FL_PyTorch

Optimization Research Simulator for Federated Learning

Konstantin Burlachenko

DistributedML 2021
2nd Workshop on Distributed Machine Learning,
co-located with CoNEXT 2021

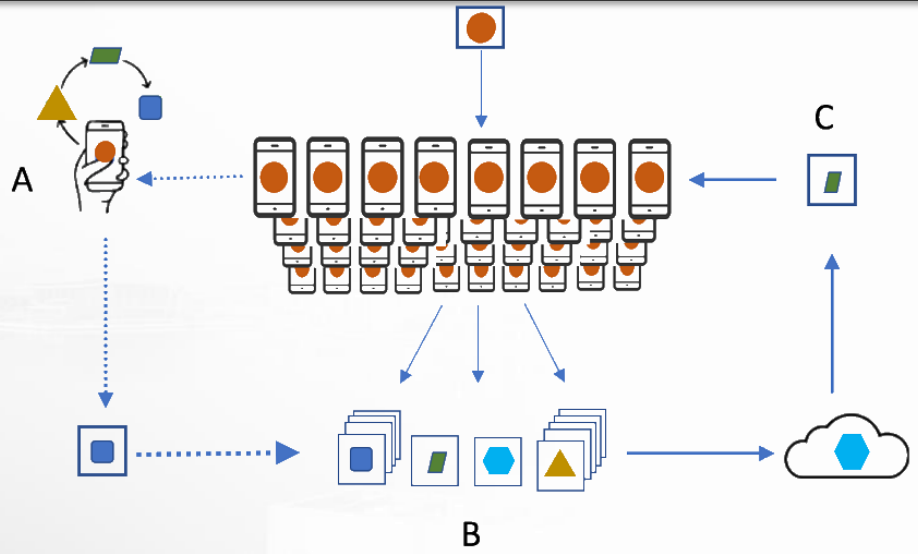
Konstantin Burlachenko
PhD student KAUST
konstantin.burlachenko@kaust.edu.sa

Samuel Horváth
PhD student KAUST
samuel.horvath@kaust.edu.sa

Peter Richtárik
Professor of Computer Science KAUST
peter.richtarik@kaust.edu.sa

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Federated Learning



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

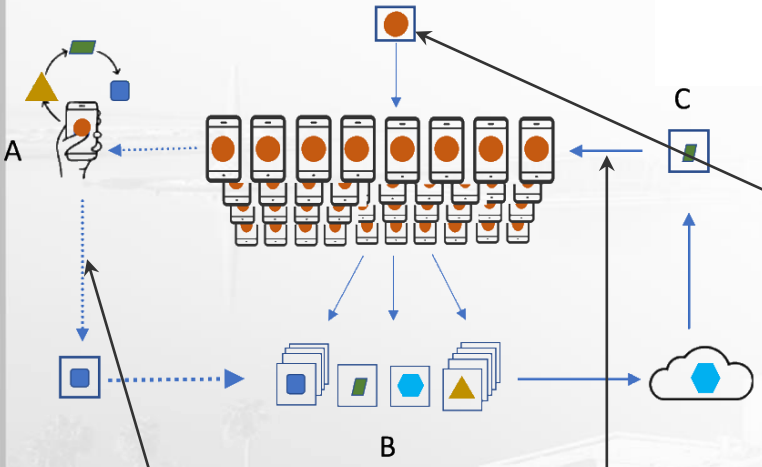
Some challenges: many devices; privacy; data heterogeneity; communication; partial participation

Federated Learning: Classical Objective

$$x^* = argmin_{x \in \mathbb{R}^d} \left(\sum_{i=1}^{N} w_i f_i(x) \right)$$

Number of clients (e.g., mobile phones)

f_i constructed from data of i-th client only



Communication is the bottleneck, f_j may be located in different parts of the world

Why do we need an optimization research simulator?

- Existing frameworks are mainly built with a focus to be deployed on real-world systems

Flower FEDML MAKING COLLABORATION INTELLIGENT TensorFlow

- 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

TensorFlow	TensorFlow Federated (TFF) is a framework for machine learning
PFL Paddle Federated Learning	Framework based on PaddlePaddle from BAIDU Research.
NVIDIA CLARA	Clara Training Framework developed by NVIDIA Research with a focus on the medical domain.
Flower	Framework targeted to bring Federated Learning into the real world and account for compute heterogeneity, memory constraints, and scaling aspects.
FEDML MAKING COLLABORATION INTELLIGENT	A valuable tool for benchmarking and experiments for Federated Learning.
Syft	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 

Hardware Utilization

Easy Debugging

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

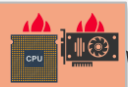
Our goals

Low Entry Bar/Simplicity 

We aim our tool to be as simple as possible

Extensibility 

We aim to provide universal abstractions

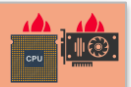
Hardware Utilization 

Easy Debugging

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

Our goals

- Low Entry Bar/ Simplicity 
- Extensibility 
- Hardware Utilization 
- Easy Debugging 

We aim to help to debug FL opt. algorithms

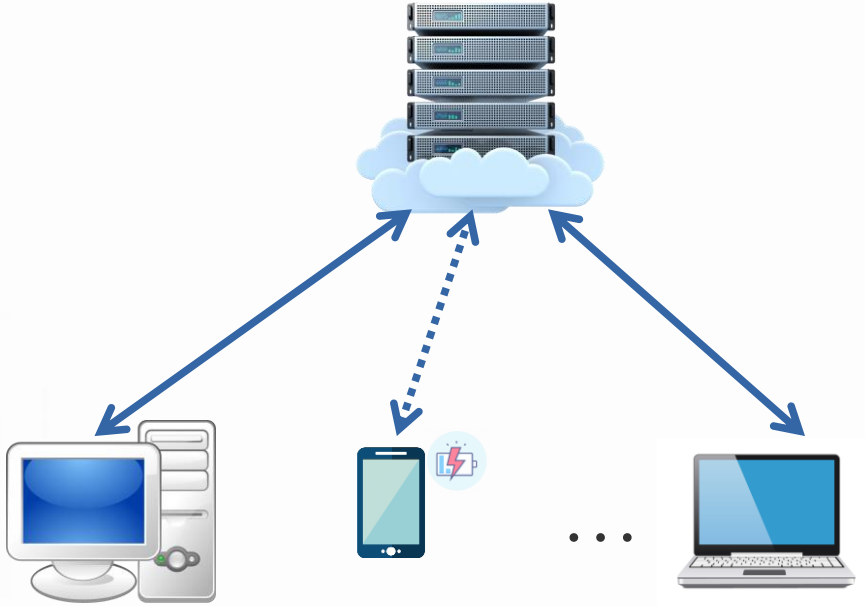
1. The user provides only a single-node implementation
2. Mapping to multi-GPU/CPU and multi-node setup happens during runtime configuration only
3. GUI/CUI provides necessary logging information
4. Means to reproduce the experiments

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 $\boldsymbol{x}^{(0)}$, ClientOpt, ServerOpt
Initialize server state $H^0 = \text{InitializeServerState}()$
for $t \in \{0,1,\ldots,T-1\}$ **do**
  Sample a subset $\mathcal{S}^{(t)}$ of available clients
  Generate state: $s^{(t)} = \text{ClientState}(H^{(t)})$
  Broadcast $(\boldsymbol{x}^{(t)}, s^{(t)})$ to workers
  for client $i \in \mathcal{S}^{(t)}$ **in parallel do**
    Initialize local model $\boldsymbol{x}_i^{(t,0)} = \boldsymbol{x}^{(t)}$
    for $k = 0,\ldots,\tau_i - 1$ **do**
      Compute local stochastic gradient $g_i = \text{LocalGradient}(\boldsymbol{x}_i^{(t,k)}, s_t)$
      Perform local update $\boldsymbol{x}_i^{(t,k+1)} = \text{ClientOpt}(\boldsymbol{x}_i^{(t,k)}, g_i, k, t)$
    end
    Compute local model changes $\Delta_i^{(t)} = \boldsymbol{x}_i^{(t,\tau_i)} - \boldsymbol{x}_i^{(t,0)}$
    Create local state update: $U_i^{(t)} = \text{LocalState}(\boldsymbol{x}^{(t)}, \boldsymbol{x}_i^{(t,\tau_i)})$
    Send $(\Delta_i^{(t)}, U_i^{(t)})$ to Master.
  end
  Obtain $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$.
  Compute $G^{(t)} = \text{ServerGradient}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)})$
  Update global model $\boldsymbol{x}^{(t+1)} = \text{ServerOpt}(\boldsymbol{x}^{(t)}, G^{(t)}, \eta_s, t)$
  Update: $H^{(t+1)} = \text{ServerGlobalState}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)})$
end

FL_PyTorch

- Written in Python language only
- Build upon PyTorch abstractions

Generalized Federated Averaging

Input: Initial model $\boldsymbol{x}^{(0)}$, ClientOpt, ServerOpt
Initialize server state $H^0 = \text{InitializeServerState}()$
for $t \in \{0,1,\ldots,T-1\}$ **do**
  Sample a subset $\mathcal{S}^{(t)}$ of available clients
  Generate state: $s^{(t)} = \text{ClientState}(H^{(t)})$
  Broadcast $(\boldsymbol{x}^{(t)}, s^{(t)})$ to workers
  for client $i \in \mathcal{S}^{(t)}$ **in parallel do**
    Initialize local model $\boldsymbol{x}_i^{(t,0)} = \boldsymbol{x}^{(t)}$
    for $k = 0,\ldots,\tau_i - 1$ **do**
      Compute local stochastic gradient $g_i = \text{LocalGradient}(\boldsymbol{x}_i^{(t,k)}, s_t)$
      Perform local update $\boldsymbol{x}_i^{(t,k+1)} = \text{ClientOpt}(\boldsymbol{x}_i^{(t,k)}, g_i, k, t)$
    end
    Compute local model changes $\Delta_i^{(t)} = \boldsymbol{x}_i^{(t,\tau_i)} - \boldsymbol{x}_i^{(t,0)}$
    Create local state update: $U_i^{(t)} = \text{LocalState}(\boldsymbol{x}^{(t)}, \boldsymbol{x}_i^{(t,\tau_i)})$
    Send $(\Delta_i^{(t)}, U_i^{(t)})$ to Master.
  end
  Obtain $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$.
  Compute $G^{(t)} = \text{ServerGradient}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)})$
  Update global model $\boldsymbol{x}^{(t+1)} = \text{ServerOpt}(\boldsymbol{x}^{(t)}, G^{(t)}, \eta_s, t)$
  Update: $H^{(t+1)} = \text{ServerGlobalState}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)})$
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 $\boldsymbol{x}^{(0)}$, ClientOpt, ServerOpt
Initialize server state $H^0 = \text{InitializeServerState}()$
for $t \in \{0,1,\ldots,T-1\}$ **do**
  Sample a subset $\mathcal{S}^{(t)}$ of available clients
  Generate state: $s^{(t)} = \text{ClientState}(H^{(t)})$
  Broadcast $(\boldsymbol{x}^{(t)}, s^{(t)})$ to workers
  for client $i \in \mathcal{S}^{(t)}$ **in parallel do**
    Initialize local model $\boldsymbol{x}_i^{(t,0)} = \boldsymbol{x}^{(t)}$
    for $k = 0,\ldots,\tau_i - 1$ **do**
      Compute local stochastic gradient $g_i = \text{LocalGradient}(\boldsymbol{x}_i^{(t,k)}, s_t)$
      Perform local update $\boldsymbol{x}_i^{(t,k+1)} = \text{ClientOpt}(\boldsymbol{x}_i^{(t,k)}, g_i, k, t)$
    end
    Compute local model changes $\Delta_i^{(t)} = \boldsymbol{x}_i^{(t,\tau_i)} - \boldsymbol{x}_i^{(t,0)}$
    Create local state update: $U_i^{(t)} = \text{LocalState}(\boldsymbol{x}^{(t)}, \boldsymbol{x}_i^{(t,\tau_i)})$
    Send $(\Delta_i^{(t)}, U_i^{(t)})$ to Master.
  end
  Obtain $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$.
  Compute $G^{(t)} = \text{ServerGradient}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)})$
  Update global model $\boldsymbol{x}^{(t+1)} = \text{ServerOpt}(\boldsymbol{x}^{(t)}, G^{(t)}, \eta_s, t)$
  Update: $H^{(t+1)} = \text{ServerGlobalState}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)})$
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 $\boldsymbol{x}^{(0)}$, ClientOpt, ServerOpt
Initialize server state H^0 = InitializeServerState()
for $t \in \{0,1,\ldots,T-1\}$ **do**
  Sample a subset $\mathcal{S}^{(t)}$ of available clients
  Generate state: $s^{(t)}$ = ClientState($H^{(t)}$)
  Broadcast ($\boldsymbol{x}^{(t)}$,$s^{(t)}$) to workers
  for client $i \in \mathcal{S}^{(t)}$ **in parallel do**
    Initialize local model $\boldsymbol{x}_i^{(t,0)} = \boldsymbol{x}^{(t)}$
    for $k = 0,\ldots,\tau_i - 1$ **do**
      Compute local stochastic gradient g_i = LocalGradient($\boldsymbol{x}_i^{(t,k)}, s_t$)
      Perform local update $\boldsymbol{x}_i^{(t,k+1)}$ = ClientOpt($\boldsymbol{x}_i^{(t,k)}, g_i, k, t$)
    end
    Compute local model changes $\Delta_i^{(t)} = \boldsymbol{x}_i^{(t,\tau_i)} - \boldsymbol{x}_i^{(t,0)}$
    Create local state update: $U_i^{(t)}$ = LocalState($\boldsymbol{x}^{(t)}, \boldsymbol{x}_i^{(t,\tau_i)}$)
    Send ($\Delta_i^{(t)}$,$U_i^{(t)}$) to Master.
  end
  Obtain ($\Delta_i^{(t)}$,$U_i^{(t)}$), $\forall i \in \mathcal{S}^{(t)}$.
  Compute $G^{(t)}$ = ServerGradient($\{\Delta_i^{(t)},U_i^{(t)}\}_{i\in S^{(t)}}, H^{(t)}$)
  Update global model $\boldsymbol{x}^{(t+1)}$ = ServerOpt($\boldsymbol{x}^{(t)}, G^{(t)},\eta_s,t$)
  Update: $H^{(t+1)}$ = ServerGlobalState($\{\Delta_i^{(t)},U_i^{(t)}\}_{i\in S^{(t)}}, H^{(t)}$)
end

LocalGradient

Obtain the algorithm specific local gradient estimator g_i .

Generalized Federated Averaging

Input: Initial model $\boldsymbol{x}^{(0)}$, ClientOpt, ServerOpt
Initialize server state $H^0 = \text{InitializeServerState}()$
for $t \in \{0,1,\ldots,T-1\}$ **do**
  Sample a subset $\mathcal{S}^{(t)}$ of available clients
  Generate state: $s^{(t)} = \text{ClientState}(H^{(t)})$
  Broadcast $(\boldsymbol{x}^{(t)}, s^{(t)})$ to workers
  for client $i \in \mathcal{S}^{(t)}$ **in parallel do**
    Initialize local model $\boldsymbol{x}_i^{(t,0)} = \boldsymbol{x}^{(t)}$
    for $k = 0,\ldots,\tau_i - 1$ **do**
      Compute local stochastic gradient $g_i = \text{LocalGradient}(\boldsymbol{x}_i^{(t,k)}, s_t)$
      Perform local update $\boldsymbol{x}_i^{(t,k+1)} = \text{ClientOpt}(\boldsymbol{x}_i^{(t,k)}, g_i, k, t)$
    end
    Compute local model changes $\Delta_i^{(t)} = \boldsymbol{x}_i^{(t,\tau_i)} - \boldsymbol{x}_i^{(t,0)}$
    Create local state update: $U_i^{(t)} = \text{LocalState}(\boldsymbol{x}^{(t)}, \boldsymbol{x}_i^{(t,\tau_i)})$
    Send $(\Delta_i^{(t)}, U_i^{(t)})$ to Master.
  end
  Obtain $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$.
  Compute $G^{(t)} = \text{ServerGradient}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)})$
  Update global model $\boldsymbol{x}^{(t+1)} = \text{ServerOpt}(\boldsymbol{x}^{(t)}, G^{(t)}, \eta_s, t)$
  Update: $H^{(t+1)} = \text{ServerGlobalState}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)})$
end

ClientOpt

Local optimization step using the obtained direction g_i.

Generalized Federated Averaging

Input: Initial model $\boldsymbol{x}^{(0)}$, ClientOpt, ServerOpt
Initialize server state H^0 = InitializeServerState()
for $t \in \{0,1,\ldots,T-1\}$ **do**
  Sample a subset $\mathcal{S}^{(t)}$ of available clients
  Generate state: $s^{(t)}$ = ClientState($H^{(t)}$)
  Broadcast ($\boldsymbol{x}^{(t)}$,$s^{(t)}$) to workers
  for client $i \in \mathcal{S}^{(t)}$ **in parallel do**
    Initialize local model $\boldsymbol{x}_i^{(t,0)} = \boldsymbol{x}^{(t)}$
    for $k = 0,\ldots,\tau_i - 1$ **do**
      Compute local stochastic gradient g_i = LocalGradient($\boldsymbol{x}_i^{(t,k)}, s_t$)
      Perform local update $\boldsymbol{x}_i^{(t,k+1)}$ = ClientOpt($\boldsymbol{x}_i^{(t,k)}, g_i, k, t$)
    end
    Compute local model changes $\Delta_i^{(t)} = \boldsymbol{x}_i^{(t,\tau_i)} - \boldsymbol{x}_i^{(t,0)}$
    Create local state update: $U_i^{(t)}$ = LocalState($\boldsymbol{x}^{(t)}, \boldsymbol{x}_i^{(t,\tau_i)}$)
    Send ($\Delta_i^{(t)}$,$U_i^{(t)}$) to Master.
  end
  Obtain ($\Delta_i^{(t)}$,$U_i^{(t)}$), $\forall i \in \mathcal{S}^{(t)}$.
  Compute $G^{(t)}$ = ServerGradient($\{\Delta_i^{(t)},U_i^{(t)}\}_{i\in S^{(t)}}, H^{(t)}$)
  Update global model $\boldsymbol{x}^{(t+1)}$ = ServerOpt($\boldsymbol{x}^{(t)}, G^{(t)},\eta_s,t$)
  Update: $H^{(t+1)}$ = ServerGlobalState($\{\Delta_i^{(t)},U_i^{(t)}\}_{i\in S^{(t)}}, H^{(t)}$)
end

LocalState

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

Generalized Federated Averaging

Input: Initial model $\boldsymbol{x}^{(0)}$, ClientOpt, ServerOpt
Initialize server state H^0 = InitializeServerState()
for $t \in \{0,1,\ldots,T-1\}$ **do**
  Sample a subset $\mathcal{S}^{(t)}$ of available clients
  Generate state: $s^{(t)}$ = ClientState($H^{(t)}$)
  Broadcast ($\boldsymbol{x}^{(t)}$,$s^{(t)}$) to workers
  for client $i \in \mathcal{S}^{(t)}$ **in parallel do**
    Initialize local model $\boldsymbol{x}_i^{(t,0)} = \boldsymbol{x}^{(t)}$
    for $k = 0,\ldots,\tau_i - 1$ **do**
      Compute local stochastic gradient g_i = LocalGradient($\boldsymbol{x}_i^{(t,k)}, s_t$)
      Perform local update $\boldsymbol{x}_i^{(t,k+1)}$ = ClientOpt($\boldsymbol{x}_i^{(t,k)}, g_i, k, t$)
    end
    Compute local model changes $\Delta_i^{(t)} = \boldsymbol{x}_i^{(t,\tau_i)} - \boldsymbol{x}_i^{(t,0)}$
    Create local state update: $U_i^{(t)}$ = LocalState($\boldsymbol{x}^{(t)}, \boldsymbol{x}_i^{(t,\tau_i)}$)
    Send ($\Delta_i^{(t)}$,$U_i^{(t)}$) to Master.
  end
  Obtain ($\Delta_i^{(t)}$,$U_i^{(t)}$), $\forall i \in \mathcal{S}^{(t)}$.
  Compute $G^{(t)}$ = ServerGradient($\{\Delta_i^{(t)},U_i^{(t)}\}_{i\in S^{(t)}}, H^{(t)}$)
  Update global model $\boldsymbol{x}^{(t+1)}$ = ServerOpt($\boldsymbol{x}^{(t)}, G^{(t)},\eta_s,t$)
  Update: $H^{(t+1)}$ = ServerGlobalState($\{\Delta_i^{(t)},U_i^{(t)}\}_{i\in S^{(t)}}, H^{(t)}$)
end

ServerGradient

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

Generalized Federated Averaging

Input: Initial model $\boldsymbol{x}^{(0)}$, ClientOpt, ServerOpt
Initialize server state H^0 = InitializeServerState()
for $t \in \{0,1,\ldots,T-1\}$ **do**
  Sample a subset $\mathcal{S}^{(t)}$ of available clients
  Generate state: $s^{(t)}$ = ClientState($H^{(t)}$)
  Broadcast $(\boldsymbol{x}^{(t)}, s^{(t)})$ to workers
  for client $i \in \mathcal{S}^{(t)}$ **in parallel do**
    Initialize local model $\boldsymbol{x}_i^{(t,0)} = \boldsymbol{x}^{(t)}$
    for $k = 0,\ldots,\tau_i - 1$ **do**
      Compute local stochastic gradient g_i = LocalGradient($\boldsymbol{x}_i^{(t,k)}, s_t$)
      Perform local update $\boldsymbol{x}_i^{(t,k+1)}$ = ClientOpt($\boldsymbol{x}_i^{(t,k)}, g_i, k, t$)
    end
    Compute local model changes $\Delta_i^{(t)} = \boldsymbol{x}_i^{(t,\tau_i)} - \boldsymbol{x}_i^{(t,0)}$
    Create local state update: $U_i^{(t)}$ = LocalState($\boldsymbol{x}^{(t)}, \boldsymbol{x}_i^{(t,\tau_i)}$)
    Send $(\Delta_i^{(t)}, U_i^{(t)})$ to Master.
  end
  Obtain $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$.
  Compute $G^{(t)}$ = ServerGradient($\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)}$)
  Update global model $\boldsymbol{x}^{(t+1)}$ = ServerOpt($\boldsymbol{x}^{(t)}, G^{(t)}, \eta_s, t$)
  Update: $H^{(t+1)}$ = ServerGlobalState($\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)}$)
end

ServerOpt

Server optimization step using the obtained direction G^t.

Generalized Federated Averaging

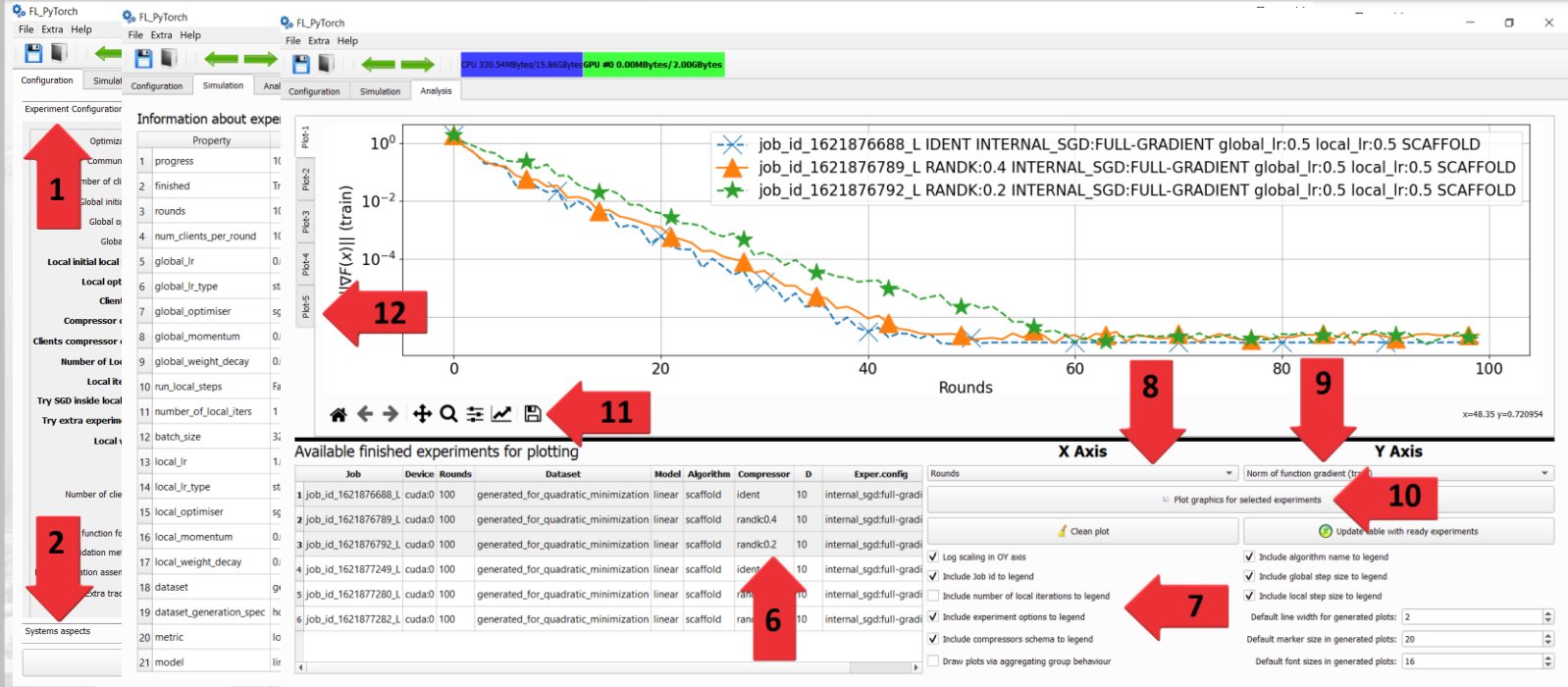
Input: Initial model $\boldsymbol{x}^{(0)}$, ClientOpt, ServerOpt
Initialize server state $H^0 = \text{InitializeServerState}()$
for $t \in \{0,1,\ldots,T-1\}$ **do**
  Sample a subset $\mathcal{S}^{(t)}$ of available clients
  Generate state: $s^{(t)} = \text{ClientState}(H^{(t)})$
  Broadcast $(\boldsymbol{x}^{(t)}, s^{(t)})$ to workers
  for client $i \in \mathcal{S}^{(t)}$ **in parallel do**
    Initialize local model $\boldsymbol{x}_i^{(t,0)} = \boldsymbol{x}^{(t)}$
    for $k = 0,\ldots,\tau_i - 1$ **do**
      Compute local stochastic gradient $g_i = \text{LocalGradient}(\boldsymbol{x}_i^{(t,k)}, s_t)$
      Perform local update $\boldsymbol{x}_i^{(t,k+1)} = \text{ClientOpt}(\boldsymbol{x}_i^{(t,k)}, g_i, k, t)$
    end
    Compute local model changes $\Delta_i^{(t)} = \boldsymbol{x}_i^{(t,\tau_i)} - \boldsymbol{x}_i^{(t,0)}$
    Create local state update: $U_i^{(t)} = \text{LocalState}(\boldsymbol{x}^{(t)}, \boldsymbol{x}_i^{(t,\tau_i)})$
    Send $(\Delta_i^{(t)}, U_i^{(t)})$ to Master.
  end
  Obtain $(\Delta_i^{(t)}, U_i^{(t)}), \forall i \in \mathcal{S}^{(t)}$.
  Compute $G^{(t)} = \text{ServerGradient}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)})$
  Update global model $\boldsymbol{x}^{(t+1)} = \text{ServerOpt}(\boldsymbol{x}^{(t)}, G^{(t)}, \eta_s, t)$
  Update: $H^{(t+1)} = \text{ServerGlobalState}(\{\Delta_i^{(t)}, U_i^{(t)}\}_{i \in S^{(t)}}, H^{(t)})$
end

ServerGlobalState

The global server state update

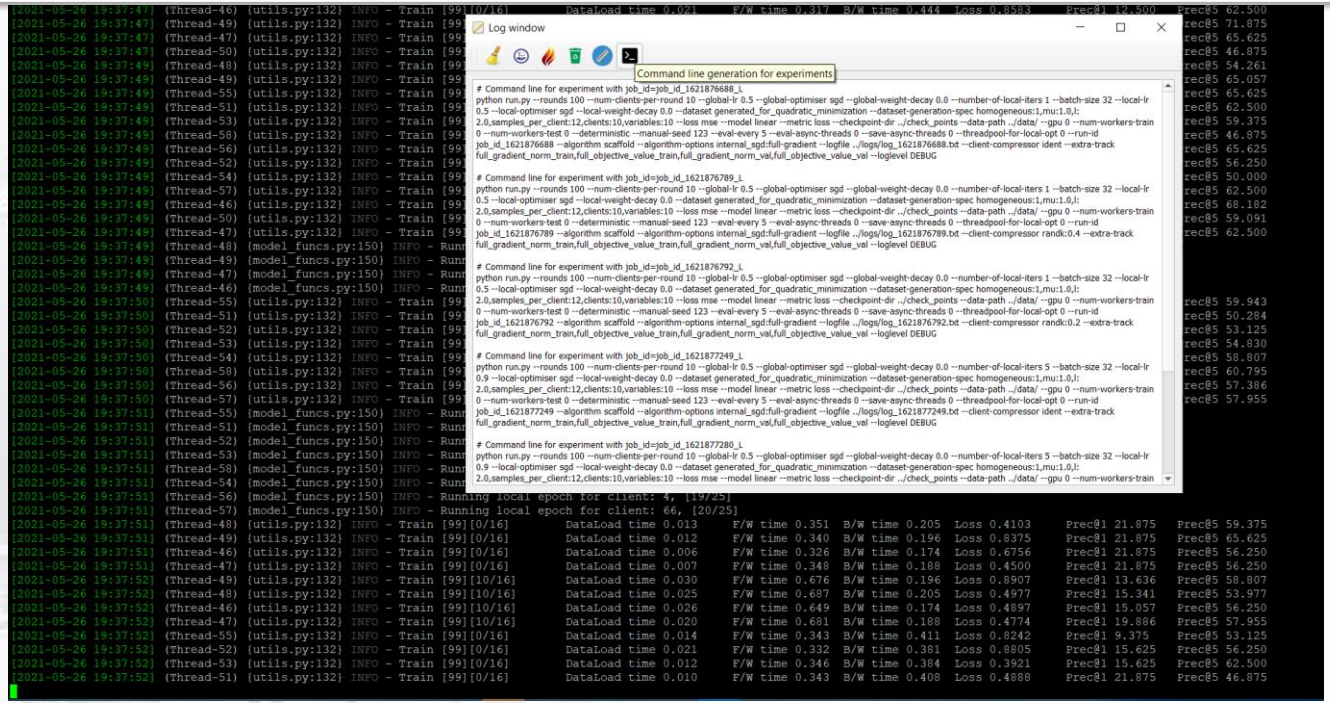
FL Optimization Research Simulator: UI

Graphical UI



FL Optimization Research Simulator: UI

Console UI



FL Optimization Research Simulator

Graphical UI

Console UI

Configuration

FL Optimization Research Simulator

Graphical UI

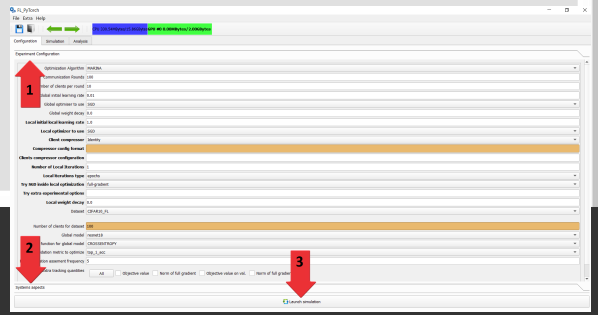
Console UI

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

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

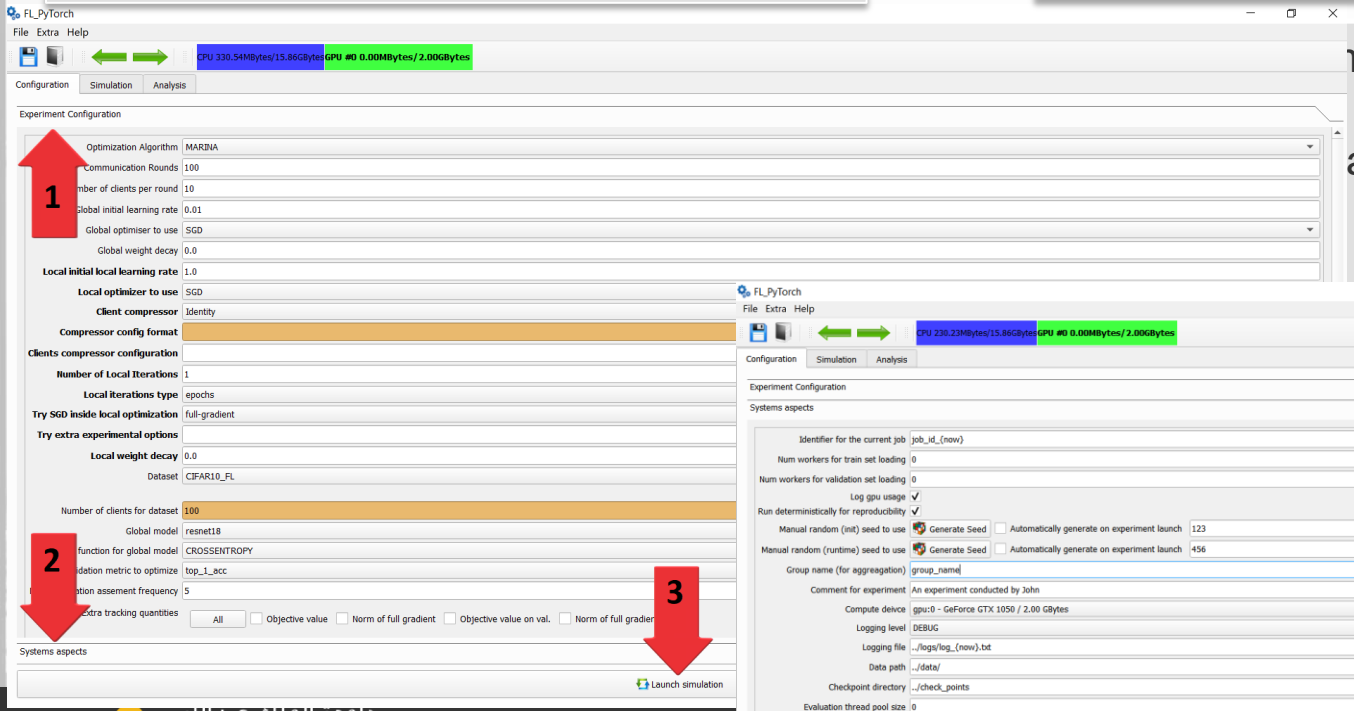
Model and Data: models' and datasets' names.

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



FL Optimization Research Simulator

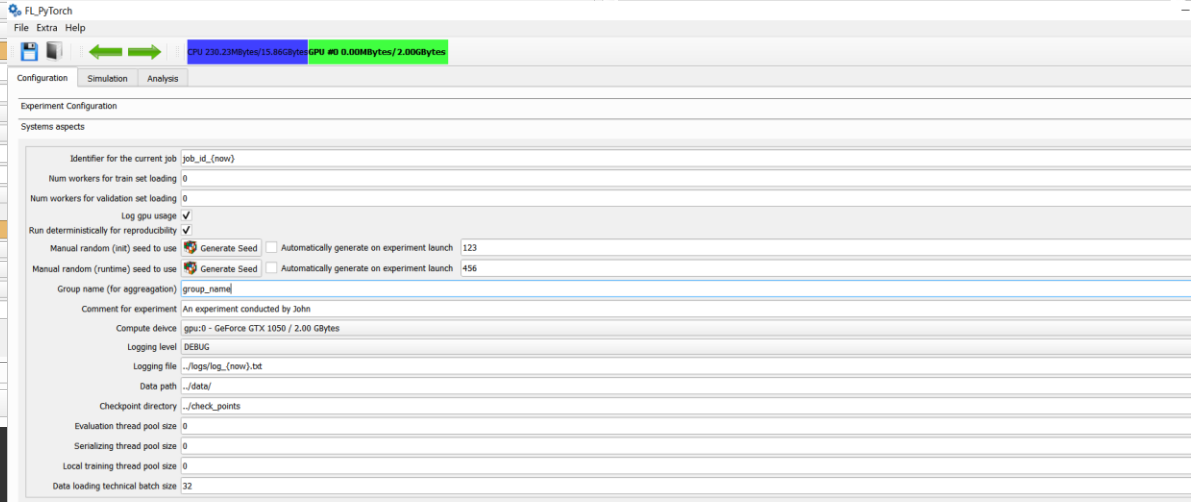
Graphical UI



Console UI

nber of sampled clients etc.

atch size, etc.



FL Optimization Research Simulator

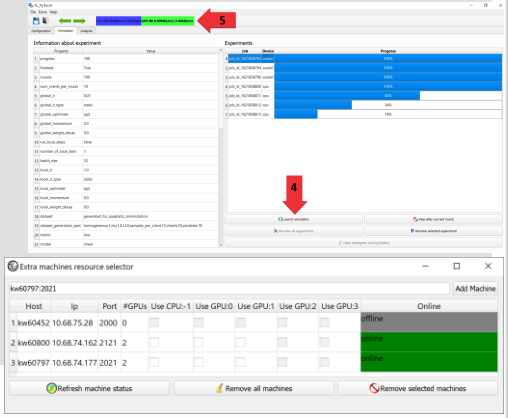
Graphical UI

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.



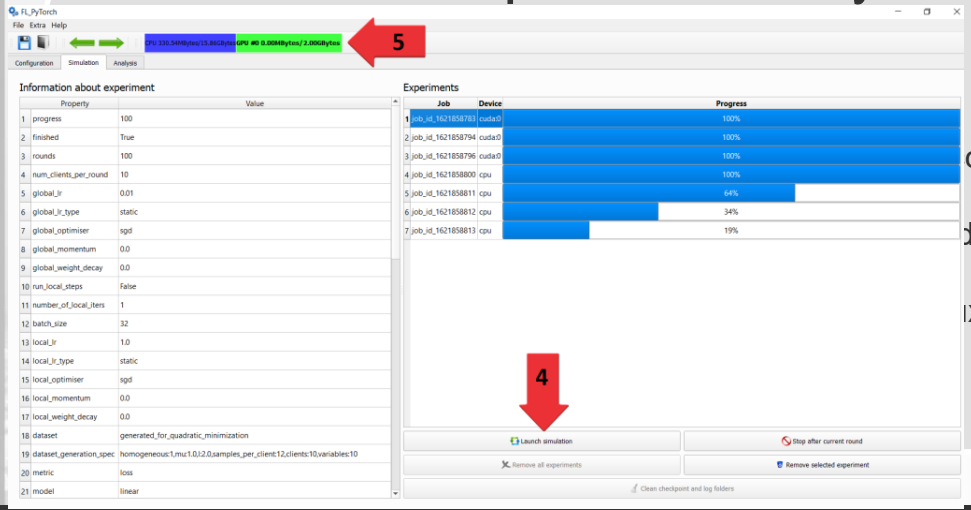
FL Optimization Research Simulator

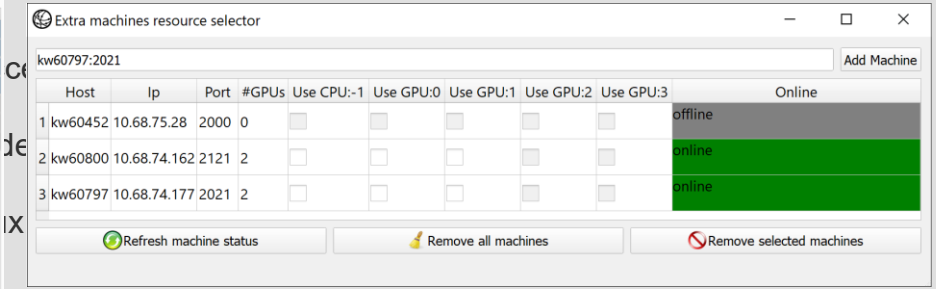
Graphical UI

Console UI

Configuration: command line or actions in GUI

Launch experiments: locally or remotely, in specified target devices





FL Optimization Research Simulator

Graphical UI

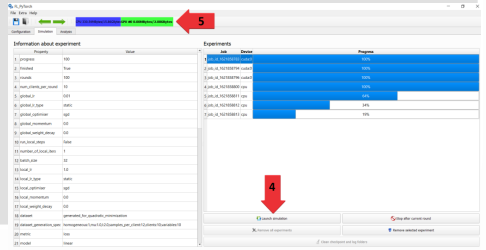
Console UI

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

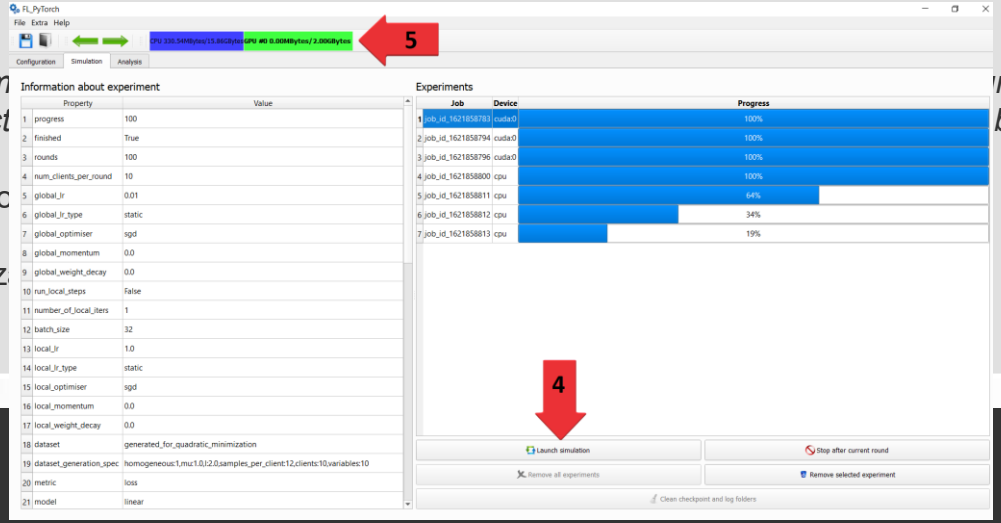
Graphical UI

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* ... *radient, the norm of the full objective gradient, funct* ... *bits.*
- Visualize the progress c
- Monitor CPU/GPU utiliz

FL Optimization Research Simulator

Graphical UI

Console UI

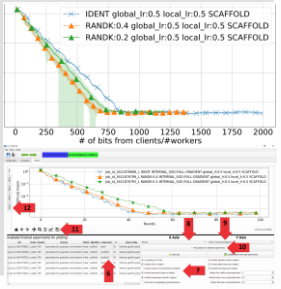
Configuration: command line or actions in GUI

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
- Save and load experimental results
- Recover arguments passed via the GUI tool in the command line format



FL Optimization Research Simulator

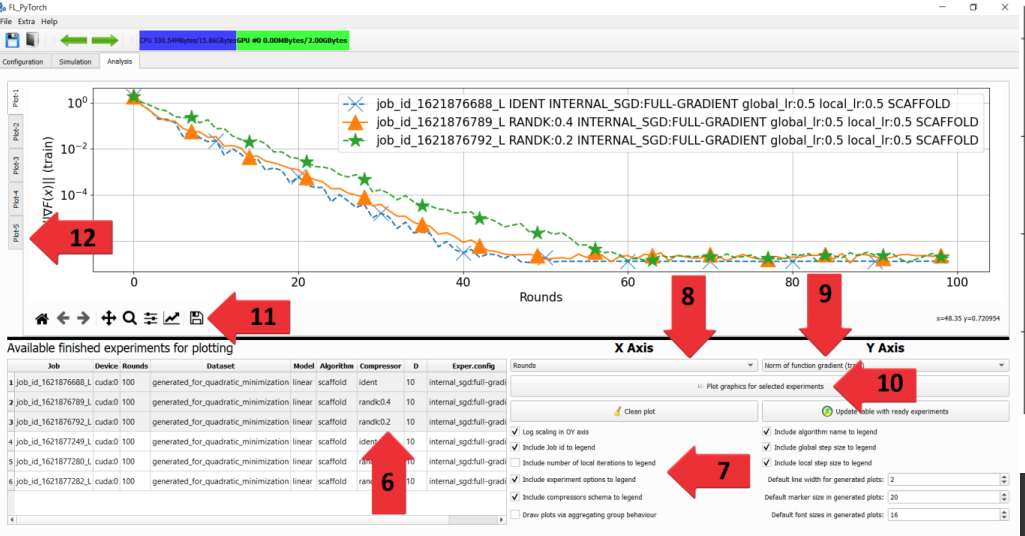
Graphical UI

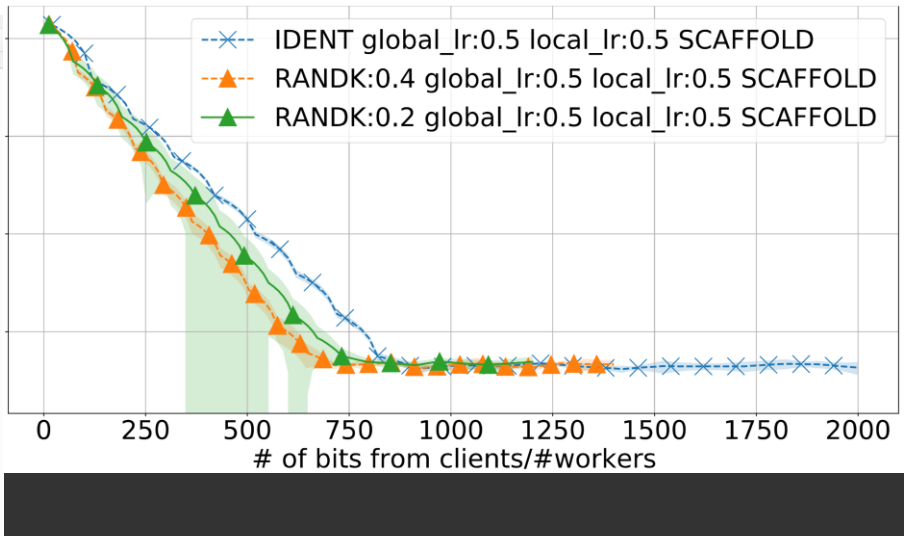
Console UI

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

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

Models and Datasets

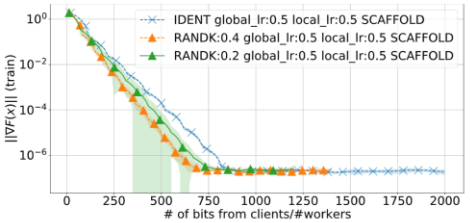
- ResNet(18,34,50,101,152), VGG(11,13,16,19)
- WideResNets (28_2, 28_4, 28_8)
- Support for simple controllable
- Quadratic functions as toy problems

- Standard FL datasets
- Synthetically generated

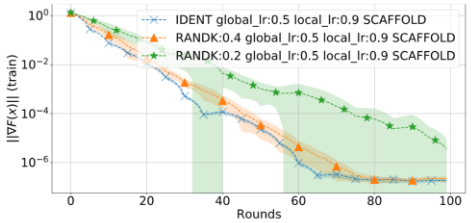
Experiment example

In the paper, we provide several experiments to showcase potential use cases of FL_PyTorch

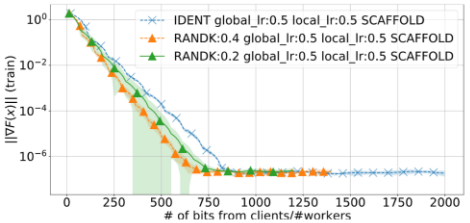
We used our framework to analyze whether SCAFFOLD can work with compression, i.e. whether messages from clients in the SCAFFOLD algorithm can be compressed.
In synthetized quadratic optimization problem experiments demonstrated that it might be worthwhile to consider an extension of SCAFFOLD by adding compression.



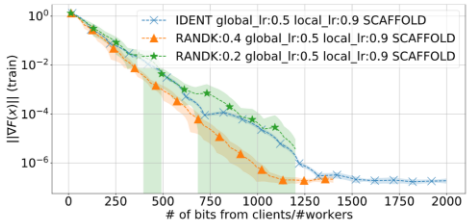
(a) One local iteration, convergence in gradient.



(b) Five local iteration,convergence in gradient.



(c) One local iteration, average communication.



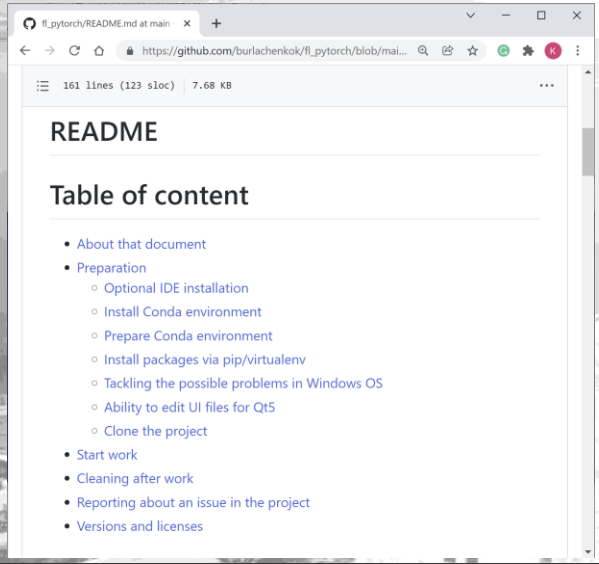
(d) Five local iterations, average communication.

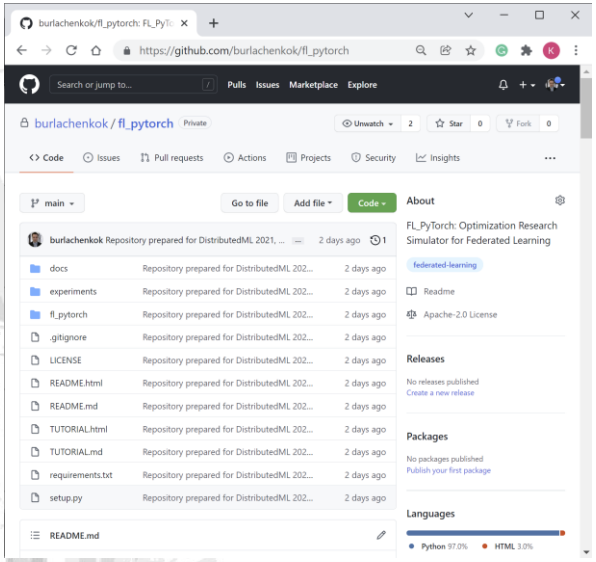
More Information

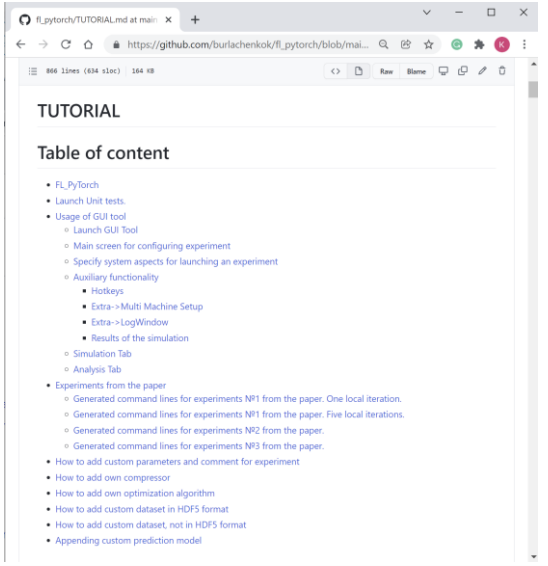
Please take a look at the original paper for motivation and goals.

Git repository: https://github.com/burlachenkok/fl_pytorch.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

https://burlachenkok.github.io/

FL_PyTorch: Optimization Research Simulator for Federated Learning

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

DISTRIBUTEDML


جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology
Extra Slides

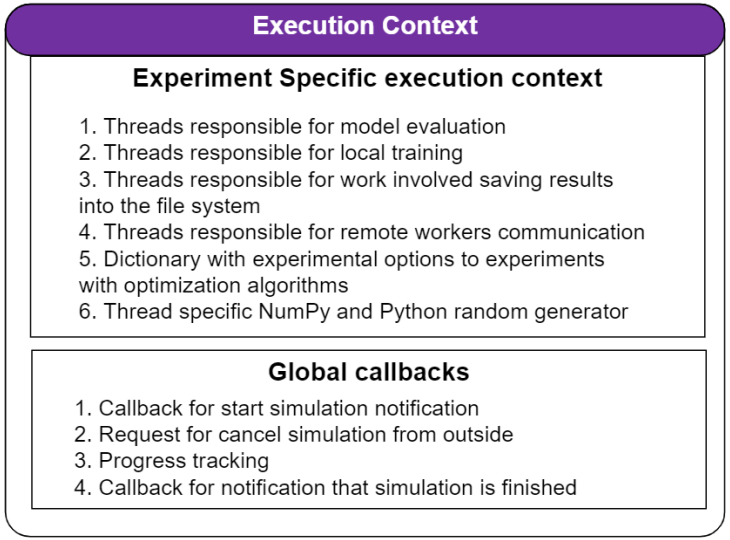
Speeds of some buses

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

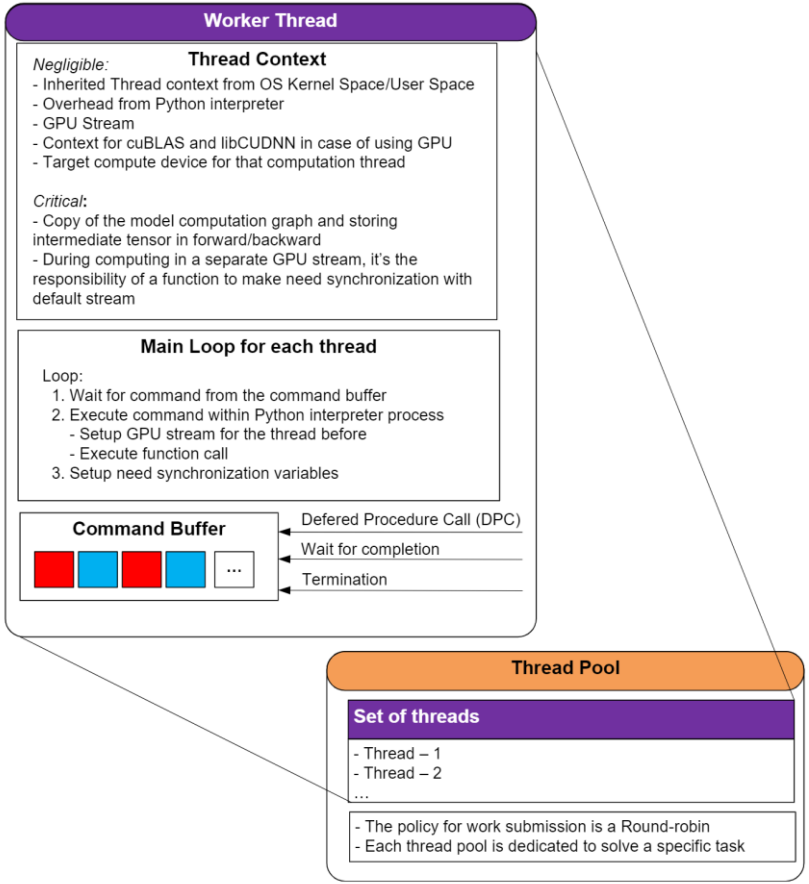
Memory Transferring direction within a mult. nodes (PCs)	Typical Speed
Fast Ethernet	12.5 MBytes/sec
Gigabit Ethernet	125 MBytes/sec
InfiniBand HDR	6250 Mbytes/sec
InfiniBand Melanox	25 000 Mbytes/sec

NVIDIA GPU Software Compute Ecosystem

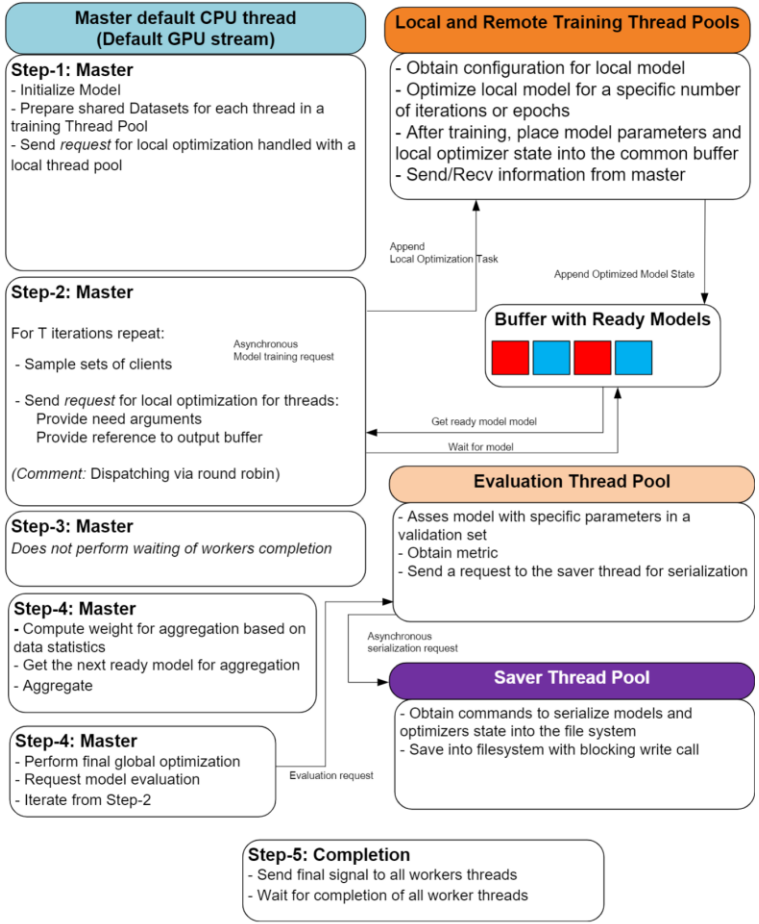
Application from specific domains									
Consumer Applications (recommendation systems)			Industrial Applications (Manufacture, Finance)				Scientific Applications (Molecular Simulation, Seismatics, etc.)		
Frameworks and Application									
MXNET	Chainer	PyTorch	TensorFlow	RAPIDS	Caffe	OCTAVE	Matlab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



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)

Parameters: learning rate $\gamma > 0$, starting point $x^0 \in \mathbb{R}^d$, compression operators $\mathcal{C}_1 \in \mathbb{B}^d(\omega_1), \ldots, \mathcal{C}_n \in \mathbb{B}^d(\omega_n), \mathcal{C} \in \mathbb{B}^d(\omega_M)$

for $k = 0, 1, 2, \ldots$ **do**
  for all workers $i \in \{1, 2, \ldots, n\}$ **in parallel do**
    Compute local gradient $\nabla f_i(x^k)$
    Compress local gradient $g_i^k = \mathcal{C}_i^k(\nabla f_i(x^k))$
    Send g_i^k to master
  Master computes the aggregate $\hat{g}^k = \frac{1}{n}\sum_{i=1}^n g_i^k$
  Master broadcasts the compressed aggregate $g^k = \mathcal{C}^k(\hat{g}^k)$ to all workers
  for all workers $i \in \{1, 2, \ldots, n\}$ **in parallel do**
    Compute the next iterate $x^{k+1} = \text{prox}_{\gamma R}(x^k - \gamma g^k)$

Algorithm 1 SCAFFOLD: Stochastic Controlled Averaging for federated learning

server input: initial $\boldsymbol{x}$ and $\boldsymbol{c}$, and global step-size η_g
client i's input: $\boldsymbol{c}_i$, and local step-size η_l
for each round $r = 1, \ldots, R$ **do**
  sample clients $\mathcal{S} \subseteq \{1, \ldots, N\}$
  communicate $(\boldsymbol{x}, \boldsymbol{c})$ to all clients $i \in \mathcal{S}$
  on client $i \in \mathcal{S}$ **in parallel do**
    initialize local model $\boldsymbol{y}_i \leftarrow \boldsymbol{x}$
    for $k = 1, \ldots, K$ **do**
      compute mini-batch gradient $g_i(\boldsymbol{y}_i)$
      $\boldsymbol{y}_i \leftarrow \boldsymbol{y}_i - \eta_l\,(g_i(\boldsymbol{y}_i) - \boldsymbol{c}_i + \boldsymbol{c})$
    end for
    $\boldsymbol{c}_i^+ \leftarrow$ (i) $g_i(\boldsymbol{x})$, or (ii) $\boldsymbol{c}_i - \boldsymbol{c} + \frac{1}{K\eta_l}(\boldsymbol{x} - \boldsymbol{y}_i)$
    communicate $(\Delta \boldsymbol{y}_i, \Delta \boldsymbol{c}_i) \leftarrow (\boldsymbol{y}_i - \boldsymbol{x}, \boldsymbol{c}_i^+ - \boldsymbol{c}_i)$
    $\boldsymbol{c}_i \leftarrow \boldsymbol{c}_i^+$
  end on client
  $(\Delta \boldsymbol{x}, \Delta \boldsymbol{c}) \leftarrow \frac{1}{|\mathcal{S}|}\sum_{i \in \mathcal{S}}(\Delta \boldsymbol{y}_i, \Delta \boldsymbol{c}_i)$
  $\boldsymbol{x} \leftarrow \boldsymbol{x} + \eta_g \Delta \boldsymbol{x}$ and $\boldsymbol{c} \leftarrow \boldsymbol{c} + \frac{|\mathcal{S}|}{N}\Delta \boldsymbol{c}$
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 $\eta, \theta_1, \theta_2, \alpha, \beta, \gamma, p$
$z^0 = y^0 = w^0 = x^0$
for $k = 0, 1, 2, \ldots$ **do**
  $x^k = \theta_1 z^k + \theta_2 w^k + (1 - \theta_1 - \theta_2) y^k$
  for all machines $i = 1, 2, \ldots, n$ **do in parallel**
    Compress shifted local gradient $\mathcal{C}_i^k(\nabla f_i(x^k) - h_i^k)$ and send to the server
    Update local shift $h_i^{k+1} = h_i^k + \alpha \mathcal{C}_i^k(\nabla f_i(w^k) - h_i^k)$
  end for
  Aggregate received compressed gradient information
  $$g^k = \frac{1}{n}\sum_{i=1}^n \mathcal{C}_i^k(\nabla f_i(x^k) - h_i^k) + h^k$$
  $$h^{k+1} = h^k + \alpha \frac{1}{n}\sum_{i=1}^n \mathcal{C}_i^k(\nabla f_i(w^k) - h_i^k)$$
  Perform update step
  $$y^{k+1} = \text{prox}_{\eta\psi}(x^k - \eta g^k)$$
  $$z^{k+1} = \beta z^k + (1 - \beta)x^k + \frac{\gamma}{\eta}(y^{k+1} - x^k)$$
  $$w^{k+1} = \begin{cases} y^k, & \text{with probability } p \\ w^k, & \text{with probability } 1 - p \end{cases}$$
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 $k = 0, 1, 2, \ldots$ **do**
  broadcast x^k to all workers
  for $i = 1, \ldots, n$ **do**
    Compute local gradient $\nabla f_i(x^k)$
    $h_i^{k+1} = h_i^k + \eta \mathcal{C}_i^k(h_i(x^k) - h_i^k)$
    $\beta_i^k = \max_{j\in[m]} \frac{h_{ij}(x^k) + 2\gamma}{h_{ij}^k + 2\gamma}$
    Send $\nabla f_i(x^k)$, β_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
  $\beta^k = \max_i\{\beta_i^k\}$
  $\mathbf{H}^k = \beta^k \mathbf{A}^k - 2\gamma \cdot \frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m a_{ij}a_{ij}^\top \in \mathbb{R}^{d\times d}$
  $$x^{k+1} = x^k - \left(\mathbf{H}^k + \lambda \mathbf{I}\right)^{-1}\left(\frac{1}{n}\sum_{i=1}^n \nabla f_i(x^k) + \lambda x^k\right)$$
  $$\mathbf{A}^{k+1} = \mathbf{A}^k + \frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m (\eta \mathcal{C}_i^k(h_i(x^k) - h_i^k))_j a_{ij}a_{ij}^\top$$
end for

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

Parameters: learning rate $\eta > 0$
Initialization: $x^0 \in \mathbb{R}^d$; $h_1^0, \ldots, h_n^0 \in \mathbb{R}_+^m$; $\mathbf{H}^0 = \frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m h_{ij}^0 a_{ij}a_{ij}^\top \in \mathbb{R}^{d\times d}$
for $k = 0, 1, 2, \ldots$ **do**
  Broadcast x^k to all workers
  for each node $i = 1, \ldots, n$ **do**
    Compute local gradient $\nabla f_i(x^k)$
    $h_i^{k+1} = [h_i^k + \eta \mathcal{C}_i^k(h_i(x^k) - h_i^k)]_+$
    Send $\nabla f_i(x^k)$ 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
  $$x^{k+1} = x^k - \left(\mathbf{H}^k + \lambda \mathbf{I}\right)^{-1}\left(\frac{1}{n}\sum_{i=1}^n \nabla f_i(x^k) + \lambda x^k\right)$$
  $$\mathbf{H}^{k+1} = \mathbf{H}^k + \frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m (h_{ij}^{k+1} - h_{ij}^k)a_{ij}a_{ij}^\top$$
end for

Algorithm 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 $k = 0, 1, 2, \ldots$ **do**
  Broadcast x^k to all clients — server
  for each worker $i = 1, \ldots, n$ 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
  $x^{k+1} = x^k - \gamma \frac{1}{n}\sum_{i=1}^n \mathcal{C}_i^k(\nabla f_i(x^k))$ — server
end for

Algorithm 1 MARINA

Input: starting point x^0, stepsize γ, probability $p \in (0, 1]$, number of iterations K
Initialize $g^0 = \nabla f(x^0)$
for $k = 0, 1, \ldots, K-1$ **do**
  Sample $c_k \sim \text{Be}(p)$
  Broadcast g^k to all workers
  for $i = 1, \ldots, n$ in parallel **do**
    $x^{k+1} = x^k - \gamma g^k$
    Set $g_i^{k+1} = \nabla f_i(x^{k+1})$ if $c_k = 1$, and $g_i^{k+1} = g^k + \mathcal{Q}\left(\nabla f_i(x^{k+1}) - \nabla f_i(x^k)\right)$ otherwise
  end for
  $g^{k+1} = \frac{1}{n}\sum_{i=1}^n g_i^{k+1}$
end for
Return: $\hat{x}^K$ chosen uniformly at random from $\{x^k\}_{k=0}^{K-1}$

Algorithm 2 VR-MARINA: finite sum case

Input: starting point x^0, stepsize γ, minibatch size b', probability $p \in (0, 1]$, number of iterations K
Initialize $g^0 = \nabla f(x^0)$
for $k = 0, 1, \ldots, K-1$ **do**
  Sample $c_k \sim \text{Be}(p)$
  Broadcast g^k to all workers
  for $i = 1, \ldots, n$ in parallel **do**
    $x^{k+1} = x^k - \gamma g^k$
    Set $g_i^{k+1} = \nabla f_i(x^{k+1})$ if $c_k = 1$, and $g_i^{k+1} = g^k + \mathcal{Q}\left(\frac{1}{b'}\sum_{j\in I'_{i,k}}(\nabla f_{ij}(x^{k+1}) - \nabla f_{ij}(x^k))\right)$ otherwise, where $I'_{i,k}$ is the set of the indices in the minibatch, $|I'_{i,k}| = b'$
  end for
  $g^{k+1} = \frac{1}{n}\sum_{i=1}^n g_i^{k+1}$
end for
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 $x^0, h_1^0, \ldots, h_n^0$ and $h^0 = \frac{1}{n}\sum_{i=1}^n h_i^0$
for $k = 0, 1, \ldots$ **do**
  broadcast x^k to all workers
  for $i = 1, \ldots, n$ **do in parallel** ▷ *worker side*
    sample g_i^k such that $\mathrm{E}\left[g_i^k \mid x^k\right] = \nabla f_i(x^k)$
    $\Delta_i^k = g_i^k - h_i^k$
    $\hat{\Delta}_i^k = Q(\Delta_i^k)$
    $h_i^{k+1} = h_i^k + \alpha \hat{\Delta}_i^k$
    $\hat{g}_i^k = h_i^k + \hat{\Delta}_i^k$
  end
  $g^k = \frac{1}{n}\sum_{i=1}^n \hat{\Delta}_i^k$ ▷ *gather quantized updates*
  $\hat{g}^k = \frac{1}{n}\sum_{i=1}^n \hat{g}_i^k$
  $x^{k+1} = \text{prox}_{\gamma R}(x^k - \gamma \hat{g}^k)$
  $h^{k+1} = \frac{1}{n}\sum_{i=1}^n h_i^{k+1}$
end

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

Input: learning rates $\alpha > 0$ and $\gamma > 0$, initial vectors $x^0, h_1^0, \ldots, h_n^0, h^0 = \frac{1}{n}\sum_{i=1}^n h_i^0$
for $k = 0, 1, \ldots$ **do**
  sample random ▷ *only for Variant 1*
  $$w^k = \begin{cases} 1, & \text{with probability } \frac{1}{m} \\ 0, & \text{with probability } 1 - \frac{1}{m} \end{cases}$$
  broadcast x^k, w^k to all workers
  for $i = 1, \ldots, n$ **do in parallel** ▷ *worker side*
    pick random $j_i^k \sim_{\text{u.a.s.}} [m]$
    $\mu_i^k = \frac{1}{m}\sum_{j=1}^m \nabla f_{ij}(w_{ij}^k)$
    $g_i^k = \nabla f_{ij_i^k}(x^k) - \nabla f_{ij_i^k}(w_{ij_i^k}^k) + \mu_i^k$
    $\Delta_i^k = Q(g_i^k - h_i^k)$
    $h_i^{k+1} = h_i^k + \alpha \Delta_i^k$
    for $j = 1, \ldots, m$ **do**
      ▷ *Variant 1 (L-SVRG): update epoch gradient if $w^k = 1$*
      $$w_{ij}^{k+1} = \begin{cases} x^k, & \text{if } w^k = 1 \\ w_{ij}^k, & \text{if } w^k = 0 \end{cases}$$
      ▷ *Variant 2 (SAGA): update gradient table*
      $$w_{ij}^{k+1} = \begin{cases} x^k, & j = j_i^k \\ w_{ij}^k, & j \neq j_i^k \end{cases}$$
    end
  end
  $h^{k+1} = h^k + \frac{\alpha}{n}\sum_{i=1}^n \Delta_i^k$ ▷ *gather quantized updates*
  $g^k = \frac{1}{n}\sum_{i=1}^n (\Delta_i^k + h_i^k)$

Algorithm 1 Local fixed-point method

Input: Initial estimate $\bar{x}^0 \in \mathbb{R}^d$, stepsize $\lambda > 0$, sequence of synchronization times $0 = t_0 < t_1 < \ldots$
Initialize: $x_i^0 := \bar{x}^0$, for all $i = 1, \ldots, M$
for $k = 0, 1, \ldots$ **do**
  for $i = 1, 2, \ldots, M$ in parallel **do**
    $h_i^{k+1} := (1 - \lambda)x_i^k + \lambda \mathcal{T}_i(x_i^k)$
    if $k + 1 = t_n$, for some $n \in \mathbb{N}$, **then**
      Communicate h_i^{k+1} to master node
    else
      $x_i^{k+1} := h_i^{k+1}$
    end if
  end for
  if $k + 1 = t_n$, for some $n \in \mathbb{N}$, **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
  end if
end for

Algorithm 2 Randomized fixed-point method

Input: Initial estimate $\bar{x}^0 \in \mathbb{R}^d$, stepsize $\lambda > 0$, communication probability $0 < p \leq 1$
Initialize: $x_i^0 = \bar{x}^0$, for all $i = 1, \ldots, M$
for $k = 1, 2, \ldots$ **do**
  for $i = 1, 2, \ldots, M$ in parallel **do**
    $h_i^{k+1} := (1 - \lambda)x_i^k + \lambda \mathcal{T}_i(x_i^k)$
  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
  else, with probability $1 - p$, **do**
    $x_i^{k+1} := h_i^{k+1}$, for all $i = 1, \ldots, M$
end for