



جامعة الملك عبد الله  
للعلوم والتكنولوجيا  
King Abdullah University of  
Science and Technology



# FL\_PyTorch

Optimization Research Simulator  
for Federated Learning

Konstantin Burlachenko



Konstantin Burlachenko  
**PhD student KAUST**  
[konstantin.burlachenko@kaust.edu.sa](mailto:konstantin.burlachenko@kaust.edu.sa)

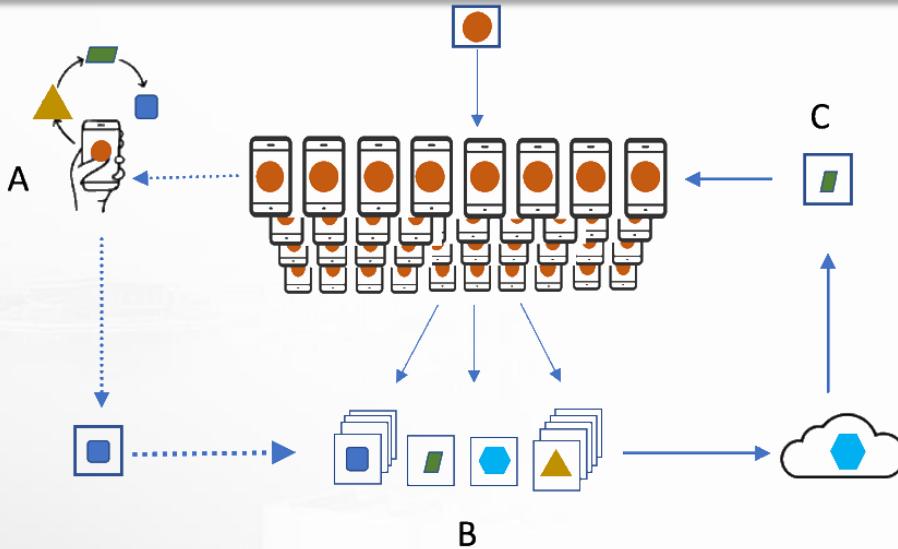


Samuel Horváth  
**PhD student KAUST**  
[samuel.horvath@kaust.edu.sa](mailto:samuel.horvath@kaust.edu.sa)



Peter Richtárik  
**Professor of Computer Science KAUST**  
[peter.richtarik@kaust.edu.sa](mailto:peter.richtarik@kaust.edu.sa)

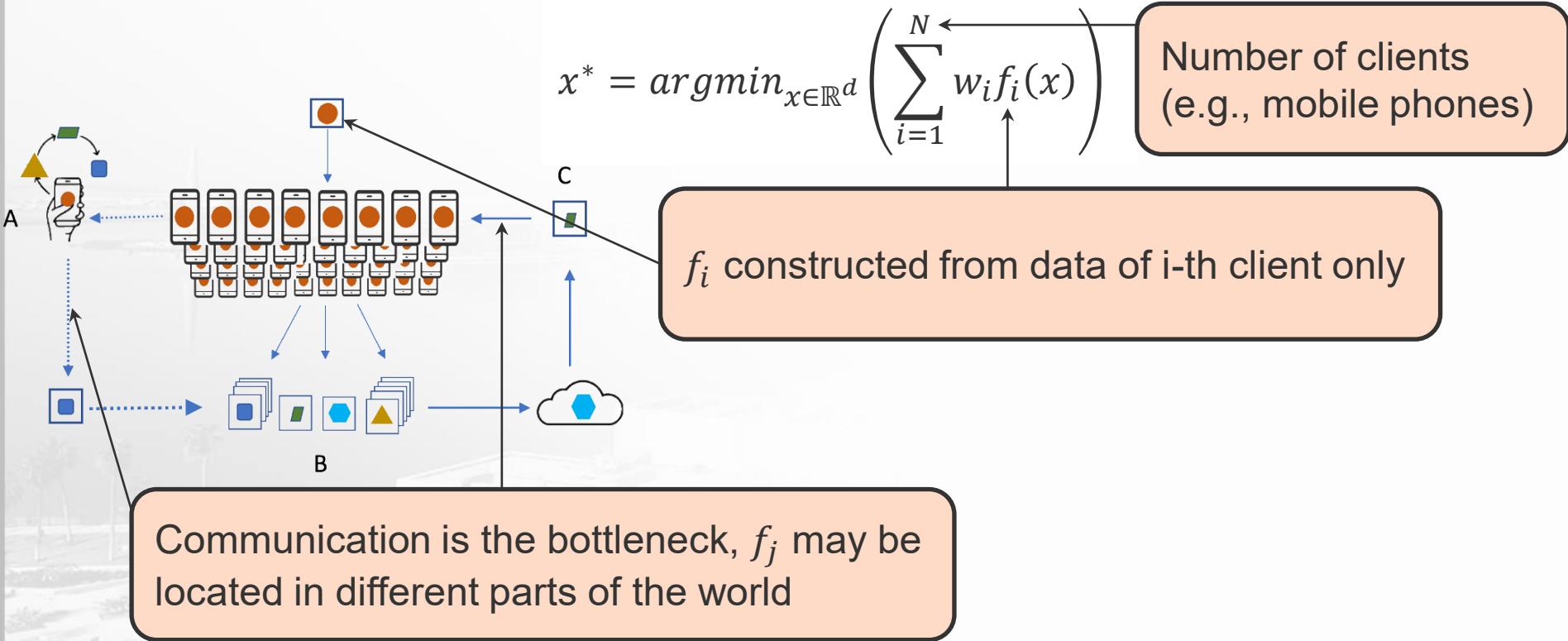
# Federated Learning



The goal of FL is to find a model deployable on each client while protecting the clients' data

Some challenges: many devices; privacy; data heterogeneity; communication; partial participation

# Federated Learning: Classical Objective



# Why do we need an optimization research simulator?

- Existing frameworks are mainly built with a focus to be deployed on real-world systems
- The entry bar for researchers tends to increase
- Frameworks require extensive experience with distributed systems or assistance from experts on a given framework



In our work, we take one step back and focus on constructing a framework whose primary goal is to provide a useful simulation environment for researchers.

# Open-source Federated Learning Frameworks

 <b>TensorFlow</b>	TensorFlow Federated (TFF) is a framework for machine learning
 <b>PFL</b> Paddle Federated Learning	Framework based on PaddlePaddle from BAIDU Research.
 <b>NVIDIA CLARA</b>	Clara Training Framework developed by NVIDIA Research with a focus on the medical domain.
 <b>Flower</b>	Framework targeted to bring Federated Learning into the real world and account for compute heterogeneity, memory constraints, and scaling aspects.
 <b>FEDML</b> MAKING COLLABORATION INTELLIGENT	A valuable tool for benchmarking and experiments for Federated Learning.
 <b>Syft</b>	PySyft extends PyTorch for private Deep Learning via providing mechanisms based on last advancements in Differential Privacy, Multi-Party Computation, Homomorphic Encryption (HE).

# Our goals

Low Entry Bar/Simplicity

Extensibility

Hardware Utilization

Easy Debugging

# Our goals

Low Entry Bar/Simplicity



Extensibility

Hardware Utilization

Easy Debugging

We aim our tool to be as simple as possible.

1. GUI interface to launch experiments
2. Communication with standard TCP/IP transport
3. Serialization results of experiments
4. Creating customizable high-quality plots

# Our goals

Low Entry Bar/Simplicity



We aim our tool to be as simple as possible

Extensibility



We aim to provide universal abstractions

1. Universal abstraction for optimization algorithms in FL
2. Easy way to bring own opt. algorithms, datasets, etc.
3. Flexibility to experiment with existing and novel approaches to advance the state-of-the-art

Hardware Utilization

Easy Debugging

# Our goals

Low Entry Bar/Simplicity



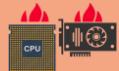
We aim our tool to be as simple as possible

Extensibility



We aim to provide universal abstractions

Hardware Utilization



We aim to provide extensive runtime configuration

1. High CPU/GPU utilization under the hood
2. Ability to include remote compute resources
3. Provide runtime configuration at a single experiment granularity
4. Monitoring tools for tracking experimental progress and device utilization

Easy Debugging

# Our goals

Low Entry Bar/ Simplicity 

Extensibility 

Hardware Utilization 

Easy Debugging 

We aim to help to debug FL opt. algorithms

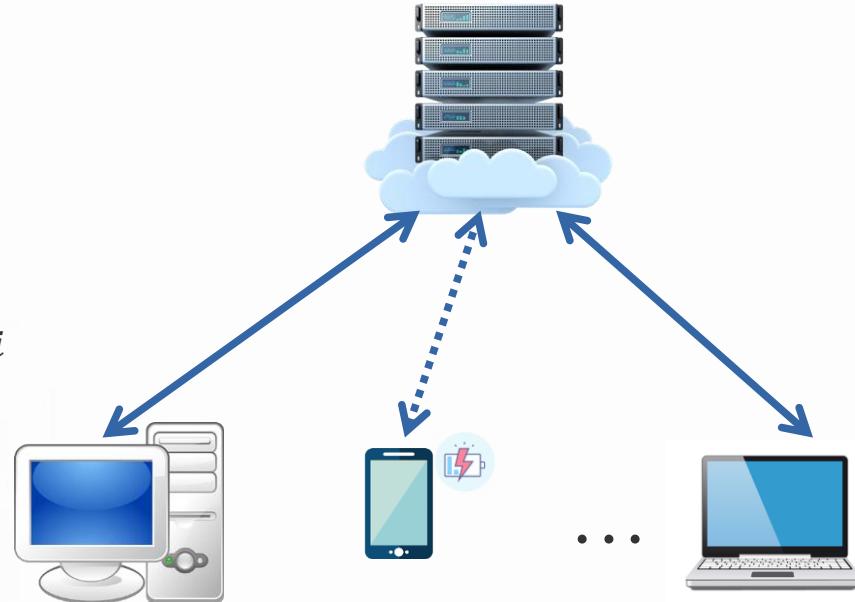
1. The user provides only a single-node implementation
2. Mapping to multi-GPU/CPU and multi-node setup happens during runtime configuration only
3. GUI/CUI provides necessary logging information
4. Means to reproduce the experiments

# Federated Learning Objective

$$\min_x F(x) \triangleq \mathbb{E}_{i \sim \mathcal{P}}[F_i(x)]$$

$$F_i(x) = \mathbb{E}_{\xi \sim D_i}[f_i(x, \xi)]$$

- $F$  is global objective with randomness inherited from the client distribution  $\mathcal{P}$
- $F_i$  is local objective with randomness inherited from the client data distribution  $D_i$



# Generalized Federated Averaging

**Input:** Initial model  $\mathbf{x}^{(0)}$ , CLIENTOPT, SERVEROPT  
Initialize server state  $H^0 = \text{INITIALSERVERSTATE}()$

**for**  $t \in \{0, 1, \dots, T-1\}$  **do**

- Sample a subset  $\mathcal{S}^{(t)}$  of available clients
- Generate state:  $s^{(t)} = \text{CLIENTSTATE}(H^{(t)})$
- Broadcast  $(\mathbf{x}^{(t)}, s^{(t)})$  to workers
- for** client  $i \in \mathcal{S}^{(t)}$  **in parallel do**

  - Initialize local model  $\mathbf{x}_i^{(t,0)} = \mathbf{x}^{(t)}$
  - for**  $k = 0, \dots, \tau_i - 1$  **do**

    - Compute local stochastic gradient  $g_i = \text{LOCALGRADIENT}(\mathbf{x}_i^{(t,k)}, s_t)$
    - Perform local update  $\mathbf{x}_i^{(t,k+1)} = \text{CLIENTOPT}(\mathbf{x}_i^{(t,k)}, g_i, k, t)$

  - end**
  - Compute local model changes  $\Delta_i^{(t)} = \mathbf{x}_i^{(t,\tau_i)} - \mathbf{x}_i^{(t,0)}$
  - Create local state update:  $U_i^{(t)} = \text{LOCALSTATE}(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t,\tau_i)})$
  - Send  $(\Delta_i^{(t)}, U_i^{(t)})$  to Master.

- end**
- Obtain  $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$ .
- Compute  $G^{(t)} = \text{SERVERGRADIENT}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$
- Update global model  $\mathbf{x}^{(t+1)} = \text{SERVEROPT}(\mathbf{x}^{(t)}, G^{(t)}, \eta_s, t)$
- Update:  $H^{(t+1)} = \text{SERVERGLOBALSTATE}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$

**end**

## FL\_PyTorch

- Written in Python language only 
- Build upon PyTorch abstractions 

# Generalized Federated Averaging

```
Input: Initial model  $\mathbf{x}^{(0)}$ , CLIENTOPT, SERVEROPT  
Initialize server state  $H^0 = \text{INITIALSERVSTATE}()$   
for  $t \in \{0, 1, \dots, T - 1\}$  do  
    Sample a subset  $\mathcal{S}^{(t)}$  of available clients  
    Generate state:  $s^{(t)} = \text{CLIENTSTATE}(H^{(t)})$   
    Broadcast  $(\mathbf{x}^{(t)}, s^{(t)})$  to workers  
    for client  $i \in \mathcal{S}^{(t)}$  in parallel do  
        Initialize local model  $\mathbf{x}_i^{(t,0)} = \mathbf{x}^{(t)}$   
        for  $k = 0, \dots, \tau_i - 1$  do  
            Compute local stochastic gradient  $g_i = \text{LOCALGRADIENT}(\mathbf{x}_i^{(t,k)}, s_t)$   
            Perform local update  $\mathbf{x}_i^{(t,k+1)} = \text{CLIENTOPT}(\mathbf{x}_i^{(t,k)}, g_i, k, t)$   
        end  
        Compute local model changes  $\Delta_i^{(t)} = \mathbf{x}_i^{(t,\tau_i)} - \mathbf{x}_i^{(t,0)}$   
        Create local state update:  $U_i^{(t)} = \text{LOCALSTATE}(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t,\tau_i)})$   
        Send  $(\Delta_i^{(t)}, U_i^{(t)})$  to Master.  
    end  
    Obtain  $(\Delta_i^{(t)}, U_i^{(t)})$ ,  $\forall i \in \mathcal{S}^{(t)}$ .  
    Compute  $G^{(t)} = \text{SERVERGRADIENT}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$   
    Update global model  $\mathbf{x}^{(t+1)} = \text{SERVEROPT}(\mathbf{x}^{(t)}, G^{(t)}, \eta_s, t)$   
    Update:  $H^{(t+1)} = \text{SERVERGLOBALSTATE}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$   
end
```

## InitializeServerState

This method returns a dictionary that initializes the server state. The method constructs and initializes the model.

# Generalized Federated Averaging

**Input:** Initial model  $\mathbf{x}^{(0)}$ , CLIENTOPT, SERVEROPT  
Initialize server state  $H^0 = \text{INITIALSERVERSTATE}()$

```
for  $t \in \{0, 1, \dots, T - 1\}$  do
    Sample a subset  $\mathcal{S}^{(t)}$  of available clients
    Generate state:  $s^{(t)} = \text{CLIENTSTATE}(H^{(t)})$ 
    Broadcast  $(\mathbf{x}^{(t)}, s^{(t)})$  to workers
    for client  $i \in \mathcal{S}^{(t)}$  in parallel do
        Initialize local model  $\mathbf{x}_i^{(t,0)} = \mathbf{x}^{(t)}$ 
        for  $k = 0, \dots, \tau_i - 1$  do
            Compute local stochastic gradient  $g_i = \text{LOCALGRADIENT}(\mathbf{x}_i^{(t,k)}, s_t)$ 
            Perform local update  $\mathbf{x}_i^{(t,k+1)} = \text{CLIENTOPT}(\mathbf{x}_i^{(t,k)}, g_i, k, t)$ 
        end
        Compute local model changes  $\Delta_i^{(t)} = \mathbf{x}_i^{(t,\tau_i)} - \mathbf{x}_i^{(t,0)}$ 
        Create local state update:  $U_i^{(t)} = \text{LOCALSTATE}(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t,\tau_i)})$ 
        Send  $(\Delta_i^{(t)}, U_i^{(t)})$  to Master.
    end
    Obtain  $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$ .
    Compute  $G^{(t)} = \text{SERVERGRADIENT}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$ 
    Update global model  $\mathbf{x}^{(t+1)} = \text{SERVEROPT}(\mathbf{x}^{(t)}, G^{(t)}, \eta_s, t)$ 
    Update:  $H^{(t+1)} = \text{SERVERGLOBALSTATE}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$ 
end
```

## ClientState

By our design client state is stateless.  
The client state is instantiated at the beginning of each round for each of the selected clients.

If you need a client state you may initialize it from the server state.

# Generalized Federated Averaging

```
Input: Initial model  $\mathbf{x}^{(0)}$ , CLIENTOPT, SERVEROPT  
Initialize server state  $H^0 = \text{INITIALSERVERSTATE}()$   
for  $t \in \{0, 1, \dots, T - 1\}$  do  
    Sample a subset  $\mathcal{S}^{(t)}$  of available clients  
    Generate state:  $s^{(t)} = \text{CLIENTSTATE}(H^{(t)})$   
    Broadcast  $(\mathbf{x}^{(t)}, s^{(t)})$  to workers  
    for client  $i \in \mathcal{S}^{(t)}$  in parallel do  
        Initialize local model  $\mathbf{x}_i^{(t,0)} = \mathbf{x}^{(t)}$   
        for  $k = 0, \dots, \tau_i - 1$  do  
            Compute local stochastic gradient  $g_i = \text{LOCALGRADIENT}(\mathbf{x}_i^{(t,k)}, s_t)$   
            Perform local update  $\mathbf{x}_i^{(t,k+1)} = \text{CLIENTOPT}(\mathbf{x}_i^{(t,k)}, g_i, k, t)$   
        end  
        Compute local model changes  $\Delta_i^{(t)} = \mathbf{x}_i^{(t,\tau_i)} - \mathbf{x}_i^{(t,0)}$   
        Create local state update:  $U_i^{(t)} = \text{LOCALSTATE}(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t,\tau_i)})$   
        Send  $(\Delta_i^{(t)}, U_i^{(t)})$  to Master.  
    end  
    Obtain  $(\Delta_i^{(t)}, U_i^{(t)})$ ,  $\forall i \in \mathcal{S}^{(t)}$ .  
    Compute  $G^{(t)} = \text{SERVERGRADIENT}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$   
    Update global model  $\mathbf{x}^{(t+1)} = \text{SERVEROPT}(\mathbf{x}^{(t)}, G^{(t)}, \eta_s, t)$   
    Update:  $H^{(t+1)} = \text{SERVERGLOBALSTATE}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$   
end
```

## LocalGradient

Obtain the algorithm specific local gradient estimator  $g_i$ .

# Generalized Federated Averaging

```
Input: Initial model  $\mathbf{x}^{(0)}$ , CLIENTOPT, SERVEROPT  
Initialize server state  $H^0 = \text{INITIALSERVERSTATE}()$   
for  $t \in \{0, 1, \dots, T - 1\}$  do  
    Sample a subset  $\mathcal{S}^{(t)}$  of available clients  
    Generate state:  $s^{(t)} = \text{CLIENTSTATE}(H^{(t)})$   
    Broadcast  $(\mathbf{x}^{(t)}, s^{(t)})$  to workers  
    for client  $i \in \mathcal{S}^{(t)}$  in parallel do  
        Initialize local model  $\mathbf{x}_i^{(t,0)} = \mathbf{x}^{(t)}$   
        for  $k = 0, \dots, \tau_i - 1$  do  
            Compute local stochastic gradient  $g_i = \text{LOCALGRADIENT}(\mathbf{x}_i^{(t,k)}, s_t)$   
            Perform local update  $\mathbf{x}_i^{(t,k+1)} = \text{CLIENTOPT}(\mathbf{x}_i^{(t,k)}, g_i, k, t)$   
        end  
        Compute local model changes  $\Delta_i^{(t)} = \mathbf{x}_i^{(t,\tau_i)} - \mathbf{x}_i^{(t,0)}$   
        Create local state update:  $U_i^{(t)} = \text{LOCALSTATE}(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t,\tau_i)})$   
        Send  $(\Delta_i^{(t)}, U_i^{(t)})$  to Master.  
    end  
    Obtain  $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$ .  
    Compute  $G^{(t)} = \text{SERVERGRADIENT}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$   
    Update global model  $\mathbf{x}^{(t+1)} = \text{SERVEROPT}(\mathbf{x}^{(t)}, G^{(t)}, \eta_s, t)$   
    Update:  $H^{(t+1)} = \text{SERVERGLOBALSTATE}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$   
end
```

## ClientOpt

Local optimization step using the obtained direction  $g_i$ .

# Generalized Federated Averaging

**Input:** Initial model  $\mathbf{x}^{(0)}$ , CLIENTOPT, SERVEROPT

Initialize server state  $H^0 = \text{INITIALSERVSTATE}()$

**for**  $t \in \{0, 1, \dots, T-1\}$  **do**

    Sample a subset  $\mathcal{S}^{(t)}$  of available clients

    Generate state:  $s^{(t)} = \text{CLIENTSTATE}(H^{(t)})$

    Broadcast  $(\mathbf{x}^{(t)}, s^{(t)})$  to workers

**for** client  $i \in \mathcal{S}^{(t)}$  **in parallel do**

        Initialize local model  $\mathbf{x}_i^{(t,0)} = \mathbf{x}^{(t)}$

**for**  $k = 0, \dots, \tau_i - 1$  **do**

            Compute local stochastic gradient  $g_i = \text{LOCALGRADIENT}(\mathbf{x}_i^{(t,k)}, s_t)$

            Perform local update  $\mathbf{x}_i^{(t,k+1)} = \text{CLIENTOPT}(\mathbf{x}_i^{(t,k)}, g_i, k, t)$

**end**

        Compute local model changes  $\Delta_i^{(t)} = \mathbf{x}_i^{(t,\tau_i)} - \mathbf{x}_i^{(t,0)}$

        Create local state update:  $U_i^{(t)} = \text{LOCALSTATE}(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t,\tau_i)})$

        Send  $(\Delta_i^{(t)}, U_i^{(t)})$  to Master.

**end**

    Obtain  $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$ .

    Compute  $G^{(t)} = \text{SERVERGRADIENT}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$

    Update global model  $\mathbf{x}^{(t+1)} = \text{SERVEROPT}(\mathbf{x}^{(t)}, G^{(t)}, \eta_s, t)$

    Update:  $H^{(t+1)} = \text{SERVERGLOBALSTATE}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$

**end**

## LocalState

Collect all the local states that are sent to the master jointly with the local update.

# Generalized Federated Averaging

```
Input: Initial model  $\mathbf{x}^{(0)}$ , CLIENTOPT, SERVEROPT  
Initialize server state  $H^0 = \text{INITIALSERVERSTATE}()$   
for  $t \in \{0, 1, \dots, T - 1\}$  do  
    Sample a subset  $\mathcal{S}^{(t)}$  of available clients  
    Generate state:  $s^{(t)} = \text{CLIENTSTATE}(H^{(t)})$   
    Broadcast  $(\mathbf{x}^{(t)}, s^{(t)})$  to workers  
    for client  $i \in \mathcal{S}^{(t)}$  in parallel do  
        Initialize local model  $\mathbf{x}_i^{(t,0)} = \mathbf{x}^{(t)}$   
        for  $k = 0, \dots, \tau_i - 1$  do  
            Compute local stochastic gradient  $g_i = \text{LOCALGRADIENT}(\mathbf{x}_i^{(t,k)}, s_t)$   
            Perform local update  $\mathbf{x}_i^{(t,k+1)} = \text{CLIENTOPT}(\mathbf{x}_i^{(t,k)}, g_i, k, t)$   
        end  
        Compute local model changes  $\Delta_i^{(t)} = \mathbf{x}_i^{(t,\tau_i)} - \mathbf{x}_i^{(t,0)}$   
        Create local state update:  $U_i^{(t)} = \text{LOCALSTATE}(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t,\tau_i)})$   
        Send  $(\Delta_i^{(t)}, U_i^{(t)})$  to Master.  
    end  
    Obtain  $(\Delta_i^{(t)}, U_i^{(t)})$ ,  $\forall i \in \mathcal{S}^{(t)}$ .  
    Compute  $G^{(t)} = \text{SERVERGRADIENT}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$   
    Update global model  $\mathbf{x}^{(t+1)} = \text{SERVEROPT}(\mathbf{x}^{(t)}, G^{(t)}, \eta_s, t)$   
    Update:  $H^{(t+1)} = \text{SERVERGLOBALSTATE}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$   
end
```

## ServerGradient

The estimator of the global direction to be used for the global model update.

# Generalized Federated Averaging

**Input:** Initial model  $\mathbf{x}^{(0)}$ , CLIENTOPT, SERVEROPT

Initialize server state  $H^0 = \text{INITIALSERVERSTATE}()$

**for**  $t \in \{0, 1, \dots, T-1\}$  **do**

    Sample a subset  $\mathcal{S}^{(t)}$  of available clients

    Generate state:  $s^{(t)} = \text{CLIENTSTATE}(H^{(t)})$

    Broadcast  $(\mathbf{x}^{(t)}, s^{(t)})$  to workers

**for** client  $i \in \mathcal{S}^{(t)}$  **in parallel do**

        Initialize local model  $\mathbf{x}_i^{(t,0)} = \mathbf{x}^{(t)}$

**for**  $k = 0, \dots, \tau_i - 1$  **do**

            Compute local stochastic gradient  $g_i = \text{LOCALGRADIENT}(\mathbf{x}_i^{(t,k)}, s_t)$

            Perform local update  $\mathbf{x}_i^{(t,k+1)} = \text{CLIENTOPT}(\mathbf{x}_i^{(t,k)}, g_i, k, t)$

**end**

        Compute local model changes  $\Delta_i^{(t)} = \mathbf{x}_i^{(t,\tau_i)} - \mathbf{x}_i^{(t,0)}$

        Create local state update:  $U_i^{(t)} = \text{LOCALSTATE}(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t,\tau_i)})$

        Send  $(\Delta_i^{(t)}, U_i^{(t)})$  to Master.

**end**

    Obtain  $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$ .

    Compute  $G^{(t)} = \text{SERVERGRADIENT}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$

    Update global model  $\mathbf{x}^{(t+1)} = \text{SERVEROPT}(\mathbf{x}^{(t)}, G^{(t)}, \eta_s, t)$

    Update:  $H^{(t+1)} = \text{SERVERGLOBALSTATE}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$

**end**

## ServerOpt

Server optimization step using the obtained direction  $G^t$ .

# Generalized Federated Averaging

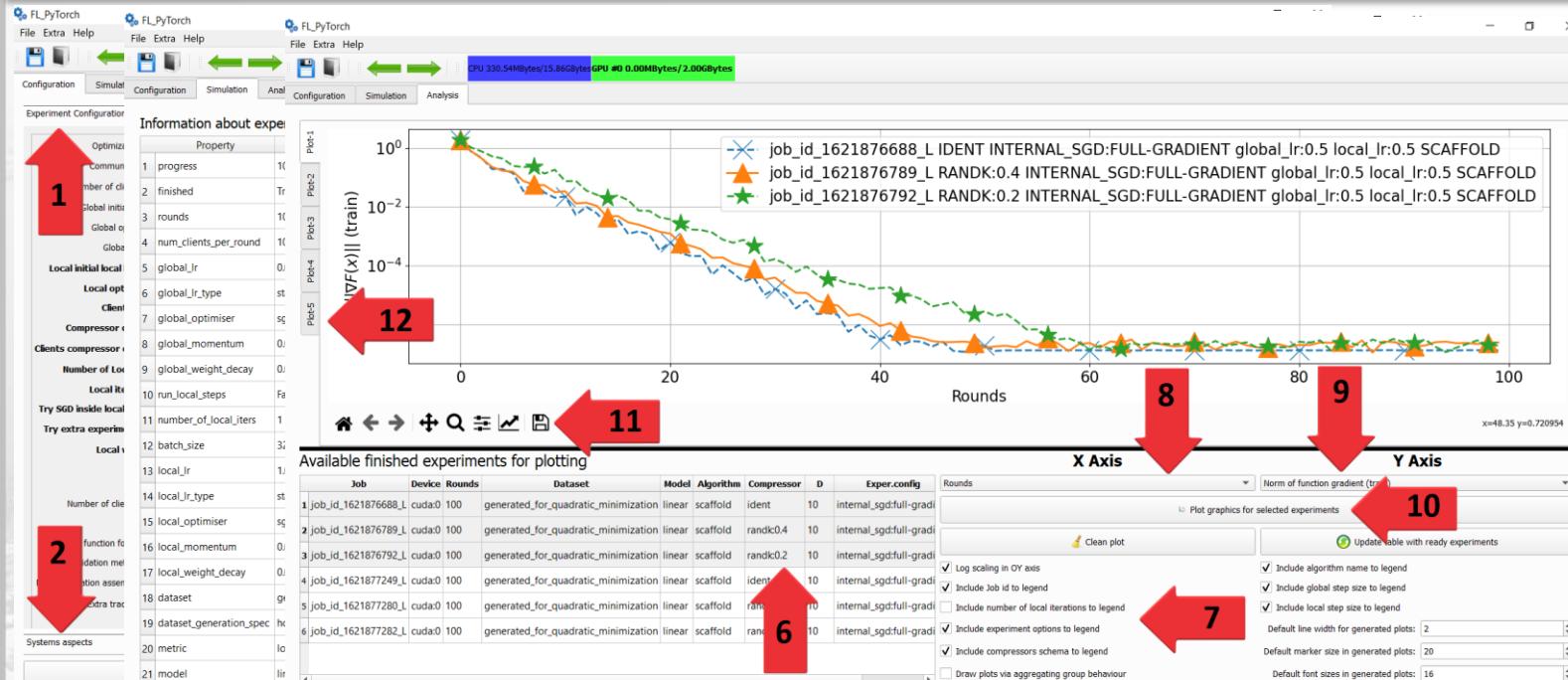
```
Input: Initial model  $\mathbf{x}^{(0)}$ , CLIENTOPT, SERVEROPT  
Initialize server state  $H^0 = \text{INITIALSERVERSTATE}()$   
for  $t \in \{0, 1, \dots, T - 1\}$  do  
    Sample a subset  $\mathcal{S}^{(t)}$  of available clients  
    Generate state:  $s^{(t)} = \text{CLIENTSTATE}(H^{(t)})$   
    Broadcast  $(\mathbf{x}^{(t)}, s^{(t)})$  to workers  
    for client  $i \in \mathcal{S}^{(t)}$  in parallel do  
        Initialize local model  $\mathbf{x}_i^{(t,0)} = \mathbf{x}^{(t)}$   
        for  $k = 0, \dots, \tau_i - 1$  do  
            Compute local stochastic gradient  $g_i = \text{LOCALGRADIENT}(\mathbf{x}_i^{(t,k)}, s_t)$   
            Perform local update  $\mathbf{x}_i^{(t,k+1)} = \text{CLIENTOPT}(\mathbf{x}_i^{(t,k)}, g_i, k, t)$   
        end  
        Compute local model changes  $\Delta_i^{(t)} = \mathbf{x}_i^{(t,\tau_i)} - \mathbf{x}_i^{(t,0)}$   
        Create local state update:  $U_i^{(t)} = \text{LOCALSTATE}(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t,\tau_i)})$   
        Send  $(\Delta_i^{(t)}, U_i^{(t)})$  to Master.  
    end  
    Obtain  $(\Delta_i^{(t)}, U_i^{(t)})$ ,  $\forall i \in \mathcal{S}^{(t)}$ .  
    Compute  $G^{(t)} = \text{SERVERGRADIENT}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$   
    Update global model  $\mathbf{x}^{(t+1)} = \text{SERVEROPT}(\mathbf{x}^{(t)}, G^{(t)}, \eta_s, t)$   
    Update:  $H^{(t+1)} = \text{SERVERGLOBALSTATE}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in \mathcal{S}^{(t)}}, H^{(t)})$   
end
```

## ServerGlobalState

The global server state update

# FL Optimization Research Simulator: UI

## Graphical UI



# FL Optimization Research Simulator: UI

## Console UI

```
[2021-05-26 19:37:47] (Thread-46) [utils.py:132] INFO - Train [99][0/16] DataLoad time 0.021 F/W time 0.317 B/W time 0.444 Loss 0.8583 Prec@1 12.500 Prec@5 62.500
[2021-05-26 19:37:47] (Thread-49) [utils.py:132] INFO - Train [99] Log window
[2021-05-26 19:37:47] (Thread-47) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:47] (Thread-50) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-48) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-49) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-55) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-51) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-53) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-58) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-50) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-56) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-52) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-54) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-57) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-46) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-50) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-47) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:49] (Thread-48) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:49] (Thread-49) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:49] (Thread-47) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:49] (Thread-46) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:50] (Thread-55) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:50] (Thread-51) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:50] (Thread-52) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:50] (Thread-53) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:50] (Thread-54) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:50] (Thread-58) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:50] (Thread-56) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:50] (Thread-57) [utils.py:132] INFO - Train [99]
[2021-05-26 19:37:51] (Thread-55) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:51] (Thread-51) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:51] (Thread-52) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:51] (Thread-53) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:51] (Thread-54) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:51] (Thread-56) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:51] (Thread-57) [model_funcs.py:150] INFO - Run [99]
[2021-05-26 19:37:51] (Thread-59) [utils.py:132] INFO - Train [99][1/16] DataLoad time 0.021 F/W time 0.317 B/W time 0.444 Loss 0.8583 Prec@1 12.500 Prec@5 62.500
[2021-05-26 19:37:51] (Thread-40) [utils.py:132] INFO - Train [99][0/16] DataLoad time 0.013 F/W time 0.351 B/W time 0.205 Loss 0.4103 Prec@1 21.875 Prec@5 59.375
[2021-05-26 19:37:51] (Thread-49) [utils.py:132] INFO - Train [99][0/16] DataLoad time 0.012 F/W time 0.346 B/W time 0.196 Loss 0.8375 Prec@1 21.875 Prec@5 65.625
[2021-05-26 19:37:51] (Thread-46) [utils.py:132] INFO - Train [99][0/16] DataLoad time 0.006 F/W time 0.326 B/W time 0.174 Loss 0.6756 Prec@1 21.875 Prec@5 56.250
[2021-05-26 19:37:51] (Thread-47) [utils.py:132] INFO - Train [99][0/16] DataLoad time 0.007 F/W time 0.348 B/W time 0.188 Loss 0.4500 Prec@1 21.875 Prec@5 56.250
[2021-05-26 19:37:52] (Thread-49) [utils.py:132] INFO - Train [99][10/16] DataLoad time 0.030 F/W time 0.676 B/W time 0.196 Loss 0.8907 Prec@1 13.636 Prec@5 58.807
[2021-05-26 19:37:52] (Thread-48) [utils.py:132] INFO - Train [99][10/16] DataLoad time 0.025 F/W time 0.687 B/W time 0.205 Loss 0.4977 Prec@1 15.341 Prec@5 53.977
[2021-05-26 19:37:52] (Thread-46) [utils.py:132] INFO - Train [99][10/16] DataLoad time 0.026 F/W time 0.649 B/W time 0.174 Loss 0.4897 Prec@1 15.057 Prec@5 56.250
[2021-05-26 19:37:52] (Thread-47) [utils.py:132] INFO - Train [99][10/16] DataLoad time 0.020 F/W time 0.681 B/W time 0.188 Loss 0.4774 Prec@1 19.886 Prec@5 57.955
[2021-05-26 19:37:52] (Thread-55) [utils.py:132] INFO - Train [99][0/16] DataLoad time 0.014 F/W time 0.343 B/W time 0.411 Loss 0.8242 Prec@1 9.375 Prec@5 53.125
[2021-05-26 19:37:52] (Thread-52) [utils.py:132] INFO - Train [99][0/16] DataLoad time 0.021 F/W time 0.332 B/W time 0.381 Loss 0.8605 Prec@1 15.625 Prec@5 56.250
[2021-05-26 19:37:52] (Thread-53) [utils.py:132] INFO - Train [99][0/16] DataLoad time 0.012 F/W time 0.346 B/W time 0.384 Loss 0.3921 Prec@1 15.625 Prec@5 62.500
[2021-05-26 19:37:52] (Thread-51) [utils.py:132] INFO - Train [99][0/16] DataLoad time 0.010 F/W time 0.343 B/W time 0.408 Loss 0.4088 Prec@1 21.875 Prec@5 46.075
```

# FL Optimization Research Simulator

Graphical UI

Console UI

Configuration

# FL Optimization Research Simulator

Graphical UI

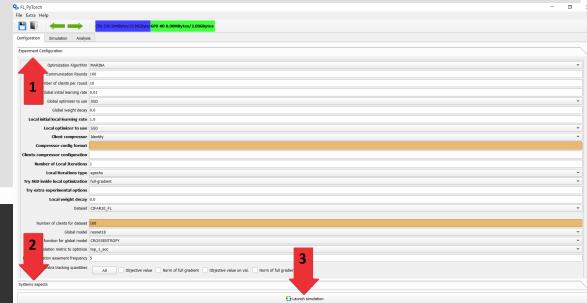
Console UI

**Server Optimizer:** number of communication rounds, the number of sampled clients etc.

**Local Optimizer:** number of local epochs or local iterations; batch sizes, etc.

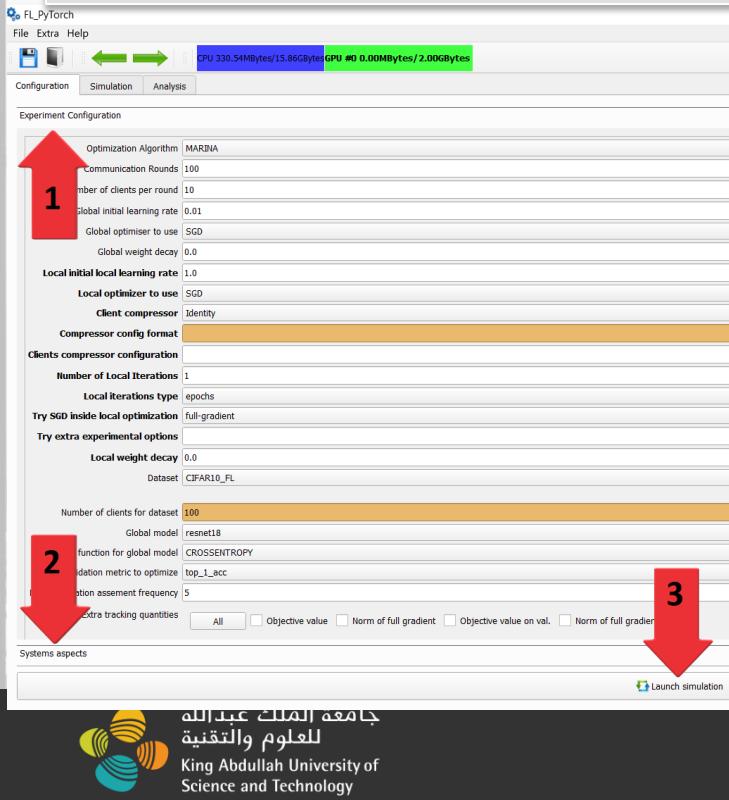
**Model and Data:** models' and datasets' names.

**System Setup:** random seeds, local/remote compute devices, thread pool sizes, etc.

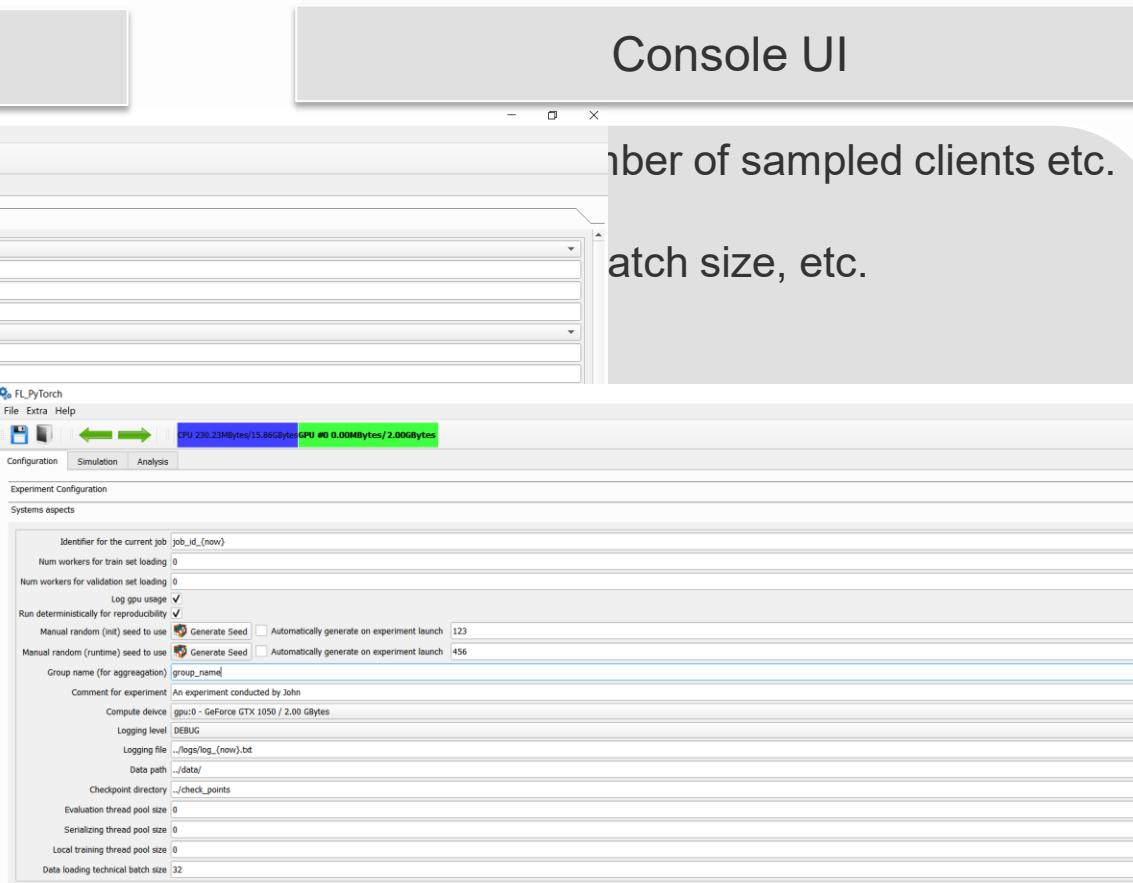


# FL Optimization Research Simulator

## Graphical UI



## Console UI



# FL Optimization Research Simulator

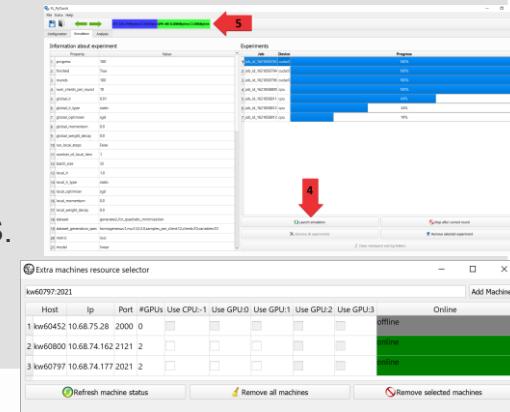
Graphical UI

Console UI

Configuration: command line or actions in GUI

Launch experiments: locally or remotely, in specified target devices

- Computation backend is a PyTorch.
- During the simulation, the selected local CPUs/GPUs are accessed in a parallel way.
- There is a possibility to use remote CPUs/GPUs with a TCP transport layer.
- Target devices are server and desktop stations running on Linux, macOS, or Windows OS.



# FL Optimization Research Simulator

Graphical UI

Console UI

Configuration: command line or actions in GUI

Launch experiments: locally or remotely, in specified target devices

The screenshot shows the FL\_PyTorch graphical interface. On the left, there's a sidebar titled 'Information about experiment' with various configuration parameters listed. On the right, a main panel titled 'Experiments' displays a table of running jobs. A red arrow labeled '5' points to the top-left corner of the window.

Property	Value
1 progress	100
2 finished	True
3 rounds	100
4 num_clients_per_round	10
5 global_lr	0.01
6 global_lr_type	static
7 global_optimizer	sgd
8 global_momentum	0.0
9 global_weight_decay	0.0
10 run_local_steps	False
11 number_of_local_iters	1
12 batch_size	32
13 local_lr	1.0
14 local_lr_type	static
15 local_optimizer	sgd
16 local_momentum	0.0
17 local_weight_decay	0.0
18 dataset	generated_for_quadratic_minimization
19 dataset_generation_spec	homogeneous:1,mu:1.0:2.0,samples_per_client:12,clients:10,variables:10
20 metric	loss
21 model	linear

Job	Device	Progress
1 job_id_1621858785	cuda:0	100%
2 job_id_1621858794	cuda:0	100%
3 job_id_1621858796	cuda:0	100%
4 job_id_1621858800	cpu	100%
5 job_id_1621858811	cpu	64%
6 job_id_1621858812	cpu	34%
7 job_id_1621858813	cpu	19%

At the bottom, there are buttons for 'Launch simulation' (red arrow '4'), 'Stop after current round', 'Remove all experiments', 'Remove selected experiment', 'Clean checkpoint and log folders', and a status message 'CPU: 335.54GB/s(0.53s) GPU: 0.00GB/s(2.00GB/s)'.

The screenshot shows the 'Extra machines resource selector' window. It lists three machines with their host, IP, port, and GPU usage status. A red arrow labeled '5' points to the top-left corner of the window.

Host	Ip	Port	#GPUs	Use CPU:-1	Use GPU:0	Use GPU:1	Use GPU:2	Use GPU:3	Online
1 kw60452	10.68.75.28	2000	0	<input type="checkbox"/>	offline				
2 kw60800	10.68.74.162	2121	2	<input type="checkbox"/>	online				
3 kw60797	10.68.74.177	2021	2	<input type="checkbox"/>	online				

Buttons at the bottom include 'Refresh machine status', 'Remove all machines', and 'Remove selected machines'.

# FL Optimization Research Simulator

Graphical UI

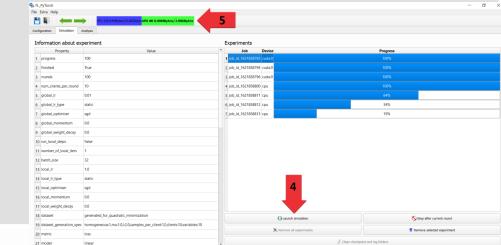
Console UI

Configuration: command line or actions in GUI

Launch experiments: locally or remotely, in specified target devices

Ability to monitor progress and current device load

- Quantities to track: *number of communication rounds, loss, accuracy, the norm of computed gradient, the norm of the full objective gradient, function values, number of gradient oracle calls, number of communicated bits.*
- Visualize the progress of their experiments in an interactive fashion.
- Monitor CPU/GPU utilization.



# FL Optimization Research Simulator

Graphical UI

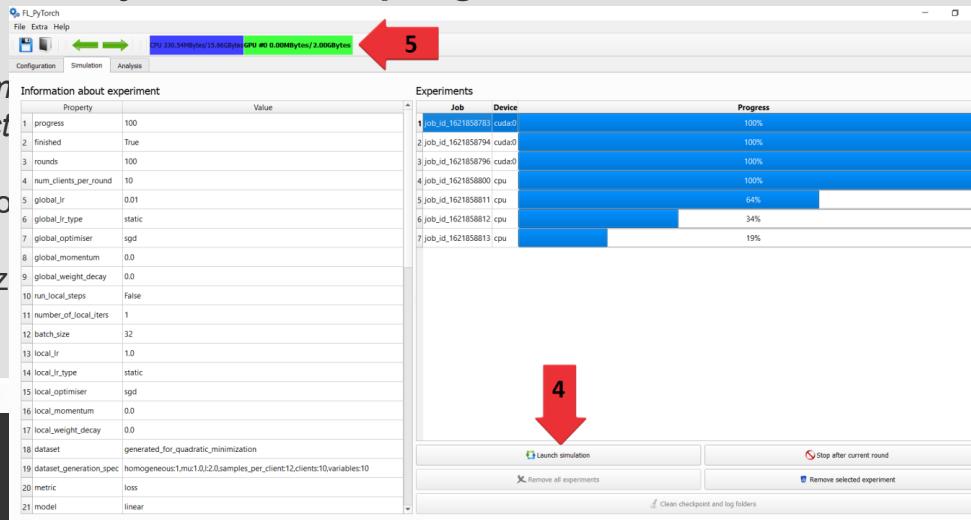
Console UI

Configuration: command line or actions in GUI

Launch experiments: locally or remotely, in specified target devices

Ability to monitor progress and current device load

- Quantities to track: *num objective gradient, function evals, time taken*
- Visualize the progress of each experiment
- Monitor CPU/GPU utilization



gradient, the norm of the full bits.

# FL Optimization Research Simulator

Graphical UI

Console UI

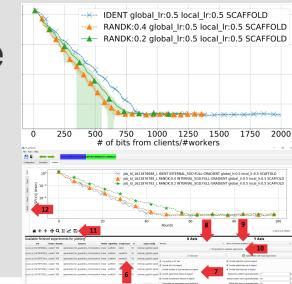
Configuration: command line or actions in GUI

Launch experiments: locally or remotely, in specified target devices

Monitoring tools per experiment progress, CPU/GPU load, current experiments

**Save, Load, Analyze results of experiments for future analysis**

- Visualize their experiments progress in an interactive fashion in the tool offline
- Export information to online tool [wanb.ai](http://wanb.ai)
- Save and load experimental results
- Recover arguments passed via the GUI tool in the command line format



# FL Optimization Research Simulator

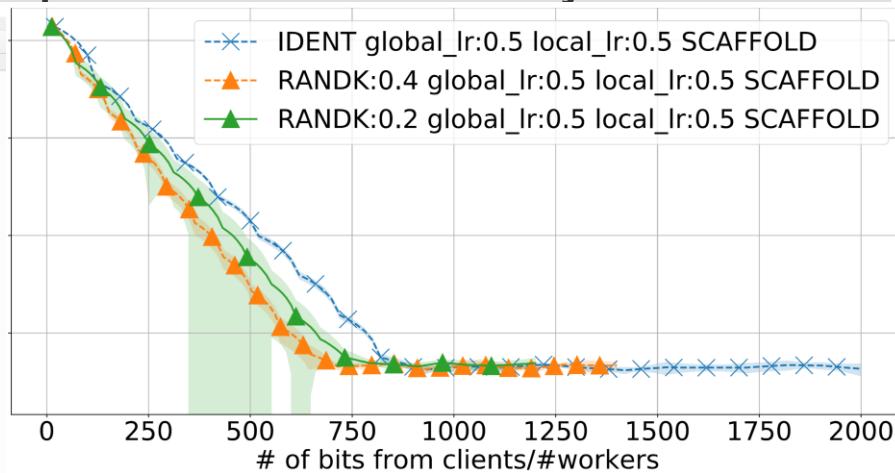
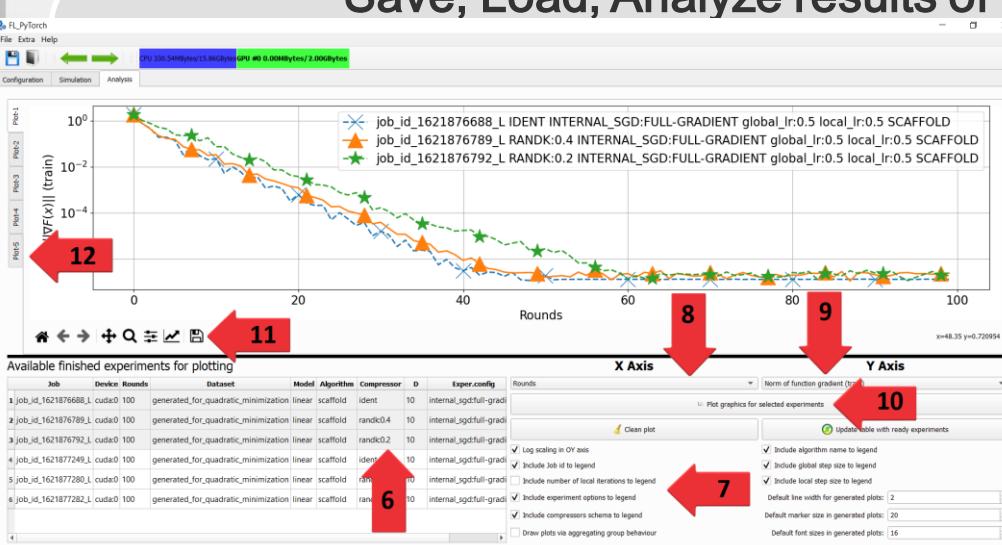
Graphical UI

Console UI

Configuration: command line or actions in GUI

...

## Save, Load, Analyze results of experiments for future analysis



# FL Optimization Research Simulator

## State-of-the-arts Opt.Algorithms

1. Distributed Compressed Gradient Descent
2. FedAVG
3. SCAFFOLD
4. FedProx
5. DIANA
6. MARINA

## Models and Datasets

- ResNet(18,34,50,101,152), VGG(11,13,16,19)
- WideResNets (28\_2, 28\_4, 28\_8)
- Support for simple controllable
- Quadratic functions as toy problems
- Standard FL datasets
- Synthetically generated

## Communication reduction mechanisms

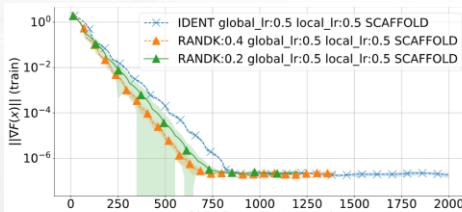
- Local updates
- Limiting the participating clients per round
- Compressors
  - Identical compressor
  - Lazy or Bernoulli compressor
  - Rand-K
  - Natural compressor
  - Standard dithering
  - Natural Dithering

# Experiment example

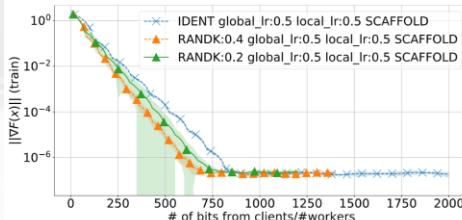
In the paper, we provide several experiments to showcase potential use cases of FL\_PyTorch

We used our framework to analyze whether SCAFFOLD can work with compression, i.e. whether messages from clients in the SCAFFOLD algorithm can be compressed.

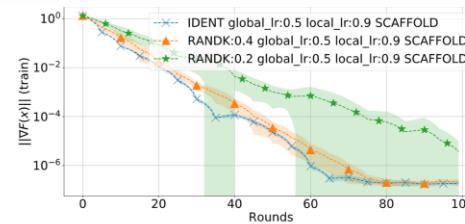
In synthesized quadratic optimization problem experiments demonstrated that it might be worthwhile to consider an extension of SCAFFOLD by adding compression.



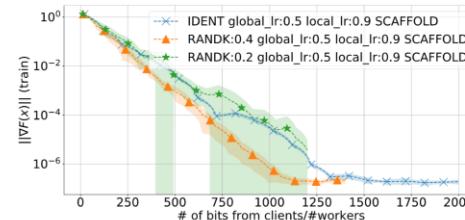
(a) One local iteration, convergence in gradient.



(c) One local iteration, average communication.



(b) Five local iteration, convergence in gradient.



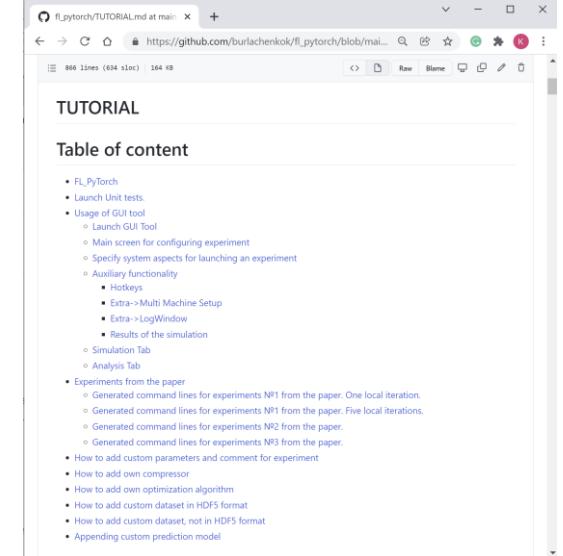
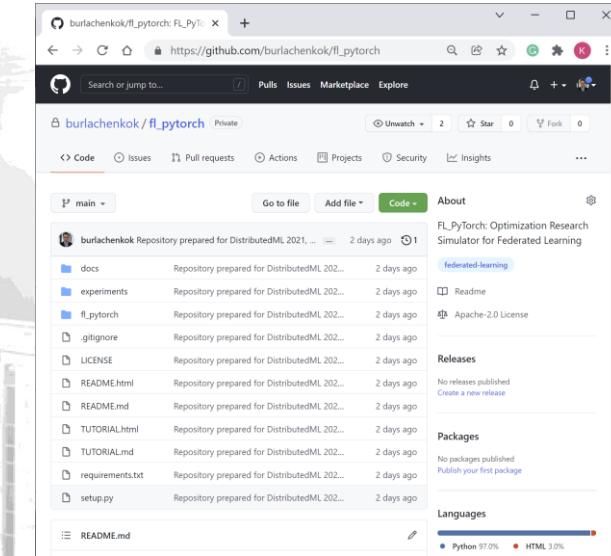
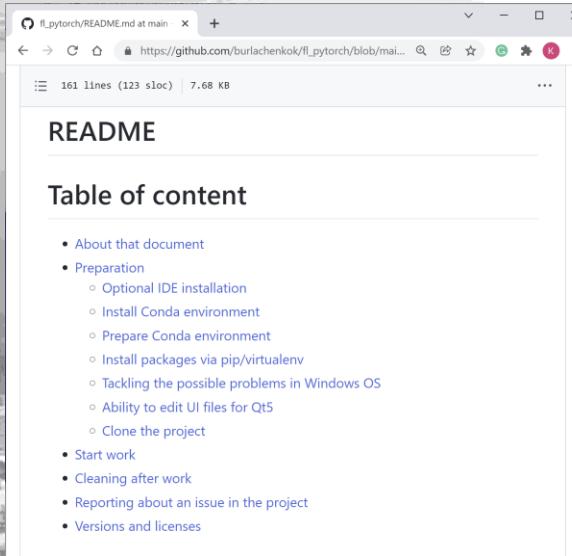
(d) Five local iterations, average communication.

# More Information

Please take a look at the original paper for motivation and goals.

Git repository: <https://github.com/burlachenkok/flpytorch.git>

Sources of information: paper, sources, readme, tutorial, documentation, unit tests.



# Thank you for your attention

Konstantin Burlachenko

**PhD student KAUST**

[konstantin.burlachenko@kaust.edu.sa](mailto:konstantin.burlachenko@kaust.edu.sa)

<https://burlachenkok.github.io/>

## FL\_PyTorch: Optimization Research Simulator for Federated Learning



جامعة الملك عبد الله  
للعلوم والتكنولوجيا  
King Abdullah University of  
Science and Technology

# Extra Slides

# Speeds of some buses

Memory Transferring direction within a single node (PC)	Approximate typical Speed
CPU <-> System DDR Memory	51 200 MBytes/sec(DDR5)
GPU core <-> GPU DDR Memory	128 000 MBytes/sec (GDDR6)
GPU DDR <-> PCI-E <-> System DDR Memory	4 000 MBytes / sec (PCI-E v5, 1 lane)
SATA 3x (HDD)	600 MBytes / sec (SATA 3.x, 6Gb/sec)
USB 3.0 (External storage)	600 MBytes / sec
GPU <-> GPU (NVLink)	50 000 Mbytes/sec
GPU <-> GPU (NVLink via NVSwitch inside DGX-2)	50 000 Mbytes/sec

Memory Transferring direction within a mult. nodes (PCs)	Typical Speed
Fast Ethernet	12.5 MBytes/sec
Gigabit Ethernet	125 MBytes/sec
InfiniBand HDR	6250 Mbytes/sec
InfiniBand Mellanox	25 000 Mbytes/sec

# NVIDIA GPU Software Compute Ecosystem

Application from specific domains												
Consumer Applications (recommendation systems)			Industrial Applications (Manufacture, Finance)				Scientific Applications (Molecular Simulation, Seismatics, etc.)					
Frameworks and Application												
MXNET	Chainer	PyTorch	TensorFlow	RAPIDS	Caffe	OCTAVE	Matlab	FFMPEG				
NVIDIA SDK AND LIBRARIES												
ML LIBRAIRES		DL LIBRARIES					HPC LIBRARIES					
cuDF	cuML	cuGRAPH	cuBLAS	cuDNN	TensorRT	cuTLAS S	cuBLAS, cuSPARSE	cuFFT	etc.			
CUDA Programming Model (including Developer Tools, Runtime, Compilers)												
GPU REAL AND VIRTUAL HARDWARE												
	GPUs		DGX Station			<a href="#">Virtual GPU</a>						

## Execution Context

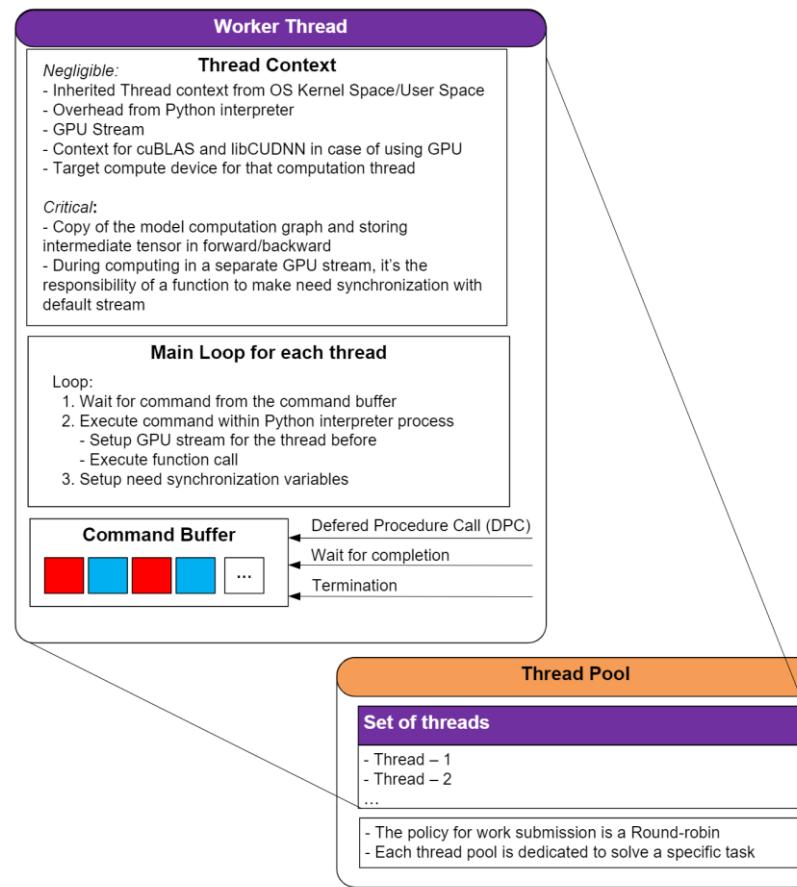
### Experiment Specific execution context

1. Threads responsible for model evaluation
2. Threads responsible for local training
3. Threads responsible for work involved saving results into the file system
4. Threads responsible for remote workers communication
5. Dictionary with experimental options to experiments with optimization algorithms
6. Thread specific NumPy and Python random generator

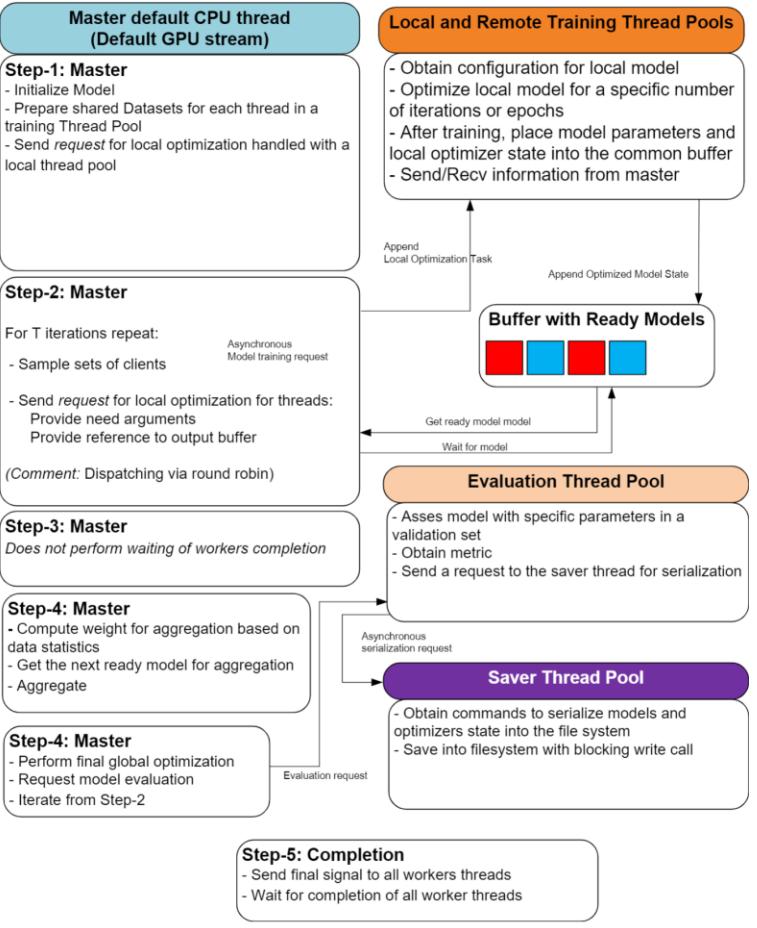
### Global callbacks

1. Callback for start simulation notification
2. Request for cancel simulation from outside
3. Progress tracking
4. Callback for notification that simulation is finished

FL\_PyTorch execution context for a single experiment. The GUI can handle several experiments at the same time.



A single worker thread structure and it's role in a thread pool.



Communication between different threads during Algorithm 1 execution

**Algorithm 10** Distributed Compressed Gradient Descent (DCGD)

```

1: Parameters: learning rate  $\gamma > 0$ , starting point  $x^0 \in \mathbb{R}^d$ , compression operators  $\mathcal{C}_1 \in \mathbb{B}^d(\omega_1), \dots, \mathcal{C}_n \in \mathbb{B}^d(\omega_n)$ ,  $\mathcal{C} \in \mathbb{B}^d(\omega_m)$ 
2: for  $k = 0, 1, 2, \dots$  do
3:   for all workers  $i \in \{1, 2, \dots, n\}$  in parallel do
4:     Compute local gradient  $\nabla f_i(x^k)$ 
5:     Compress local gradient  $g_i^k = \mathcal{C}_i^k(\nabla f_i(x^k))$ 
6:     Send  $g_i^k$  to master
7:   Master computes the aggregate  $\hat{g}^k = \frac{1}{n} \sum_{i=1}^n g_i^k$ 
8:   Master broadcasts the compressed aggregate  $\hat{g}^k = \mathcal{C}^k(\hat{g}^k)$  to all workers
9:   for all workers  $i \in \{1, 2, \dots, n\}$  in parallel do
10:    Compute the next iterate  $x^{k+1} = \text{prox}_{\eta}(\mathcal{X}^k - \gamma \hat{g}^k)$ 

```

**Algorithm 5** Distributed Compressed Gradient Descent (DCGD)

Parameters: learning rate  $\gamma > 0$ , compression operators  $\mathcal{C}_i$  with variance bounds  $\omega_i \geq 0$

**Initialization:**  $x^0 \in \mathbb{R}^d$

**for**  $k = 0, 1, 2, \dots$  **do**

- Broadcast  $x^k$  to all clients
- for each worker**  $i = 1, \dots, n$  **in parallel do**

  - Compute local gradient  $\nabla f_i(x^k)$
  - Sample a copy of  $\mathcal{C}_i$  as  $\mathcal{C}_i^k$
  - Compress local gradient  $\nabla f_i(x^k)$  into  $\mathcal{C}_i^k(\nabla f_i(x^k))$  and send to server

- end for**
- $x^{k+1} = x^k - \gamma \frac{1}{n} \sum_{i=1}^n \mathcal{C}_i^k(\nabla f_i(x^k))$

**end for**

**Algorithm 1** SCAFFOLD: Stochastic Controlled Averaging for federated learning

```

1: server input: initial  $x$  and  $c$ , and global step-size  $\eta_g$ 
2: client i's input:  $c_i$ , and local step-size  $\eta_l$ 
3: for each round  $r = 1, \dots, R$  do
4:   sample clients  $S \subseteq \{1, \dots, N\}$ 
5:   communicate  $(x, c)$  to all clients  $i \in S$ 
6:   on client i in parallel do
7:     initialize local model  $y_i \leftarrow x$ 
8:     for  $k = 1, \dots, K$  do
9:       compute mini-batch gradient  $g_i(y_i)$ 
10:       $y_i \leftarrow y_i - \eta_l(g_i(y_i) - c_i + c)$ 
11:     end for
12:      $c_i^+ \leftarrow (i) g_i(x)$ , or (ii)  $c_i - c + \frac{1}{K\eta_l}(x - y_i)$ 
13:     communicate  $(\Delta y_i, \Delta c_i) \leftarrow (y_i - x, c_i^+ - c_i)$ 
14:      $c_i \leftarrow c_i^+$ 
15:   end for
16:    $(\Delta x, \Delta c) \leftarrow \frac{1}{|S|} \sum_{i \in S} (\Delta y_i, \Delta c_i)$ 
17:    $x \leftarrow x + \eta_g \Delta x$  and  $c \leftarrow c + \frac{|S|}{N} \Delta c$ 
18: end for

```

**Algorithm 2** Accelerated DIANA (ADIANA)

**Input:** initial point  $x^0, \{h_i^0\}_{i=1}^n, h^0 = \frac{1}{n} \sum_{i=1}^n h_i^0$ , parameters  $\eta, \theta_1, \theta_2, \alpha, \beta, \gamma, p$

- 1:  $z^0 = y^0 = w^0 = x^0$
- 2: **for**  $k = 0, 1, 2, \dots$  **do**
- 3:  $z^k = \theta_1 z^{k-1} + \theta_2 w^{k-1} + (1 - \theta_1 - \theta_2)y^k$
- 4: **for all machines**  $i = 1, 2, \dots, n$  **do in parallel**
- 5: Compress shifted local gradient  $\mathcal{C}_i^k(\nabla f_i(x^k) - h_i^k)$  and send to the server
- 6: Update local shift  $h_i^{k+1} = h_i^k + \alpha \mathcal{C}_i^k(\nabla f_i(w^k) - h_i^k)$
- 7: **end for**
- 8: Aggregate received compressed gradient information  

$$g^k = \frac{1}{n} \sum_{i=1}^n \mathcal{C}_i^k(\nabla f_i(x^k) - h_i^k) + h^k$$

$$h^{k+1} = h^k + \alpha \frac{1}{n} \sum_{i=1}^n \mathcal{C}_i^k(\nabla f_i(w^k) - h_i^k)$$
- 9: Perform update step  

$$y^{k+1} = \text{prox}_{\eta g}(x^k - \eta g^k)$$

$$z^{k+1} = \beta z^k + (1 - \beta)x^k + \frac{\gamma}{\eta}(y^{k+1} - x^k)$$
- 10:  $w^{k+1} = \begin{cases} y^k, & \text{with probability } p \\ w^k, & \text{with probability } 1 - p \end{cases}$
- 11: **end for**

**Algorithm 2** NL2: NEWTON-LEARN (general case)

**Parameters:**  $\eta > 0; \gamma > 0$

**Initialization:**  $x^0 \in \mathbb{R}^d, h_i^0 \in \mathbb{R}_+^m, \mathbf{A}^0 = \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m (h_{ij}^0 + 2\gamma) a_{ij} a_{ij}^\top \in \mathbb{R}^{d \times d}$

**for**  $k = 0, 1, 2, \dots$  **do**

- broadcast  $x^k$  to all workers
- for**  $i = 1, \dots, n$  **do**

  - Compute local gradient  $\nabla f_i(x^k)$
  - $h_i^{k+1} = h_i^k + \eta \mathcal{C}_i^k(h_i(x^k) - h_i^k)$
  - $\beta_i^k = \max_{j \in [m]} \frac{h_{ij}^k + 2\gamma}{h_{ij}^k + 2\gamma}$
  - Send  $\nabla f_i(x^k), \beta_i^k$ , and  $\eta \mathcal{C}_i^k(h_i(x^k) - h_i^k)$  to server

- Option 1:** Send  $\{a_{ij} : h_{ij}^{k+1} - h_{ij}^k \neq 0\}$  to server
- Option 2:** Do nothing if server knows  $\{a_{ij} : \forall j\}$

**end for**

$\beta^k = \max_i \{\beta_i^k\}$

$\mathbf{H}^k = \beta^k \mathbf{A}^k - 2\gamma - \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m a_{ij} a_{ij}^\top \in \mathbb{R}^{d \times d}$

$x^{k+1} = x^k - (\mathbf{H}^k + \mathbf{A}^k)^{-1} \left( \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^k) + \lambda x^k \right)$

$\mathbf{A}^{k+1} = \mathbf{A}^k + \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m (\eta \mathcal{C}_i^k(h_i(x^k) - h_i^k)) a_{ij} a_{ij}^\top$

**end for**

**Algorithm 1** NL1: NEWTON-LEARN ( $\lambda > 0$  case)

**Parameters:** learning rate  $\eta > 0$

**Initialization:**  $x^0 \in \mathbb{R}^d, h_i^0, \dots, h_n^0 \in \mathbb{R}_+^m, \mathbf{H}^0 = \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m h_{ij} a_{ij} a_{ij}^\top \in \mathbb{R}^{d \times d}$

**for each node**  $i = 1, \dots, n$  **do**

- Broadcast  $x^k$  to all workers
- for each node**  $j = 1, \dots, n$  **do**

  - Compute local gradient  $\nabla f_j(x^k)$
  - $h_{ij}^{k+1} = [h_{ij}^k + \eta \mathcal{C}_i^k(h_i(x^k) - h_i^k)]_+$
  - Send  $\nabla f_j(x^k)$  and  $\mathcal{C}_i^k(h_i(x^k) - h_i^k)$  to server

- Option 1:** Send  $\{a_{ij} : h_{ij}^{k+1} - h_{ij}^k \neq 0\}$  to server
- Option 2:** Do nothing if server knows  $\{a_{ij} : \forall j\}$

**end for**

$x^{k+1} = x^k - (\mathbf{H}^k + \mathbf{A}^k)^{-1} \left( \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^k) + \lambda x^k \right)$

$\mathbf{H}^{k+1} = \mathbf{H}^k + \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m (h_{ij}^{k+1} - h_{ij}^k) a_{ij} a_{ij}^\top$

**end for**

**Algorithm 1** MARINA

**1: Input:** starting point  $x^0$ , stepsize  $\gamma$ , minibatch size  $K \in [0, 1]$ , number of iterations  $K$

**2: Initialize**  $g^0 = \nabla f(x^0)$

**3: for**  $k = 0, 1, \dots, K - 1$  **do**

**4: for**  $i = 0, 1, \dots, K - 1$  **do**

**5: Sample**  $c_k \sim \text{Be}(p)$

**6: Broadcast**  $g^k$  to all workers

**7: for**  $i = 1, \dots, n$  **in parallel do**

**8:** $x^{k+1} = x^k - \gamma g^k$

**9:**Set  $g_i^{k+1} = \nabla f_i(x^{k+1})$  if  $c_k = 1$ , and  $g_i^{k+1} = g^k + \mathcal{Q}\left(\frac{1}{b} \sum_{j \in I_{i,k}^c} (\nabla f_{ij}(x^{k+1}) - \nabla f_{ij}(x^k))\right)$  otherwise, where  $I_{i,k}^c$  is the set of the indices in the minibatch,  $|I_{i,k}^c| = b'$

**9: end for**

**10:** $g^{k+1} = \frac{1}{n} \sum_{i=1}^n g_i^{k+1}$

**11: end for**

**12: Return:**  $\hat{x}^K$  chosen uniformly at random from  $\{x^k\}_{k=0}^{K-1}$

**Algorithm 2** VR-MARINA: finite sum case

**1: Input:** starting point  $x^0$ , stepsize  $\gamma$ , minibatch size  $b', p \in [0, 1]$ , number of iterations  $K$

**2: Initialize**  $g^0 = \nabla f(x^0)$

**3: for**  $k = 0, 1, \dots, K - 1$  **do**

**4: for**  $i = 0, 1, \dots, K - 1$  **do**

**5: Sample**  $c_k \sim \text{Be}(p)$

**6: Broadcast**  $g^k$  to all workers

**7: for**  $i = 1, \dots, n$  **in parallel do**

**8:** $x^{k+1} = x^k - \gamma g^k$

**9:**Set  $g_i^{k+1} = \nabla f_i(x^{k+1})$  if  $c_k = 1$ , and  $g_i^{k+1} = g^k + \mathcal{Q}\left(\frac{1}{b'} \sum_{j \in I_{i,k}^c} (\nabla f_{ij}(x^{k+1}) - \nabla f_{ij}(x^k))\right)$  otherwise, where  $I_{i,k}^c$  is the set of the indices in the minibatch,  $|I_{i,k}^c| = b'$

**9: end for**

**10:** $g^{k+1} = \frac{1}{n} \sum_{i=1}^n g_i^{k+1}$

**11: end for**

**12: Return:**  $\hat{x}^K$  chosen uniformly at random from  $\{x^k\}_{k=0}^{K-1}$

**Algorithm 1** DIANA with arbitrary unbiased quantization

**Input:** learning rates  $\alpha > 0$  and  $\gamma > 0$ , initial vectors  $x^0, h_0^0, \dots, h_n^0$  and  $h^0 = \frac{1}{n} \sum_{i=1}^n h_i^0$

**1 for**  $k = 0, 1, \dots$  **do**

**2 sample**  $x^k$  with probability  $\frac{1}{n}$

**3  $u^k = \begin{cases} 1, & \text{with probability } \frac{1}{n}, \\ 0, & \text{with probability } 1 - \frac{1}{n} \end{cases}$**

**4 broadcast**  $x^k$  to all workers

**5 for**  $i = 1, \dots, n$  **in parallel**  $\triangleright$  worker side

**6 sample**  $g_i^k$  such that  $E[g_i^k | x^k] = \nabla f_i(x^k)$

**7**  $\Delta_i^k = g_i^k - h_i^k$

**8**  $\hat{\Delta}_i^k = Q(\Delta_i^k)$

**9**  $h_i^{k+1} = h_i^k + \alpha \hat{\Delta}_i^k$

**10**  $\hat{g}_i^k = h_i^k + \Delta_i^k$

**11** **end for**

**12**  $g^k = \frac{1}{n} \sum_{i=1}^n \hat{g}_i^k \triangleright$  gather quantized updates

**13**  $\hat{g}^k = \frac{1}{n} \sum_{i=1}^n \hat{g}_i^k$

**14**  $x^{k+1} = x^k - (\mathbf{H}^k + \mathbf{A}^k)^{-1} \left( \frac{1}{n} \sum_{i=1}^n \nabla f_i(x^k) + \lambda x^k \right)$

**15**  $\mathbf{H}^{k+1} = \mathbf{H}^k + \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m (h_{ij}^{k+1} - h_{ij}^k) a_{ij} a_{ij}^\top$

**16** **end for**

**Algorithm 2** VR-DIANA based on L-SVRG (Variant 1), SAGA (Variant 2)

**Input:** learning rates  $\alpha > 0$  and  $\gamma > 0$ , initial vectors  $x^0, h_0^0, \dots, h_n^0$  and  $h^0 = \frac{1}{n} \sum_{i=1}^n h_i^0$

**1 for**  $k = 0, 1, \dots$  **do**

**2 sample**  $x^k$  with probability  $\frac{1}{n}$

**3  $u^k = \begin{cases} 1, & \text{with probability } \frac{1}{n}, \\ 0, & \text{with probability } 1 - \frac{1}{n} \end{cases}$**

**4 broadcast**  $x^k$  to all workers

**5 for**  $i = 1, \dots, n$  **do in parallel**  $\triangleright$  worker side

**6 if**  $u^k = 1$ , then  $a_{ij} = 1$  for all  $j \in [m]$

**7**  $\mu_i^k = \frac{1}{b} \sum_{j \in I_{i,k}} \nabla f_{ij}(x^k) + \mu_i^0$

**8**  $\Delta_i^k = \sum_{j \in I_{i,k}} a_{ij} \mu_j^k - \nabla f_{ij}(x^k) + \mu_i^k$

**9**  $\hat{\Delta}_i^k = Q(\Delta_i^k)$

**10**  $h_i^{k+1} = h_i^k + \alpha \hat{\Delta}_i^k$

**11** **end for**

**12**  $\hat{g}^k = \frac{1}{n} \sum_{i=1}^n \hat{g}_i^k \triangleright$  gather quantized updates

**13**  $\hat{g}^k = \frac{1}{n} \sum_{i=1}^n \hat{g}_i^k$

**14** **end for**

**Algorithm 1** Local fixed-point method

**Input:** Initial estimate  $\bar{x}^0 \in \mathbb{R}^d$ , stepsize  $\lambda > 0$ , sequence of synchronization times  $0 = t_0 < t_1 < \dots$

**Initialize:**  $x^0 = \bar{x}^0$ , for all  $i = 1, \dots, M$

**for**  $k = 1, 2, \dots, M$  **do**

**for**  $i = 1, 2, \dots, M$  in parallel **do**

**11**  $h_i^{k+1} := (1 - \lambda)x_i^k + \lambda T_i(x_i^k)$

**if**  $k + 1 = t_m$ , for some  $m \in \mathbb{N}$ , then

    Communicate  $h_i^{k+1}$  to master node

**else**

**12**  $x_i^{k+1} := h_i^{k+1}$

**end if**

**end for**

**if**  $k + 1 = t_m$ , for some  $m \in \mathbb{N}$ , then

    At master node:  $\bar{x}^{k+1} := \frac{1}{M} \sum_{i=1}^M h_i^{k+1}$

    Broadcast:  $x_i^{k+1} := \bar{x}^{k+1}$ , for all  $i = 1, \dots, M$

**end if**

**Algorithm 2** Randomized fixed-point method

**Input:** Initial estimate  $\bar{x}^0 \in \mathbb{R}^d$ , stepsize  $\lambda > 0$ , communication probability  $0 < p \leq 1$

**for**  $k = 0, 1, \dots$  **do**

**for**  $i = 1, 2, \dots, M$  in parallel **do**

**11**  $h_i^{k+1} := (1 - \lambda)x_i^k + \lambda T_i(x_i^k)$

**if**  $k + 1 = t_m$ , for some  $m \in \mathbb{N}$ , then

    Communicate  $h_i^{k+1}$  to master node

**else**

**12**  $x_i^{k+1} := h_i^{k+1}$

**end if**

**end for**

Flip a coin and **with probability**  $p$  **do**

    Communicate  $h_i^{k+1}$  to master, for all  $i$

At master node:  $\bar{x}^{k+1} := \frac{1}{M} \sum_{i=1}^M h_i^{k+1}$

Broadcast:  $x_i^{k+1} := \bar{x}^{k+1}$ , for all  $i = 1, \dots, M$

**end for**