up and running now. Now we move on to Client Notebooks

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Now let's go back to client notebooks

First two steps are same as server we will install APPFL package and then set the working directory.

2. Set working directory

```
import os
import warnings

warnings.filterwarnings('ignore')
# setting the current working directory
target_directory = '/home/jupyter/lab/APPFL/examples'

os.chdir(target_directory)
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

3. Create federated learning client agent from configurations

We need to update the `server_uri` to the URL obtained from the **server** notebook.

Steps:

- Obtain the `server` URL:
  - For example, the server URL from the server notebook is `172.31.79.131:50051`.
- Update the code:
  - Replace the placeholder in the following line with the actual `server` URL obtained.
  - Update the code as follows:

```
client_agent_config.comm_configs.grpc_configs["server_uri"] = "172.31.79.131:50051"
```

In the client configurations, it has four main parts:

- `client_id`: A unique identifier for the client
- `train_configs`: Client-specific training related configruations, such as the device and logging directories
- `data_configs`: Information about the dataloader file that can create a PyTorch dataset for the AI-READI data
- `comm_configs`: Information needed to connect to the server notebook

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Remember to replace ***server_uri*** with the output from **server notebook** before running this cell

```
from omegaconf import OmegaConf
from appfl.agent import ClientAgent
client_agent_config = OmegaConf.load("./resources/configs/aireadi/client_1_ec2.yaml")

## update URL here from server notebook
client_agent_config.comm_configs.grpc_configs["server_uri"] = "172.31.79.131:50051" #
print("==========Client Configuration==========")
print(OmegaConf.to_yaml(client_agent_config))
print("========================================")
client_agent = ClientAgent(client_agent_config=client_agent_config)
```

If you forgot, it's ok you can put the server url and re-run the cell

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Client configurations will be printed.

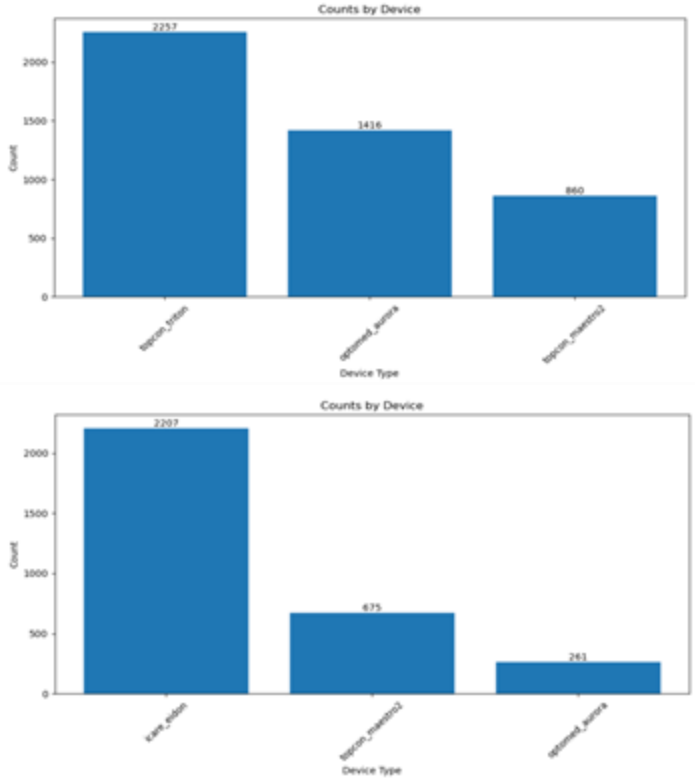
```
==========Client Configuration==========
client_id: Client1
train_configs:
  device: cuda
  logging_output_dirname: ./output
  logging_output_filename: result
data_configs:
  dataset_path: ./resources/dataset/ai_readi_dataset.py
  dataset_name: get_ai_readi
  dataset_kwargs:
    num_clients: 2
    client_id: 0
    partition_strategy: class_noniid
    sampling_factor: 0.5
    visualization: true
    output_dirname: ./output
    output_filename: visualization.pdf
    data_path: full/
comm_configs:
  grpc_configs:
    server_uri: 172.31.79.131:50051
    max_message_size: 1048576
    use_ssl: false
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Now, let's visualize the data distribution for each client before starting FL process.

```
%matplotlib inline

import matplotlib.pyplot as plt

train_distribution = client_agent.train_dataset.label_counts()

# Plotting
plt.figure(figsize=(10, 6))
bars = plt.bar(train_distribution.keys(), train_distribution.values())

plt.xlabel('Device Type')
plt.ylabel('Count')
plt.title('Counts by Device')
plt.xticks(rotation=45)
plt.tight_layout()

# Annotate each bar with its value
for bar in bars:
    height = bar.get_height()
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        height,
        f'{int(height)}',
        ha='center',
        va='bottom'
    )

plt.show()
```



HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Create client communicator to send various requests to FL server via gRPC.

5. Create Client Communicator

Now, we create a grpc client communicator for sending various requests to the server.

```
from appfl.comm.grpc import GRPCClientCommunicator

client_communicator = GRPCClientCommunicator(
    client_id=client_agent.get_id(),
    **client_agent_config.comm_configs.grpc_configs,
)
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Start the training loop.

In this main training loop, there are four main types of request to send to the server:

(1) *get_configuration()*: Get general client configurations for local training

(2) *get_global_model(init_model=True)*: Get the initial global model for training

(3) *update_global_model()*: Send the trained local model to update the global model, and get the updated model back for further local training

(4) *invoke_custom_action(action="close_connection")*: Close the connection with the server

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

```
state = init_plots()

state["h_loss"] = display(state["fig_loss"], display_id=True)
state["h_acc"]  = display(state["fig_acc"], display_id=True)
state["h_log"]  = display(HTML(""), display_id="log")

# Get general client configurations
client_config = client_communicator.get_configuration()
client_agent.load_config(client_config)

# Get initial global model parameters
init_global_model = client_communicator.get_global_model(init_model=True)
client_agent.load_parameters(init_global_model)

# Start local training loop
while True:
    buf = io.StringIO()
    with redirect_stdout(buf):
        client_agent.train()
    local_model = client_agent.get_parameters()
    if isinstance(local_model, tuple):
        local_model, metadata = local_model[0], local_model[1]
    else:
        metadata = {}
    flush_log(state, buf.getvalue())
    refresh_plots(state, metadata["round"], metadata["pre_val_loss"], metadata["pre_val_accuracy"])
    new_global_model, metadata = client_communicator.update_global_model(
        local_model, **metadata
    )
    if metadata["status"] == "DONE":
        break
    client_agent.load_parameters(new_global_model)

# Close connection
client_communicator.invoke_custom_action(action="close_connection")
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Client notebook's training outputs.

	Round	Epoch	Pre Val?	Time	Train Loss	Train Accuracy	Val Loss	Val Accuracy
appfl: ✅[2025-04-30 20:26:21,819 Client2]:								
appfl: ✅[2025-04-30 20:26:24,272 Client2]:	0	0	Y				22.0451	22.0874
appfl: ✅[2025-04-30 20:26:35,114 Client2]:	0	0	N	10.8400	0.8799	73.8466	22.0451	22.0874
appfl: ✅[2025-04-30 20:26:49,641 Client2]:	1	0	Y				1.1754	50.3641
appfl: ✅[2025-04-30 20:27:00,458 Client2]:	1	0	N	10.8152	0.6860	83.5826	1.1754	50.3641
appfl: ✅[2025-04-30 20:27:13,596 Client2]:	2	0	Y				0.8685	74.9393
appfl: ✅[2025-04-30 20:27:24,394 Client2]:	2	0	N	10.7955	0.5339	88.7369	0.8685	74.9393
appfl: ✅[2025-04-30 20:27:38,423 Client2]:	3	0	Y				0.6852	74.5146
appfl: ✅[2025-04-30 20:27:49,263 Client2]:	3	0	N	10.8382	0.4048	92.0140	0.6852	74.5146
appfl: ✅[2025-04-30 20:28:02,041 Client2]:	4	0	Y				0.5818	73.1189
appfl: ✅[2025-04-30 20:28:12,850 Client2]:	4	0	N	10.8069	0.2821	95.1002	0.5818	73.1189
appfl: ✅[2025-04-30 20:28:26,568 Client2]:	5	0	Y				0.5402	69.8422
appfl: ✅[2025-04-30 20:28:37,344 Client2]:	5	0	N	10.7748	0.2003	96.5002	0.5402	69.8422
appfl: ✅[2025-04-30 20:28:49,997 Client2]:	6	0	Y				0.5539	67.1723
appfl: ✅[2025-04-30 20:29:00,807 Client2]:	6	0	N	10.8081	0.1337	98.0274	0.5539	67.1723
appfl: ✅[2025-04-30 20:29:14,188 Client2]:	7	0	Y				0.4682	72.4515
appfl: ✅[2025-04-30 20:29:25,718 Client2]:	7	0	N	11.5283	0.0911	99.1091	0.4682	72.4515
appfl: ✅[2025-04-30 20:29:37,678 Client2]:	8	0	Y				0.4382	75.7282
appfl: ✅[2025-04-30 20:29:48,580 Client2]:	8	0	N	10.9002	0.0661	99.3318	0.4382	75.7282
appfl: ✅[2025-04-30 20:30:01,981 Client2]:	9	0	Y				0.4423	76.3956
appfl: ✅[2025-04-30 20:30:12,822 Client2]:	9	0	N	10.8396	0.0459	99.6500	0.4423	76.3956
appfl: ✅[2025-04-30 20:30:25,817 Client2]:	10	0	Y				0.3136	84.2840
appfl: ✅[2025-04-30 20:30:36,661 Client2]:	10	0	N	10.8421	0.0354	99.8409	0.3136	84.2840

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

Server notebook's training outputs.

```
appfl: ✅[2025-04-29 22:04:49,647 server]: Received GetConfiguration request from client Client1
appfl: ✅[2025-04-29 22:04:49,958 server]: Received GetGlobalModel request from client Client1
appfl: ✅[2025-04-29 22:04:56,256 server]: Received GetConfiguration request from client Client2
appfl: ✅[2025-04-29 22:04:56,673 server]: Received GetGlobalModel request from client Client2
appfl: ✅[2025-04-29 22:05:46,331 server]: Received UpdateGlobalModel request from client Client1
appfl: ✅[2025-04-29 22:05:46,425 server]: Received the following meta data from Client1:
{'pre_val_accuracy': 22.0873786407767,
 'pre_val_loss': 22.045146501981296,
 'round': 1,
 'val_accuracy': [89.62378640776699],
 'val_loss': [0.3039210931612895]}
appfl: ✅[2025-04-29 22:05:48,362 server]: Received UpdateGlobalModel request from client Client2
appfl: ✅[2025-04-29 22:05:48,418 server]: Received the following meta data from Client2:
{'pre_val_accuracy': 22.0873786407767,
 'pre_val_loss': 22.045146501981296,
 'round': 1,
 'val_accuracy': [86.22572815533981],
 'val_loss': [0.3992317697176567]}
appfl: ✅[2025-04-29 22:06:12,116 server]: Received UpdateGlobalModel request from client Client1
appfl: ✅[2025-04-29 22:06:12,206 server]: Received the following meta data from Client1:
{'pre_val_accuracy': 53.45873786407767,
 'pre_val_loss': 1.2737316833092616,
 'round': 2,
 'val_accuracy': [77.18446601941747],
 'val_loss': [0.6979049960008035]}
```

HANDS ON TUTORIAL

A Case Study on the AI-READI Dataset

From the results, we can see that the accuracy goes more than 90 and the model is able to identify the device with much higher accuracy compared to locally trained models.

You can go check out all the notebooks from today here:
https://github.com/AdioosinUIUC/aireadi-course