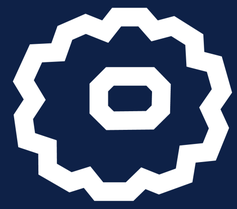


Federating Large Language Models from Scratch

On the opportunities and challenges of the federated pre-training of large language models

Lorenzo Sani, 7th October 2024 @ Alan Turing Institute - “*Robots in Disguise*: Fundamental AI Reading Group”

Federated Learning

Privacy and more...

"***Federated learning*** *is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective."*

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:

initialize w_0

for each round $t = 1, 2, \dots$ **do**

$m \leftarrow \max(C \cdot K, 1)$

$S_t \leftarrow$ (random set of m clients)

for each client $k \in S_t$ **in parallel do**

$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$

$m_t \leftarrow \sum_{k \in S_t} n_k$

$w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ *// Erratum*[4]

ClientUpdate(k, w): *// Run on client k*

$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)

for each local epoch i from 1 to E **do**

for batch $b \in \mathcal{B}$ **do**

$w \leftarrow w - \eta \nabla \ell(w; b)$

return w to server

Federated Learning

Privacy and more...

*"**Federated learning** is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective."*

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:

initialize w_0

for each round $t = 1, 2, \dots$ **do**

  $m \leftarrow \max(C \cdot K, 1)$

  $S_t \leftarrow$ (random set of m clients)

  for each client $k \in S_t$ **in parallel do**

    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$

  $m_t \leftarrow \sum_{k \in S_t} n_k$

  $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ *// Erratum*[4]

ClientUpdate(k, w): *// Run on client k*

  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)

  for each local epoch i from 1 to E **do**

    for batch $b \in \mathcal{B}$ **do**

      $w \leftarrow w - \eta \nabla \ell(w; b)$

  return w to server

[1] Communication-Efficient Learning of Deep Networks from Decentralized Data. (McMahan, H.B., et al., '17)
[2] Advances and Open Problems in Federated Learning. (Kairouz, P., et al., '19)

Is FL itself enough for privacy guarantees?

- Protecting the privacy
  - clients and/or server
- Leaking privacy from clients' updates
- Pseudo-gradient vs model updates
- Hyperparameters and trade-offs

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $m_t \leftarrow \sum_{k \in S_t} n_k$
  $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ *// Erratum*[4]

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

Is FL itself enough for privacy guarantees?

- Protecting the privacy
  - clients and/or server
- Leaking privacy from clients' updates
- Pseudo-gradient vs model updates
- Hyperparameters and trade-offs

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  $m_t \leftarrow \sum_{k \in S_t} n_k$
  $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$ *// Erratum*[4]

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

[1] Communication-Efficient Learning of Deep Networks from Decentralized Data. (McMahan, H.B., et al., '17)
[2] Advances and Open Problems in Federated Learning. (Kairouz, P., et al., '19)

From Distributed to Federated

The LLM pre-training example

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$

Output: Trained model parameters θ_T

From Distributed to Federated

The LLM pre-training example

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

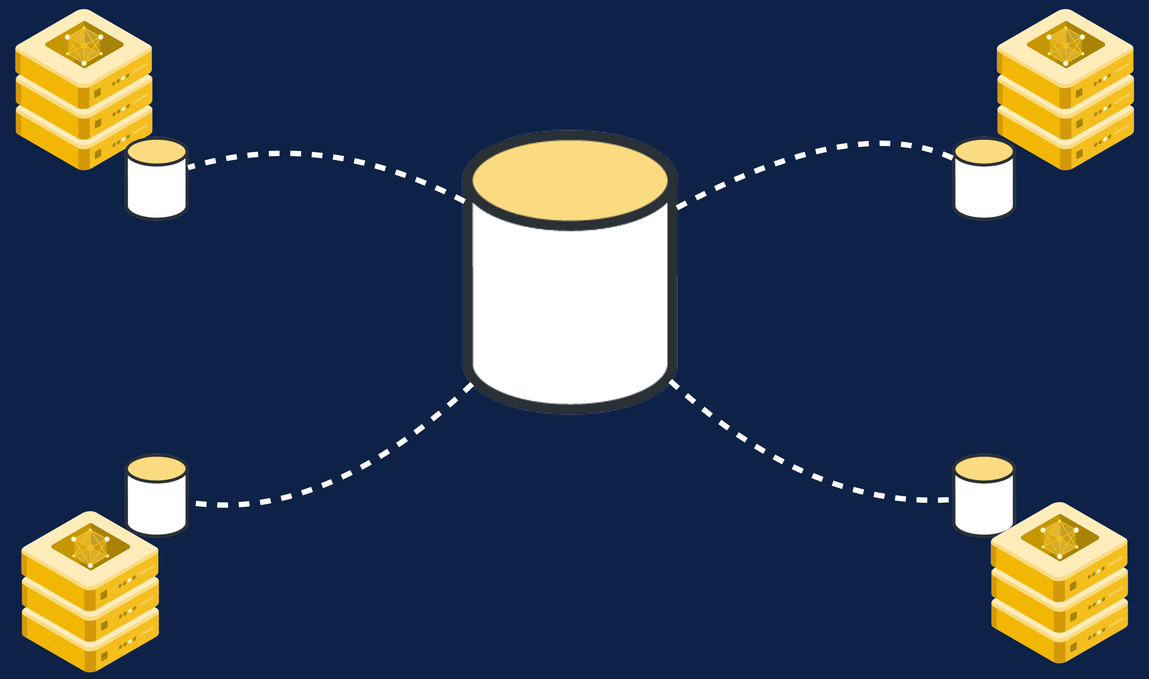
Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$

Output: Trained model parameters θ_T



From Distributed to Federated

The LLM pre-training example

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$

Output: Trained model parameters θ_T

From Distributed to Federated

The LLM pre-training example

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

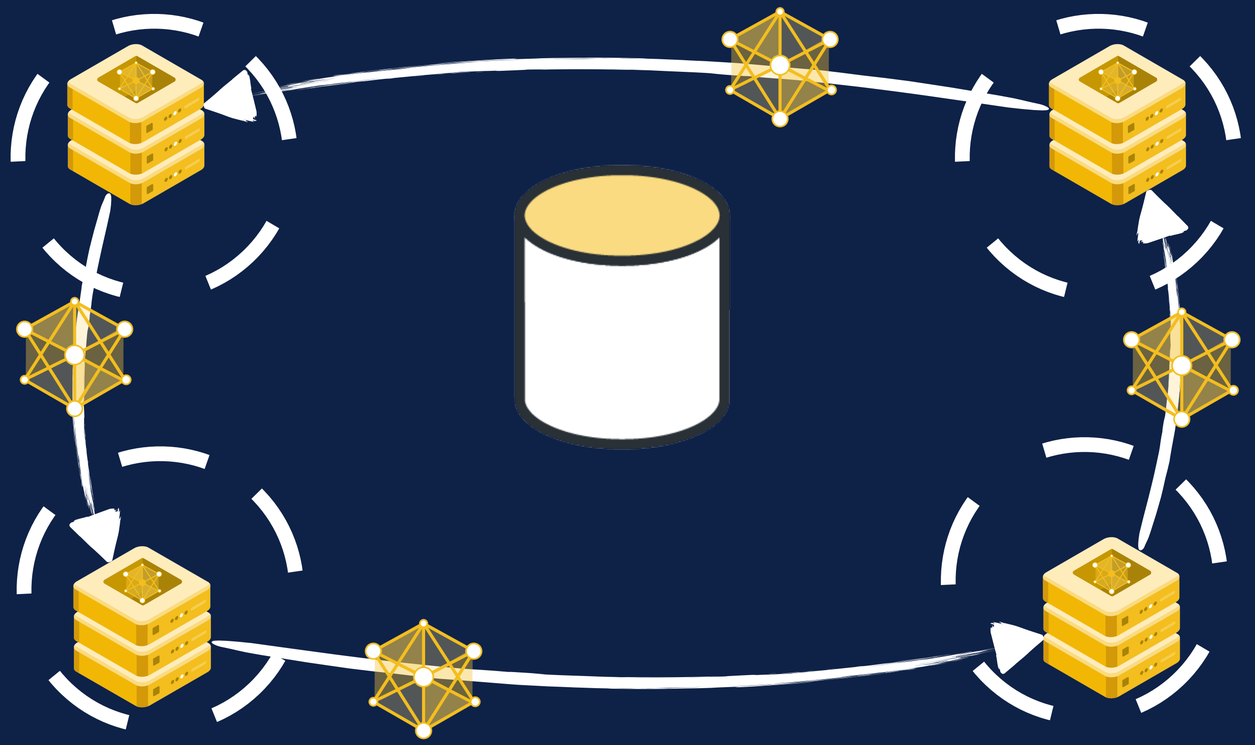
Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$

Output: Trained model parameters θ_T



From Distributed to Federated

The LLM pre-training example

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

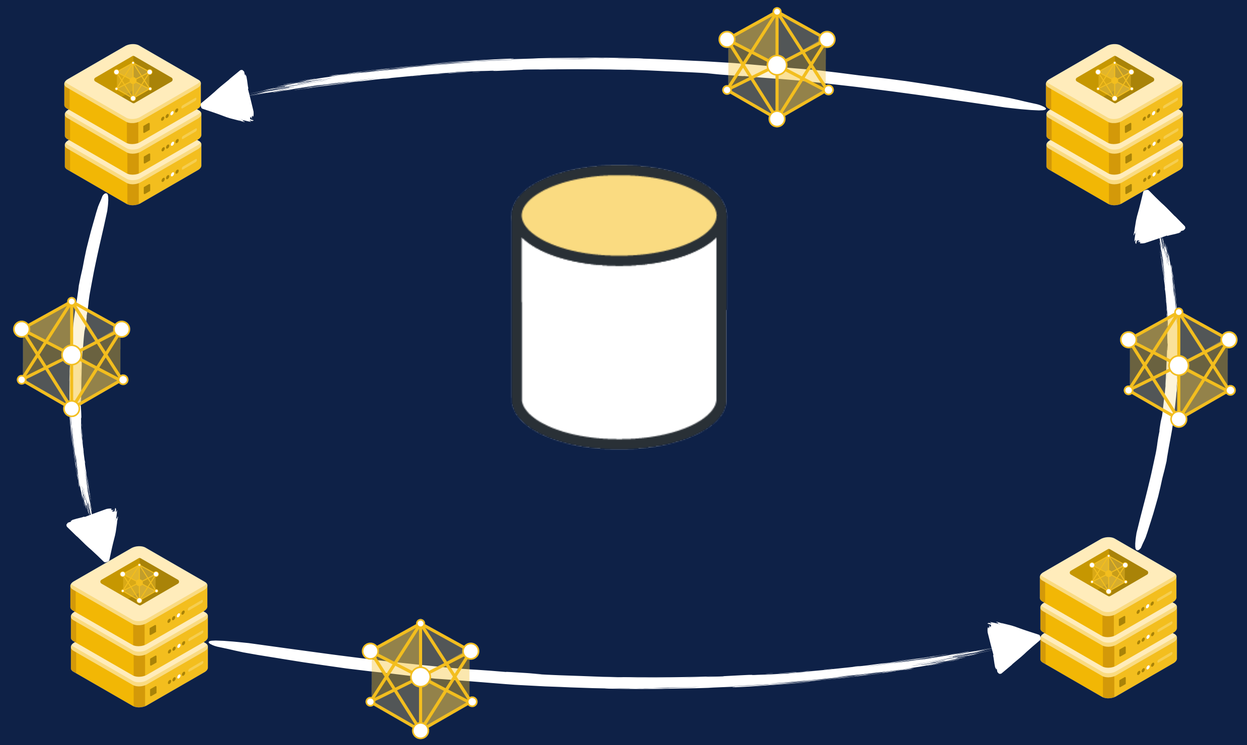
Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$

Output: Trained model parameters θ_T



From Distributed to Federated

The LLM pre-training example

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

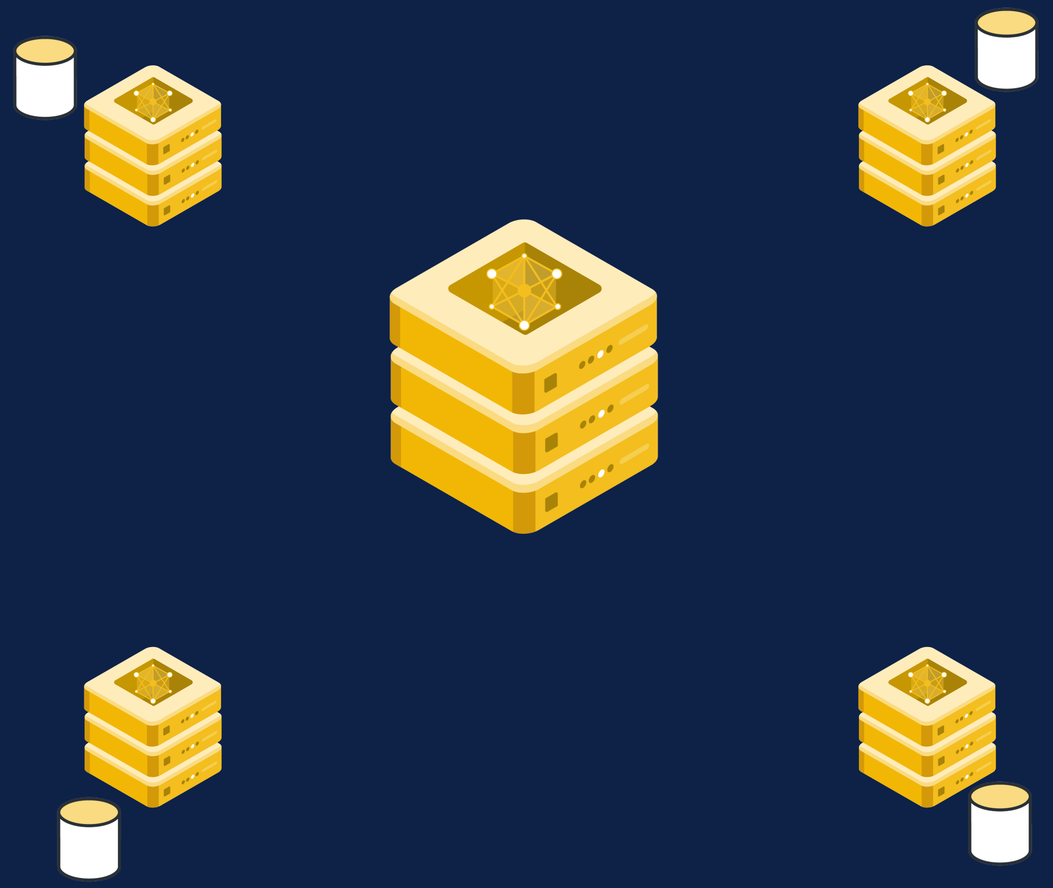
Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
  Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

  Step 1: Broadcast model parameters
  Server sends θ_t to all N clients

  Step 2: Parallel Local Training
  for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
    $\omega_{i,0} \leftarrow \theta_{t-1}$
    for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
      Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
      Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
      $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
    $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

  Step 3: Global Model Update (on the server)
  $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$
Output: Trained model parameters θ_T



From Distributed to Federated

The LLM pre-training example

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

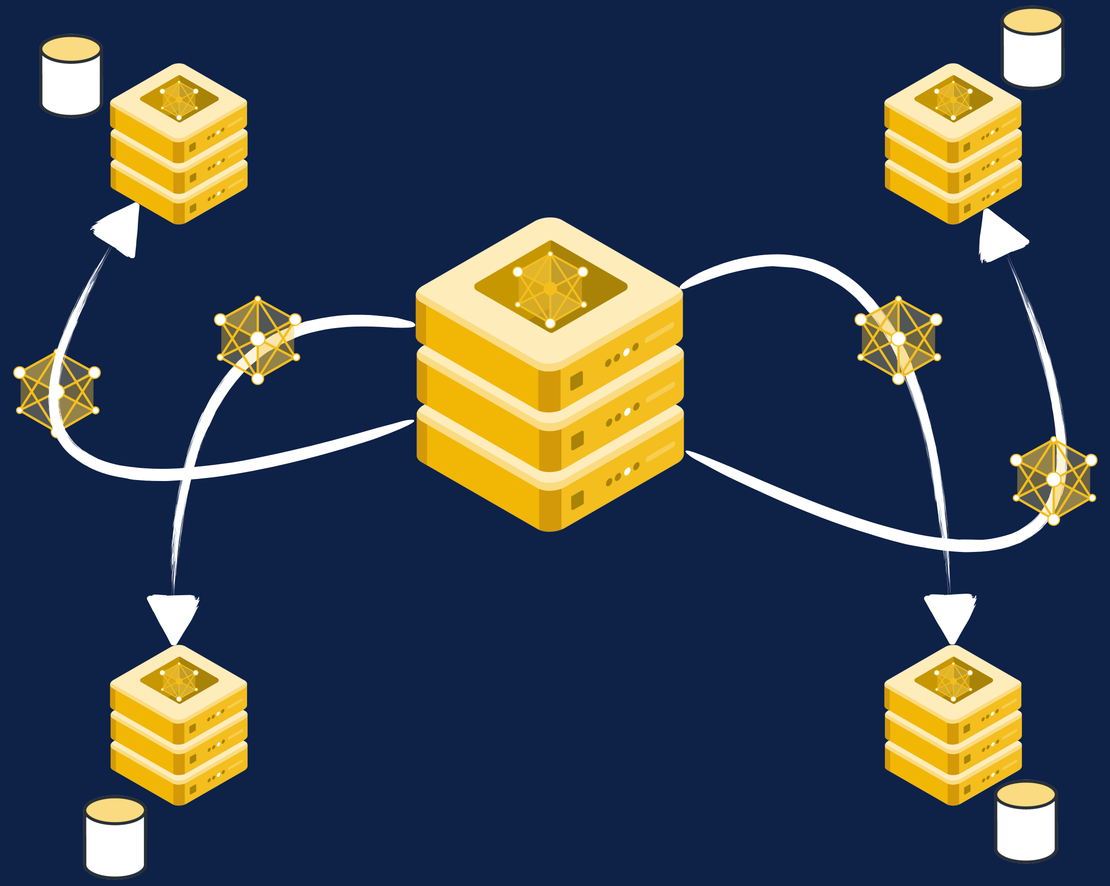
Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
  Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

  Step 1: Broadcast model parameters
  Server sends θ_t to all N clients

  Step 2: Parallel Local Training
  for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
    $\omega_{i,0} \leftarrow \theta_{t-1}$
    for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
      Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
      Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
      $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
    $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

  Step 3: Global Model Update (on the server)
  $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$
Output: Trained model parameters θ_T



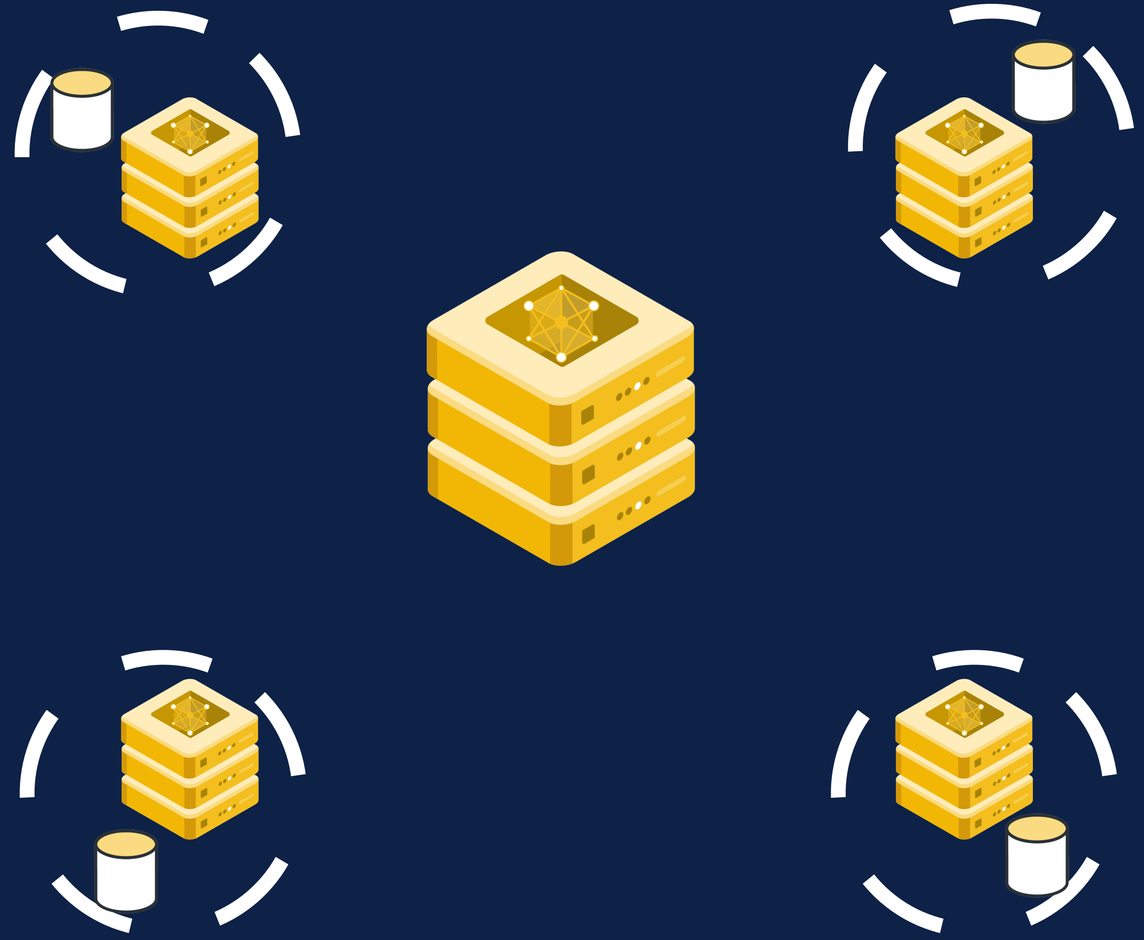
From Distributed to Federated

The LLM pre-training example

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
  Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**
  Step 1: Broadcast model parameters
  Server sends θ_t to all N clients
  Step 2: Parallel Local Training
  for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
    $\omega_{i,0} \leftarrow \theta_{t-1}$
    for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
      Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
      Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
      $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
    $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$
  Step 3: Global Model Update (on the server)
  $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$
Output: Trained model parameters θ_T



From Distributed to Federated

The LLM pre-training example

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

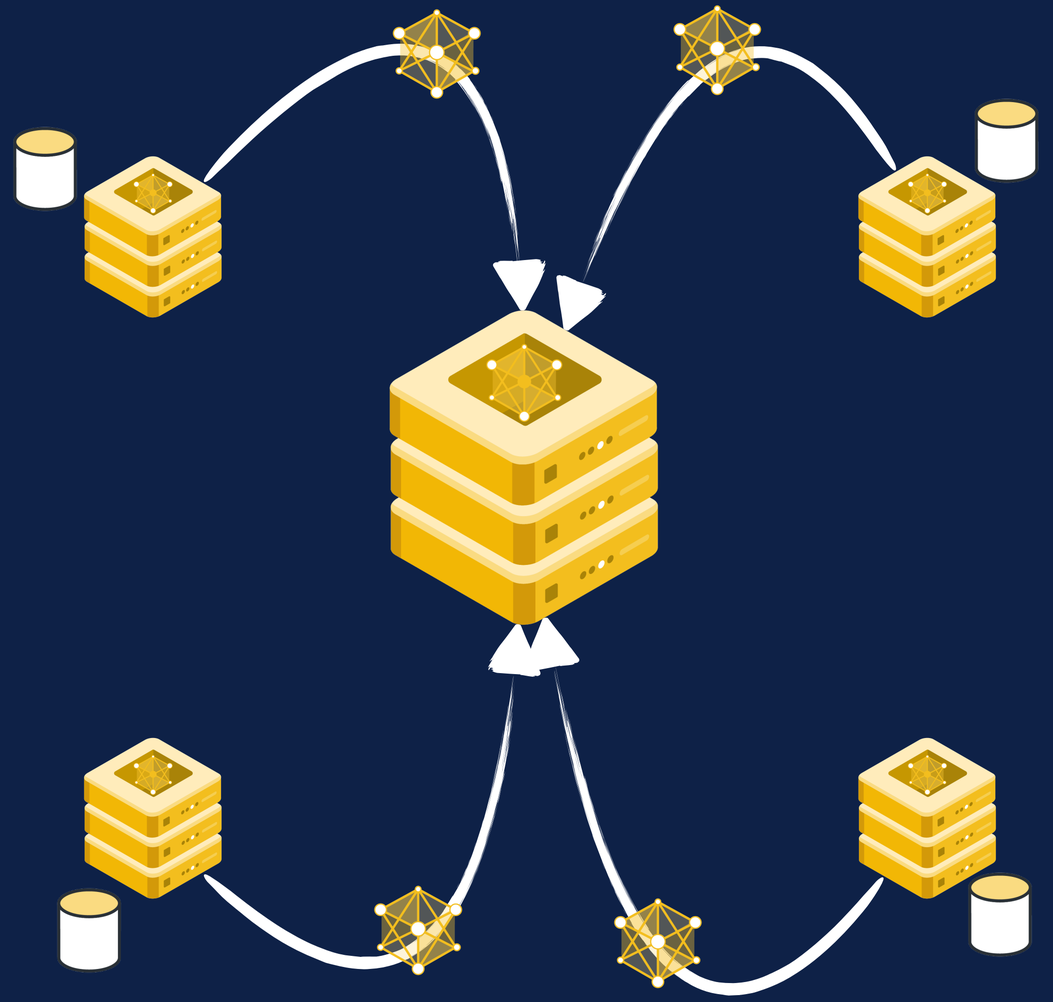
Initialize:
  Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

  Step 1: Broadcast model parameters
  Server sends θ_t to all N clients

  Step 2: Parallel Local Training
  for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
    $\omega_{i,0} \leftarrow \theta_{t-1}$
    for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
      Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
      Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
      $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
    $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

  Step 3: Global Model Update (on the server)
  $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$

Output: Trained model parameters θ_T



From Distributed to Federated

The LLM pre-training example

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

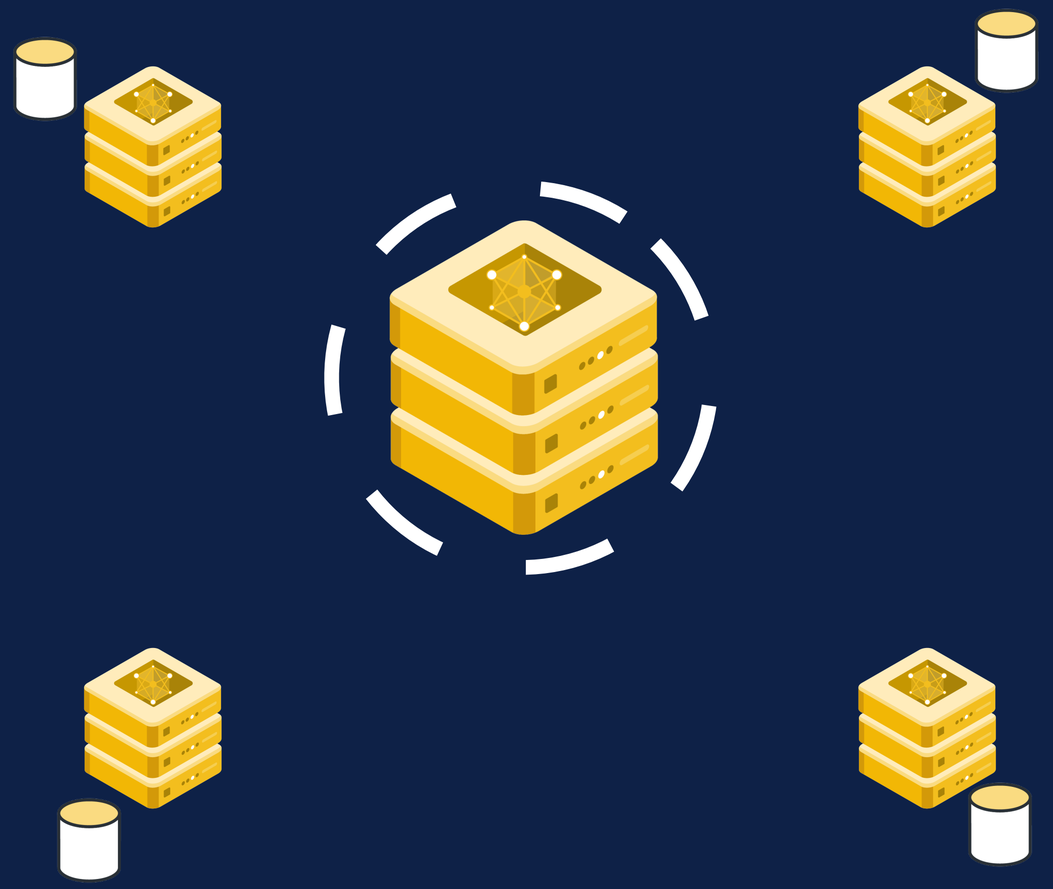
Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
  Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

  Step 1: Broadcast model parameters
  Server sends θ_t to all N clients

  Step 2: Parallel Local Training
  for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
    $\omega_{i,0} \leftarrow \theta_{t-1}$
    for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
      Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
      Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
      $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
    $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

  Step 3: Global Model Update (on the server)
  $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$
Output: Trained model parameters θ_T



From Distributed to Federated

DDP

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$
Output: Trained model parameters θ_T

Cross-silo FL

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
    Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

    Step 1: Broadcast model parameters
    Server sends θ_t to all N clients

    Step 2: Parallel Local Training
    for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\omega_{i,0} \leftarrow \theta_{t-1}$
        for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
            Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
            Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
            $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
        $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

    Step 3: Global Model Update (on the server)
    $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$
Output: Trained model parameters θ_T

From Distributed to Federated

DDP

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$

Output: Trained model parameters θ_T

Cross-silo FL

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
    Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

    Step 1: Broadcast model parameters
    Server sends θ_t to all N clients

    Step 2: Parallel Local Training
    for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\omega_{i,0} \leftarrow \theta_{t-1}$
        for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
            Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
            Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
            $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
        $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

    Step 3: Global Model Update (on the server)
    $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$

Output: Trained model parameters θ_T

From Distributed to Federated

DDP

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$

Output: Trained model parameters θ_T

Cross-silo FL

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
    Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

    Step 1: Broadcast model parameters
    Server sends θ_t to all N clients

    Step 2: Parallel Local Training
    for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\omega_{i,0} \leftarrow \theta_{t-1}$
        for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
            Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
            Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
            $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
        $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

    Step 3: Global Model Update (on the server)
    $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$

Output: Trained model parameters θ_T

From Distributed to Federated

DDP

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$

Output: Trained model parameters θ_T

Cross-silo FL

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
    Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

    Step 1: Broadcast model parameters
    Server sends θ_t to all N clients

    Step 2: Parallel Local Training
    for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\omega_{i,0} \leftarrow \theta_{t-1}$
        for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
            Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
            Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
            $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
        $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

    Step 3: Global Model Update (on the server)
    $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$

Output: Trained model parameters θ_T

From Distributed to Federated

DDP

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$
Output: Trained model parameters θ_T

Cross-silo FL

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
    Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

    Step 1: Broadcast model parameters
    Server sends θ_t to all N clients

    Step 2: Parallel Local Training
    for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\omega_{i,0} \leftarrow \theta_{t-1}$
        for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
            Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
            Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
            $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
        $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

    Step 3: Global Model Update (on the server)
    $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$
Output: Trained model parameters θ_T

From Distributed to Federated

DDP

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$

Output: Trained model parameters θ_T

Cross-silo FL

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
    Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

    Step 1: Broadcast model parameters
    Server sends θ_t to all N clients

    Step 2: Parallel Local Training
    for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\omega_{i,0} \leftarrow \theta_{t-1}$
        for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
            Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
            Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
            $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
        $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

    Step 3: Global Model Update (on the server)
    $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$

Output: Trained model parameters θ_T

From Distributed to Federated

DDP

Algorithm 1 Distributed Data Parallel (DDP) Training Algorithm

Require: N: Number of devices (workers), f_θ: Model with parameters θ
Require: T: Number of epochs
Require: $\mathcal{D}$: Dataset partitioned across devices $\mathcal{D}_i$ where $i \in \{1, 2, \ldots, N\}$
Require: `RingAllReduce`: All-reduce operation to aggregate across devices on a ring
Require: `Opt`: Optimizer for updating θ with gradients

Initialize:
    Randomly initialize model parameters θ_0 on each device
for $t = 1$ to T **do**

    Step 1: Parallel Local Training
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        Compute local mini-batch loss $\mathcal{L}_i(\theta_{t-1}, \mathcal{D}_i)$
        Compute local gradients $\nabla_{\theta_{t-1}} \mathcal{L}_i(\theta_{t-1})$

    Step 2: Distributed `RingAllReduce` Gradient Aggregation
    $\nabla_{\theta_{t-1}} \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\theta_{t-1}} \mathcal{L}_i$
    Each device now possesses the global gradient $\nabla_{\theta_{t-1}} \mathcal{L}$

    Step 3: Parallel Model Update
    for each device $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\theta_t = \texttt{Opt}(\theta_{t-1}, \nabla_{\theta_{t-1}} \mathcal{L})$

Output: Trained model parameters θ_T

Cross-silo FL

Algorithm 2 Cross-silo Federated Learning (FL) Algorithm

Require: N: Number of clients, f_θ: Model with parameters θ
Require: T: Number of federated rounds, K: Number of local steps
Require: $\{\mathcal{D}_i\}$: Federated dataset, i.e., a set of private $\mathcal{D}_i$, $i \in \{1, \ldots, N\}$
Require: `ClientOpt`: local client optimizer
Require: `ServerOpt`: server optimizer

Initialize:
    Randomly initialize global model parameters θ_0 on the server
for $t = 1$ to T **do**

    Step 1: Broadcast model parameters
    Server sends θ_t to all N clients

    Step 2: Parallel Local Training
    for each client $i \in \{1, 2, \ldots, N\}$ **in parallel do**
        $\omega_{i,0} \leftarrow \theta_{t-1}$
        for each local iteration $k \in \{1, 2, \ldots, K\}$ **do**
            Compute local mini-batch loss $\mathcal{L}_i(\omega_{i,k-1}, \mathcal{D}_i)$
            Compute local gradients $\nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1})$
            $\omega_{i,k} \leftarrow \texttt{ClientOpt}(\omega_{i,k-1}, \nabla_{\omega_{i,k-1}} \mathcal{L}_i(\omega_{i,k-1}))$
        $\Delta\theta_{t-1,i} \leftarrow \omega_{i,K} - \theta_{t-1}$

    Step 3: Global Model Update (on the server)
    $\theta_t = \texttt{ServerOpt}(\theta_{t-1}, \{\Delta\theta_{t-1,i}\})$

Output: Trained model parameters θ_T

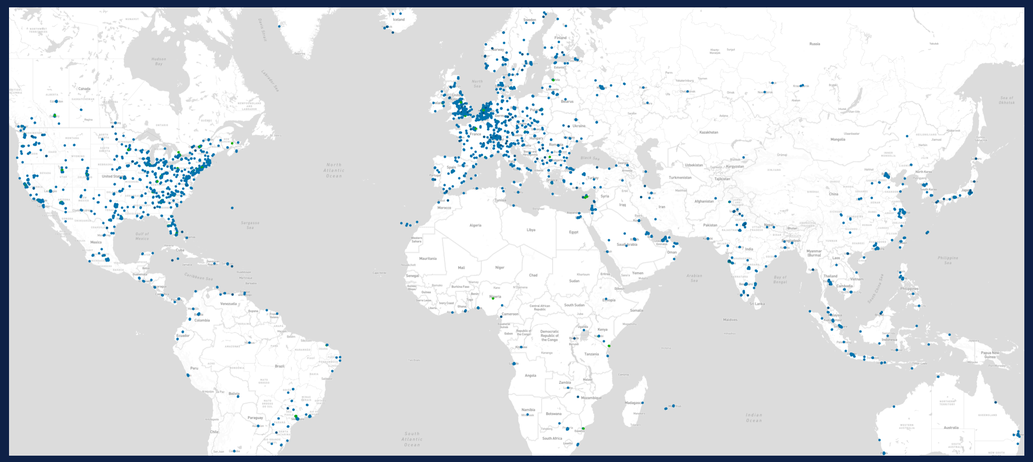
Can FL be useful to LLMs?

Can LLMs be useful to FL?

Can FL be useful to LLMs?

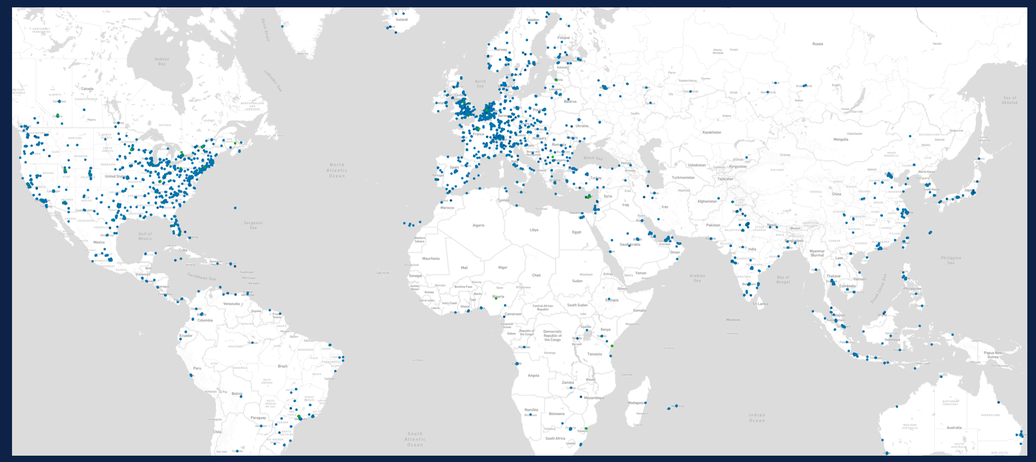
Can LLMs be useful to FL?

Can FL be useful to LLMs?



Can LLMs be useful to FL?

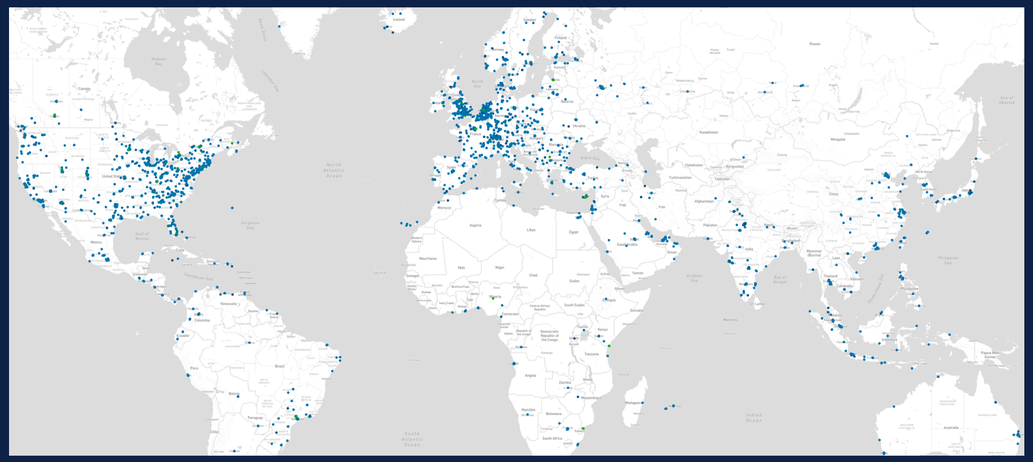
Can FL be useful to LLMs?



YES!

Can LLMs be useful to FL?

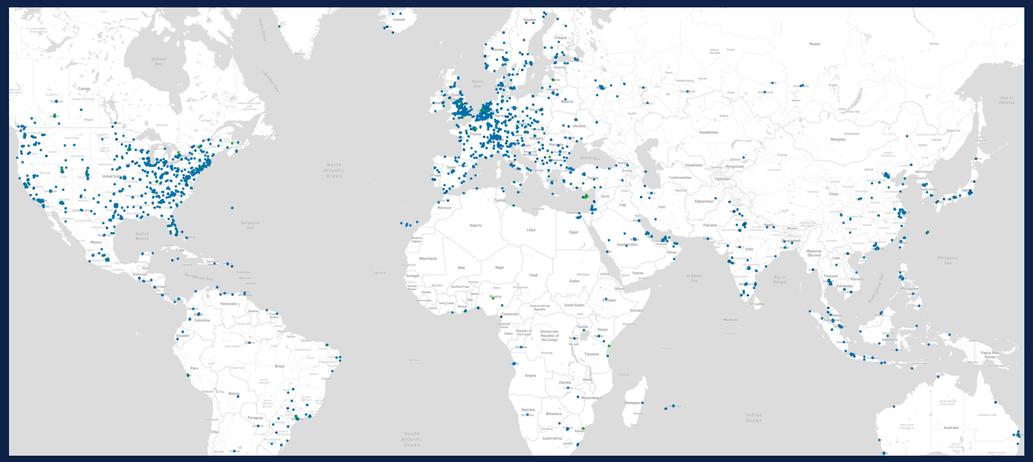
Can FL be useful to LLMs?



YES!

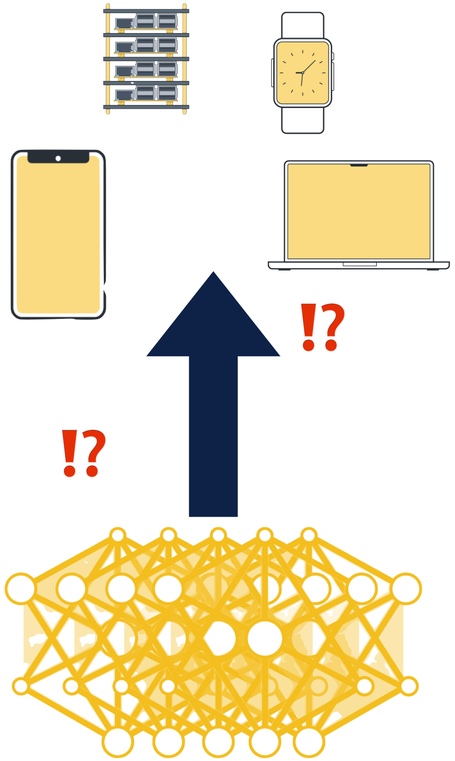
Can LLMs be useful to FL?

Can FL be useful to LLMs?

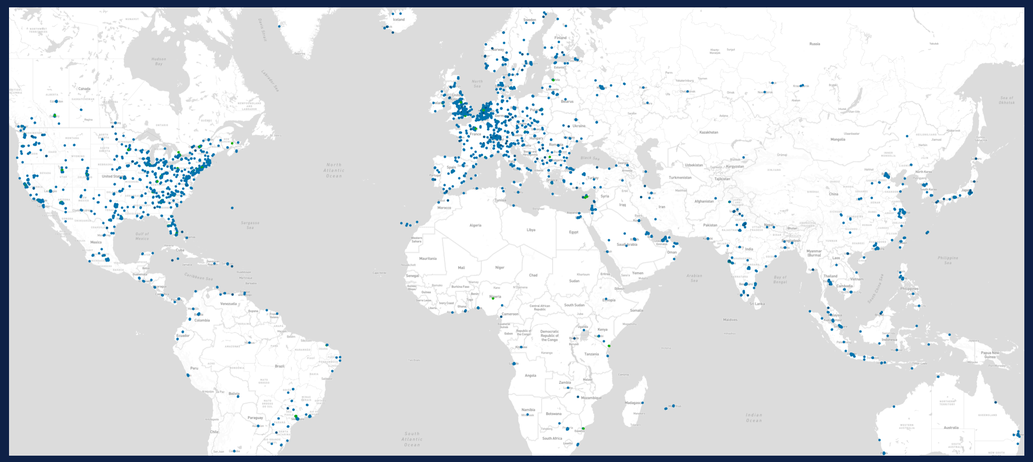


YES!

Can LLMs be useful to FL?

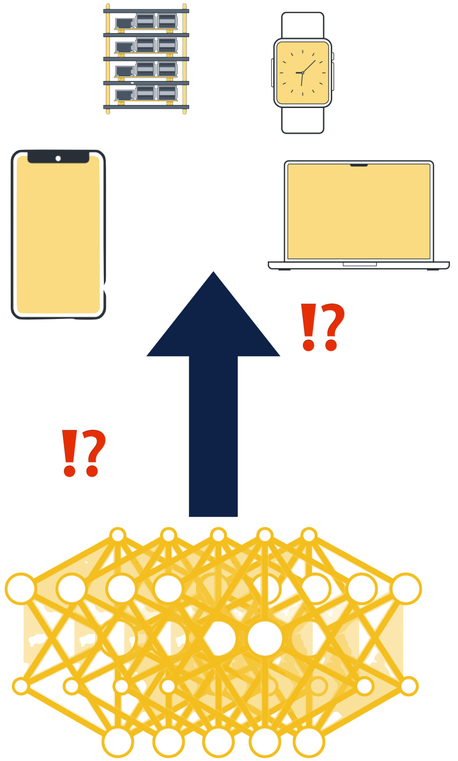


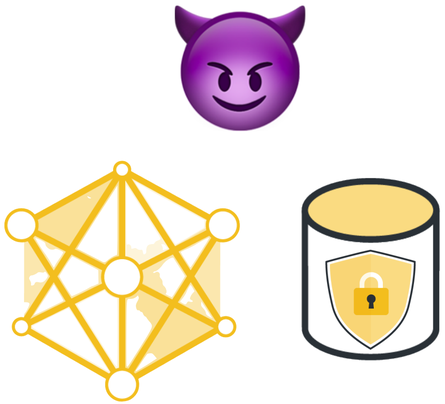
Can FL be useful to LLMs?



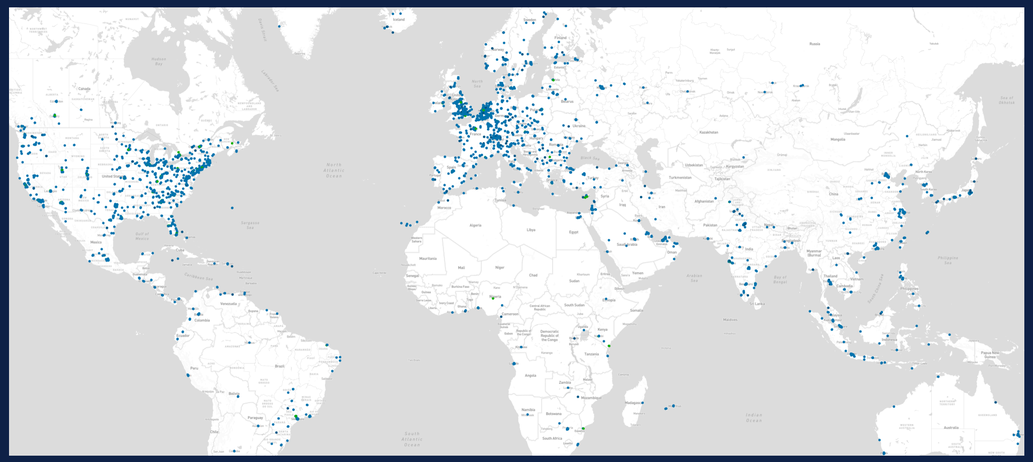
YES!

Can LLMs be useful to FL?



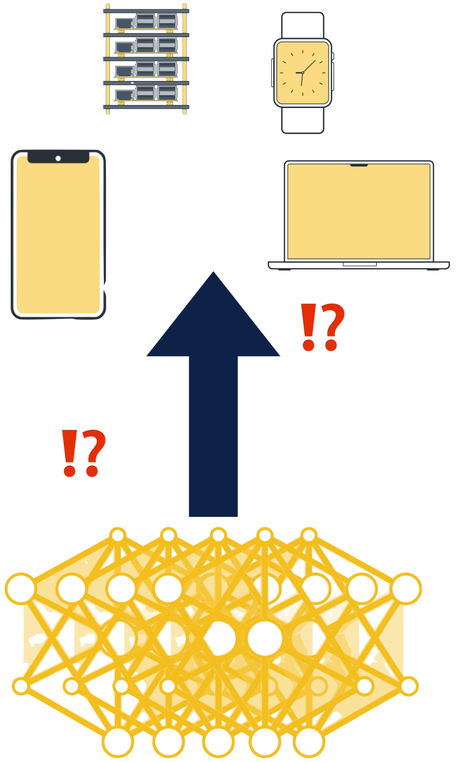


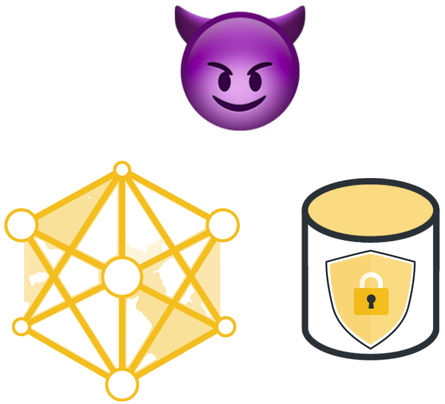
Can FL be useful to LLMs?



YES!

Can LLMs be useful to FL?





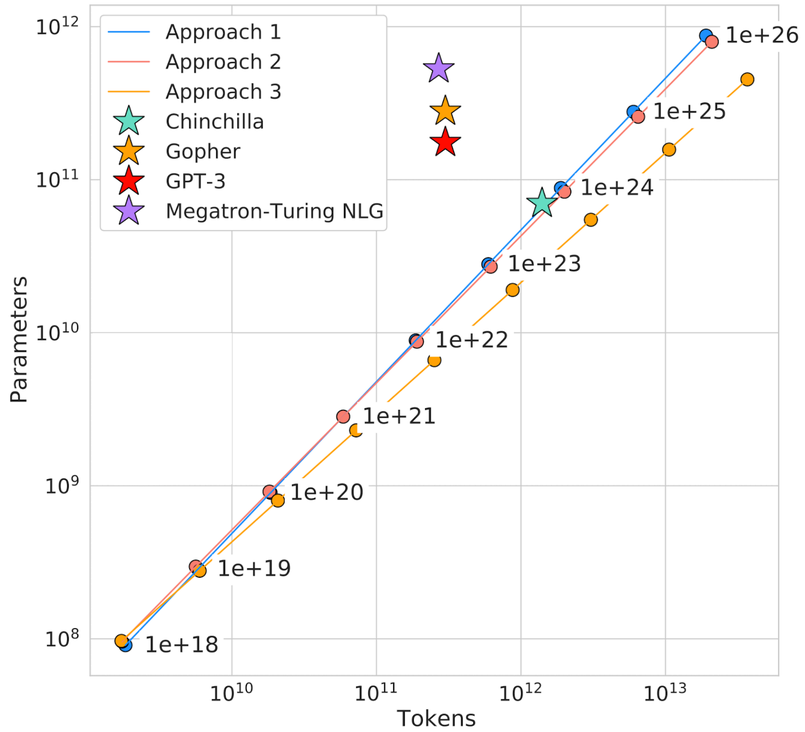
challenges and opportunities!

More motivations

Limitations of LLMs scaling laws [3,4] in context

More motivations

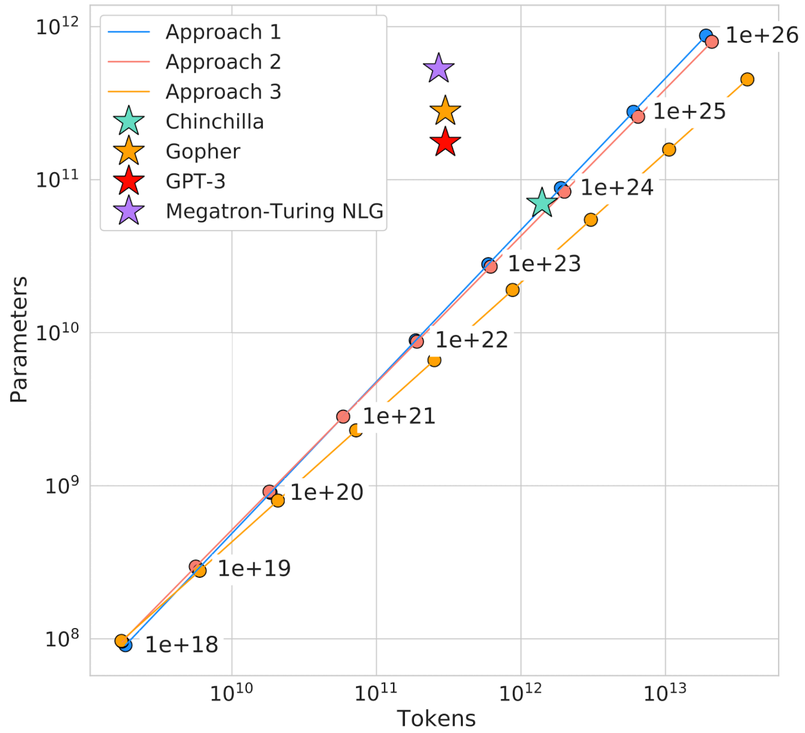
Limitations of LLMs scaling laws [3,4] in context



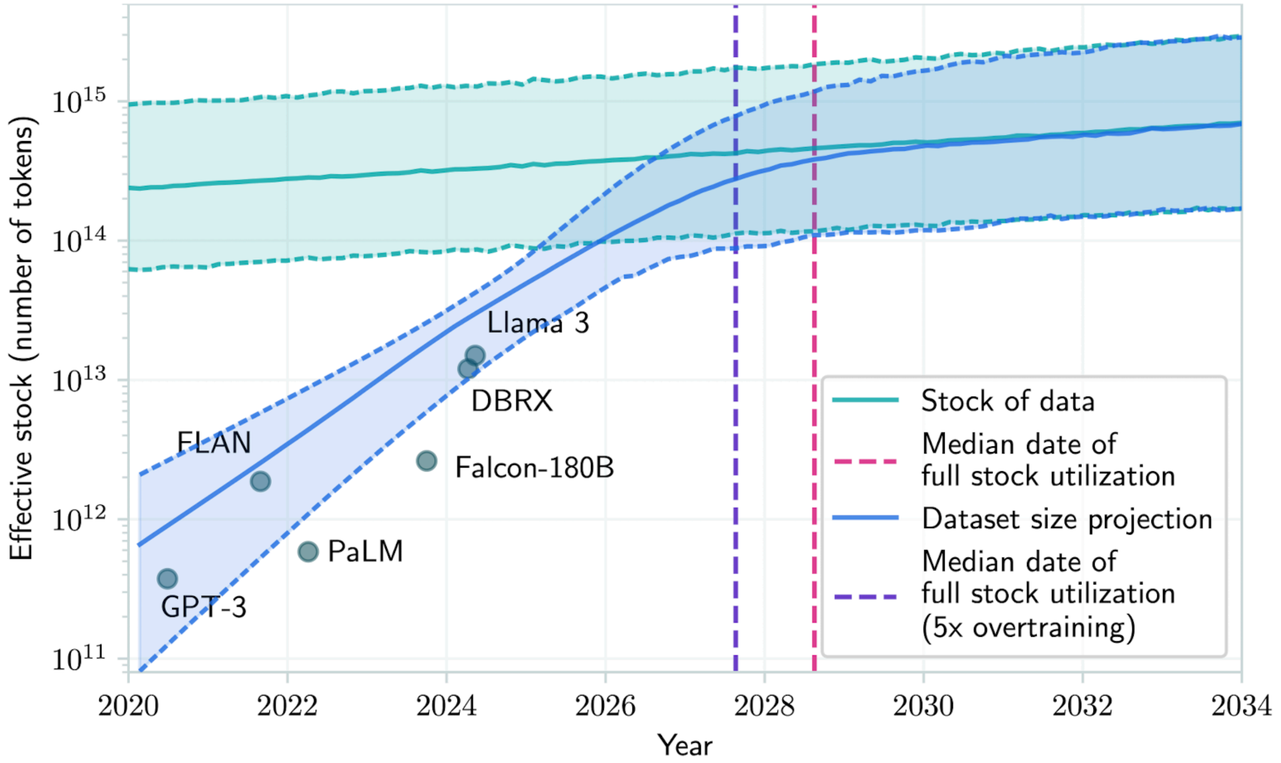
[3] Training Compute-Optimal Large Language Models (Hoffmann, J., et al., '22)

More motivations

Limitations of LLMs scaling laws [3,4] in context



[3] Training Compute-Optimal Large Language Models (Hoffmann, J., et al., '22)



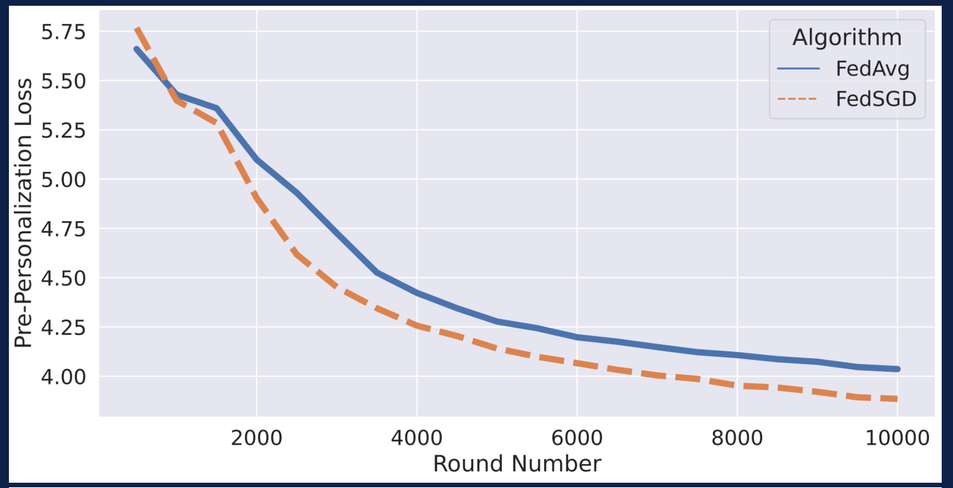
[5] Will we run out of data? An analysis of the limits of scaling datasets in Machine Learning (Villalobos et al., '22)

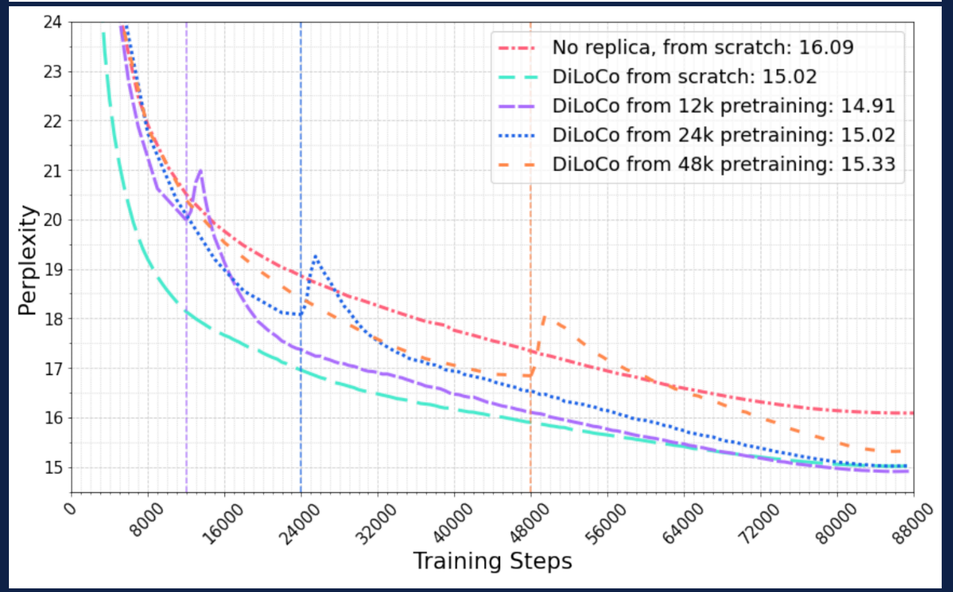
[3] Training Compute-Optimal Large Language Models. (Hoffmann, J., et al., '22)
[4] Beyond Chinchilla-Optimal: Accounting for Inference in Language Model Scaling Laws (Sardana, L., et al., '24)
[5] Will we run out of data? An analysis of the limits of scaling datasets in Machine Learning (Villalobos et al., '22)

LLMs and FL - Related Works

LLMs and FL - Related Works

- **Pre-training LLMs in federated settings**
  ⇒ Dataset Grouper [6]
  ⇒ DiLoCo-DiPaCo (semi-federated) [7,8]





[6] Towards Federated Foundation Models: Scalable Dataset Pipelines for Group-Structured Learning. (Charles, Z., et al., '23)
[7] DiLoCo: Distributed Low-Communication Training of Language Models. (Douillard, A., et al., '23)
[8] DiPaCo: Distributed Path Composition (Douillard, A., et al., '24)

LLMs and FL - Related Works

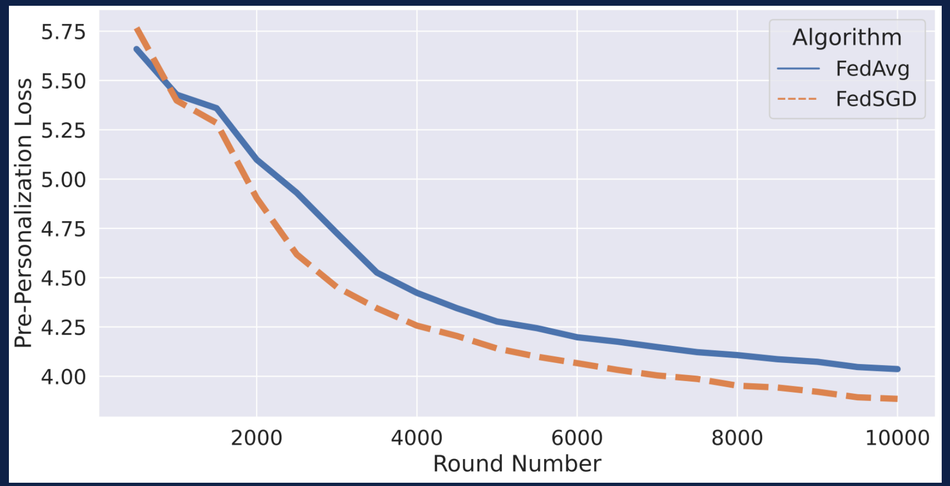
- **Pre-training LLMs in federated settings**
  ⇒ Dataset Grouper [6]

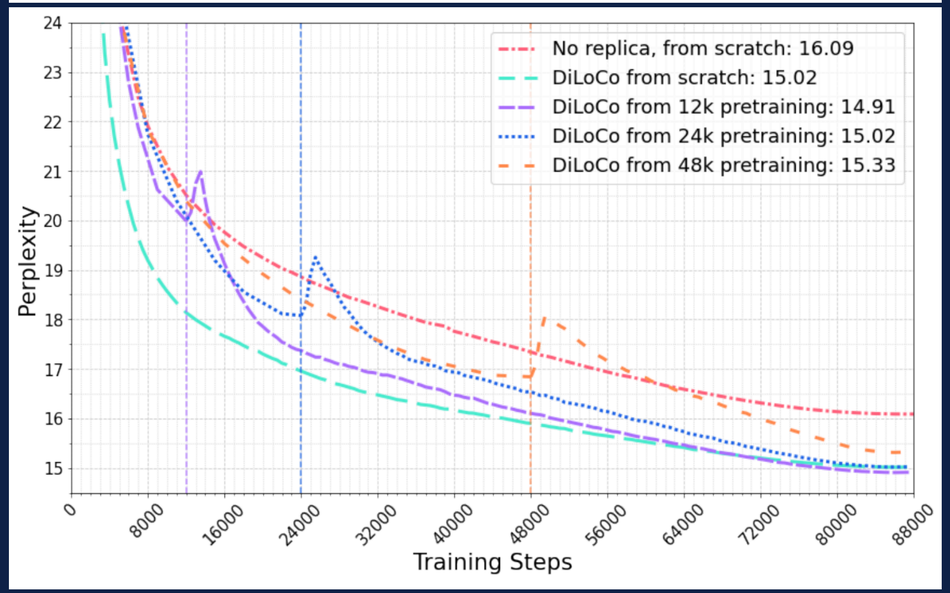
  ⇒ DiLoCo-DiPaCo (semi-federated) [7,8]

- Federated Fine-tuning of LMs
  ⇒ FL > centralized | some heterogeneous settings

- Federated PEFT and Federated LoRA
  ⇒ communication and computation efficiency

- Federated Prompt Tuning
  ⇒ improve generalization





[6] Towards Federated Foundation Models: Scalable Dataset Pipelines for Group-Structured Learning. (Charles, Z., et al., '23)
[7] DiLoCo: Distributed Low-Communication Training of Language Models. (Douillard, A., et al., '23)
[8] DiPaCo: Distributed Path Composition (Douillard, A., et al., '24)

!?

FL scale like centralized

!?

!?

model the token-to-parameter optimality in FL

Burning question?

FL and architecture-wise optimizations (MoEs, split learning, etc.)

!?

!?

multilinguality
meta learning
plasticity

!?

FL scale like centralized

model the token-to-parameter optimality in FL

Photon

FL and architecture-wise optimizations (MoEs, split learning, etc.)

multilinguality
meta learning
plasticity

FL scale like centralized

model the token-to-parameter optimality in FL

Photon

FL and architecture-wise optimizations (MoEs, split learning, etc.)

multilinguality
meta learning
plasticity

Photon's Fantastic Four [7]

The design principles for obtaining an inclusive FL system for LLM pre-training

[9] The Future of Large Language Model Pre-training is Federated. (Sani, L., et al., '24)

Photon's Fantastic Four [7]

The design principles for obtaining an inclusive FL system for LLM pre-training

Broad Access to Data and Compute

- data >> compute: data-rich but compute-poor is welcome
- contribute with data, compute, or both

[9] The Future of Large Language Model Pre-training is Federated. (Sani, L., et al., '24)

Photon's Fantastic Four [7]

The design principles for obtaining an inclusive FL system for LLM pre-training

Broad Access to Data and Compute

- data >> compute: data-rich but compute-poor is welcome
- contribute with data, compute, or both

Limited Communication Requirements

- infrequent communication
- heterogeneous networks

[9] The Future of Large Language Model Pre-training is Federated. (Sani, L., et al., '24)

Photon's Fantastic Four [7]

The design principles for obtaining an inclusive FL system for LLM pre-training

Broad Access to Data and Compute

- data >> compute: data-rich but compute-poor is welcome
- contribute with data, compute, or both

Limited Communication Requirements

- infrequent communication
- heterogeneous networks

Broad Hardware Inclusivity

- support a spectrum of local topologies
- optimize for local efficiency

[9] The Future of Large Language Model Pre-training is Federated. (Sani, L., et al., '24)

Photon's Fantastic Four [7]

The design principles for obtaining an inclusive FL system for LLM pre-training

Broad Access to Data and Compute

- data >> compute: data-rich but compute-poor is welcome
- contribute with data, compute, or both

Broad Hardware Inclusivity

- support a spectrum of local topologies
- optimize for local efficiency

Limited Communication Requirements

- infrequent communication
- heterogeneous networks

Scalable Local Training Pipeline

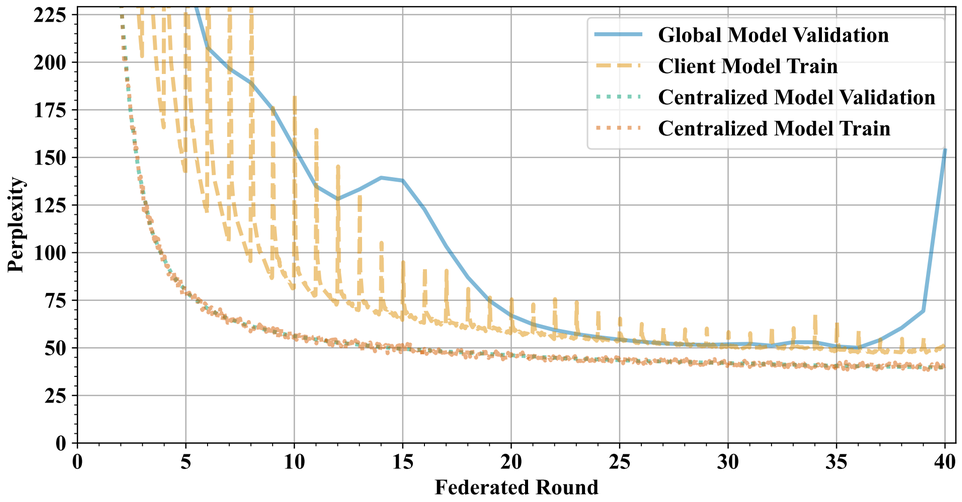
- many sizes, many scales, one recipe

[9] The Future of Large Language Model Pre-training is Federated. (Sani, L., et al., '24)

FL robust at different model scales

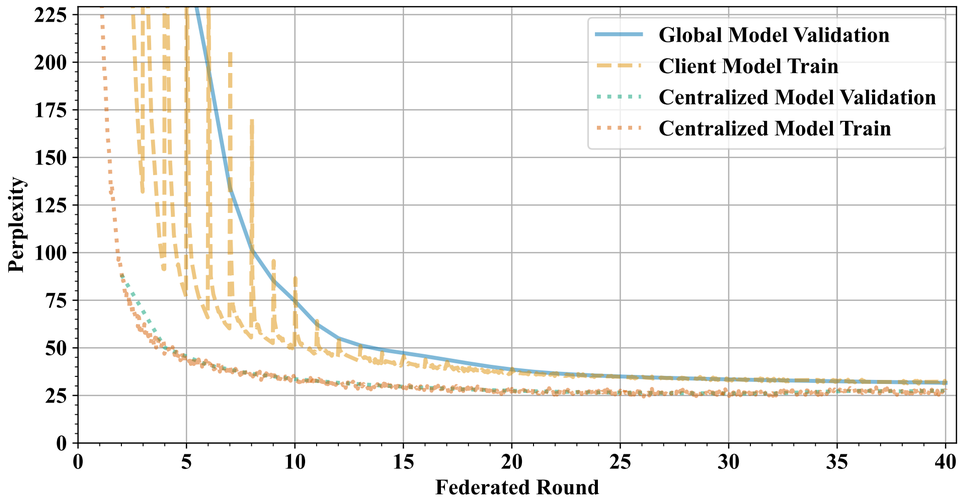
!?

!?



75M

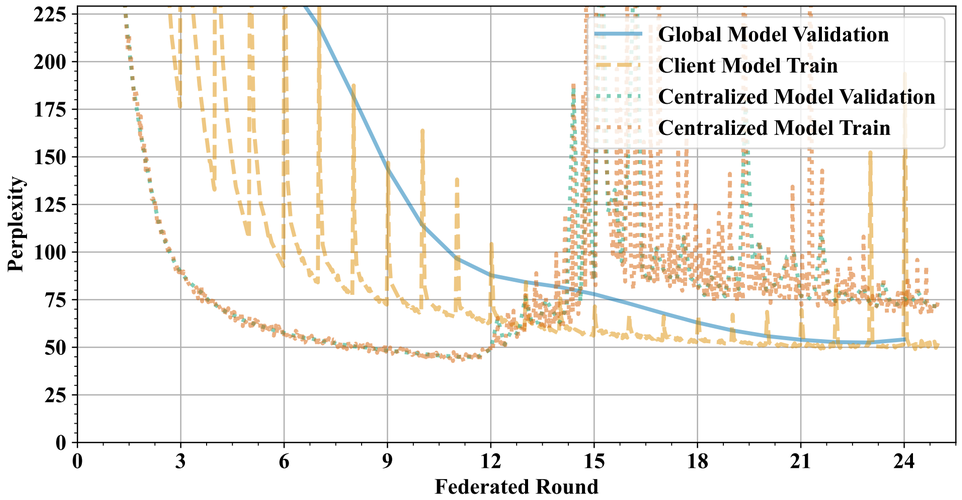
125M



C4 (IID) - P = 8 clients - 100% sampling rate

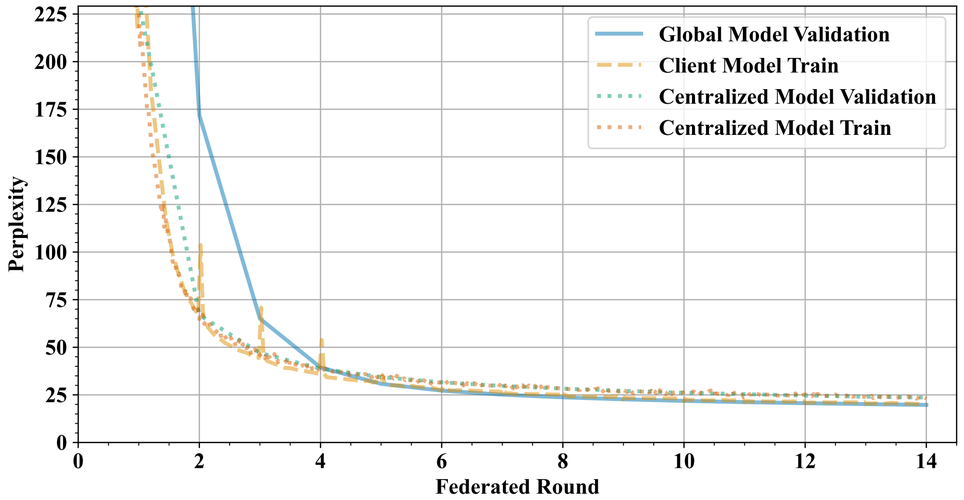
ServerOpt ⇒ SGD + Nestorov

ClientOpt ⇒ AdamW + cos LR with warmup

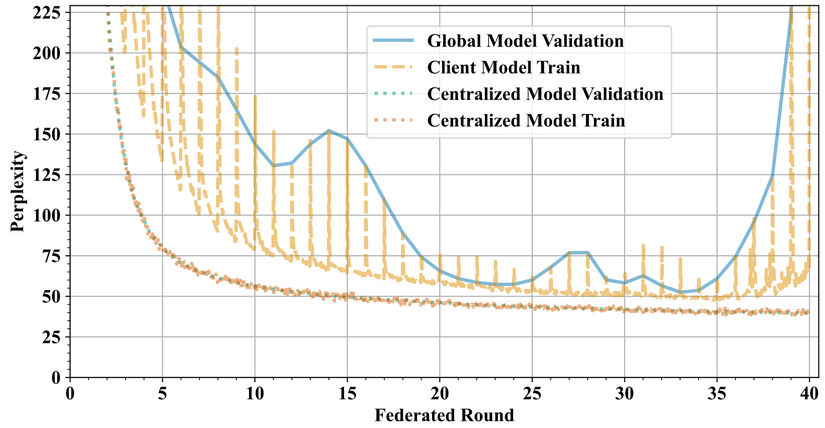


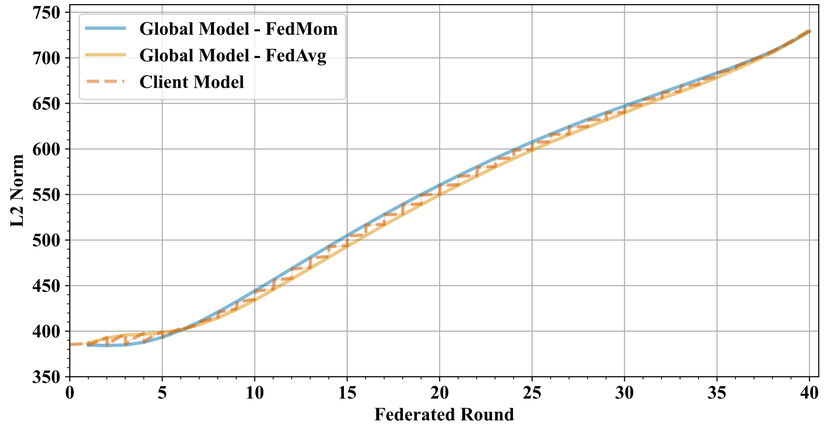
350M

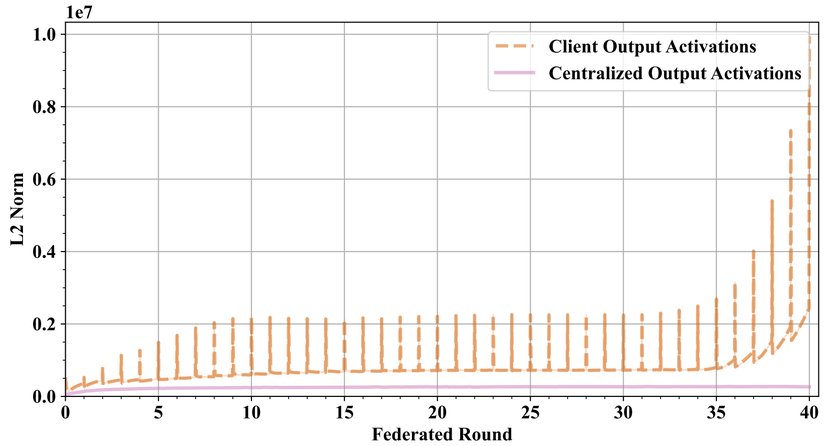
1.3B



75M
64
clients
6.25% s.r.

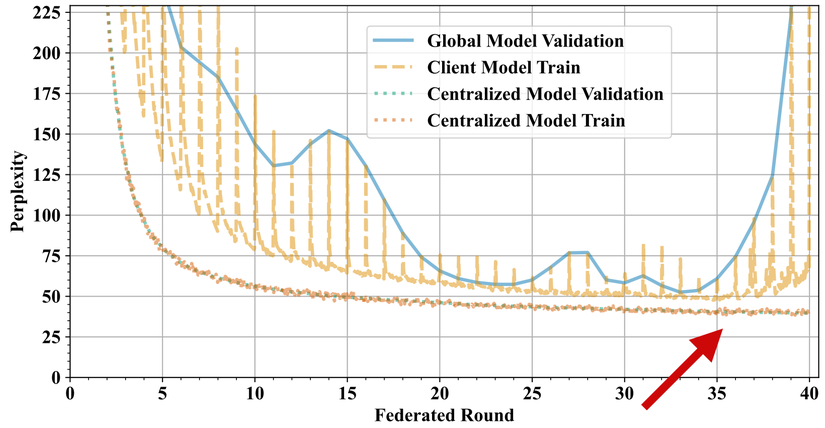


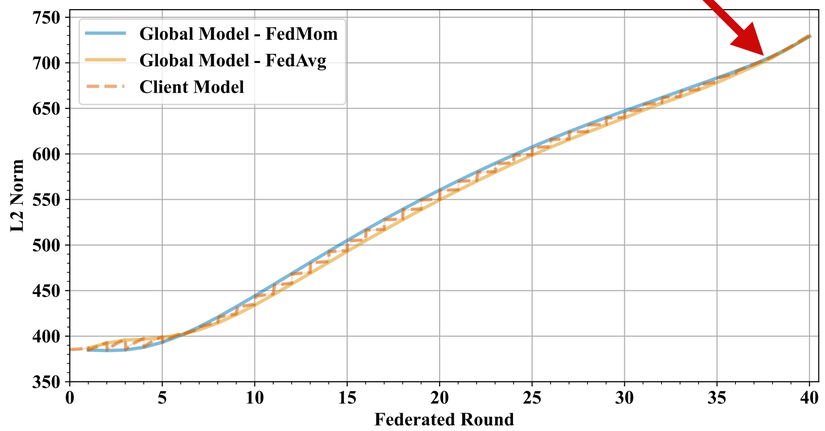


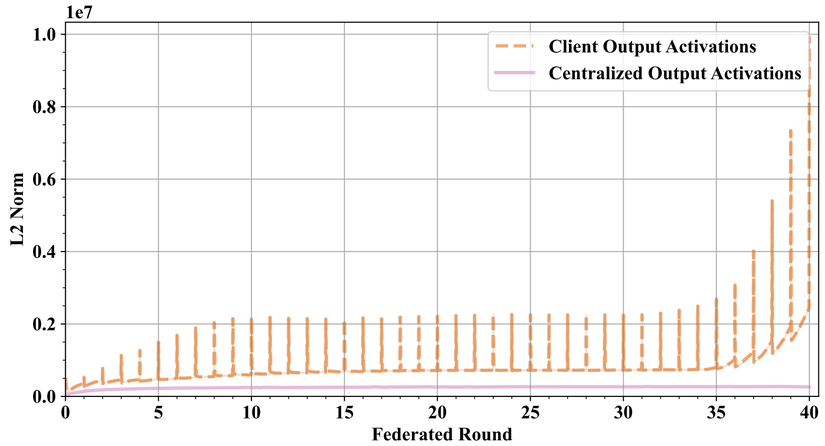


characterizing the blow-up
and
varying server learning rate

75M
64
clients
6.25% s.r.

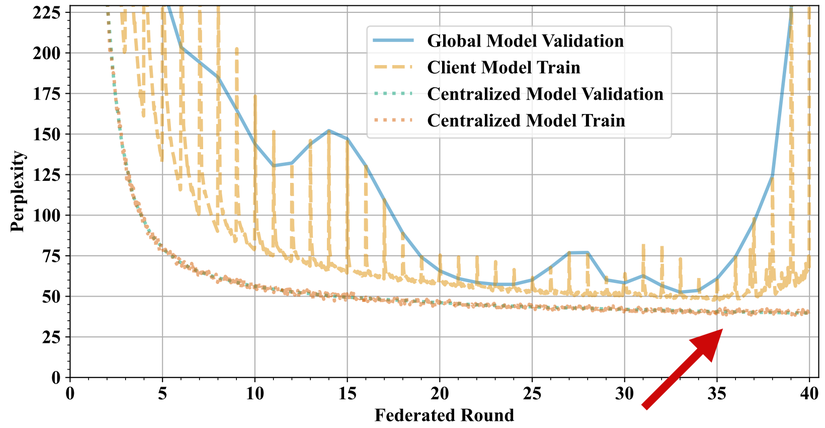


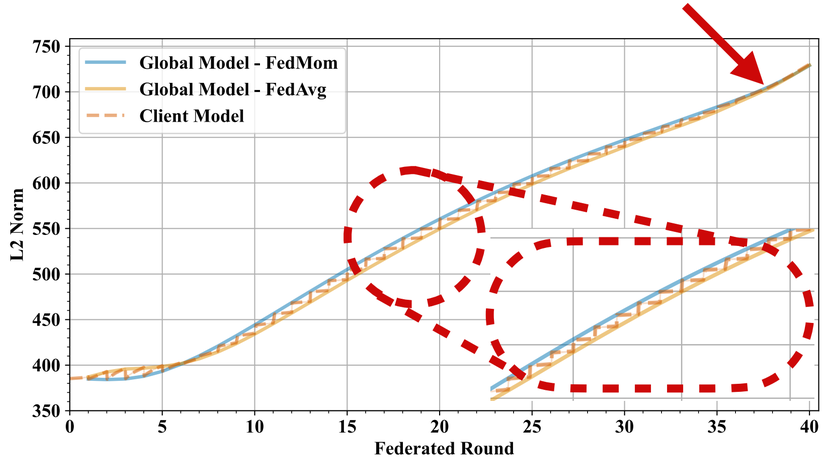


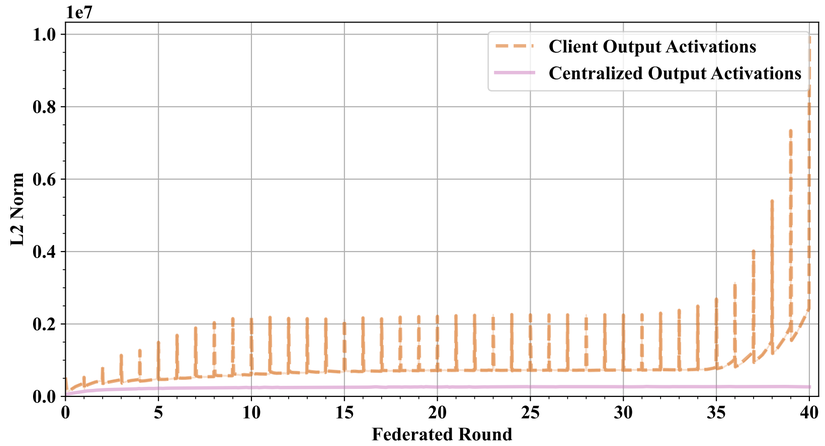


characterizing the blow-up and varying server learning rate

75M
64
clients
6.25% s.r.

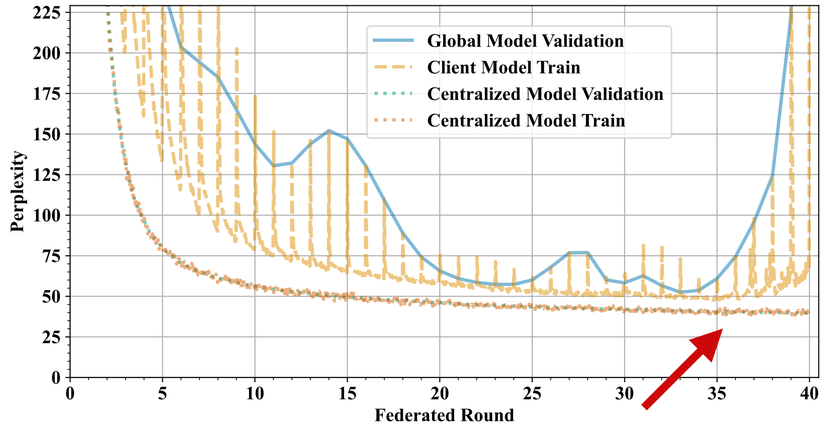


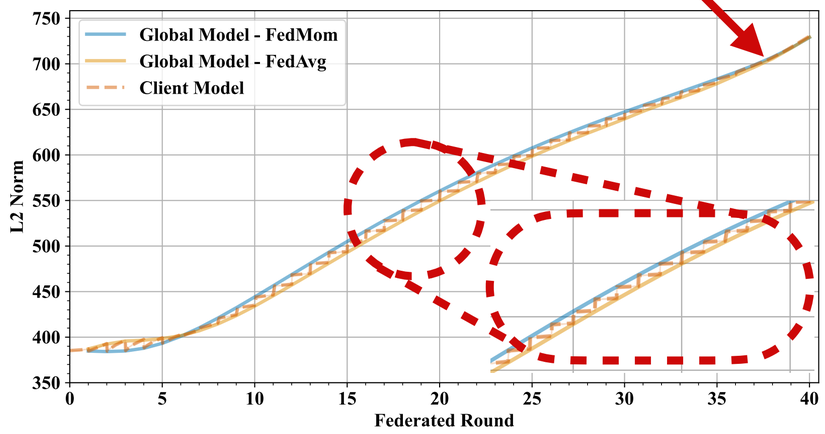


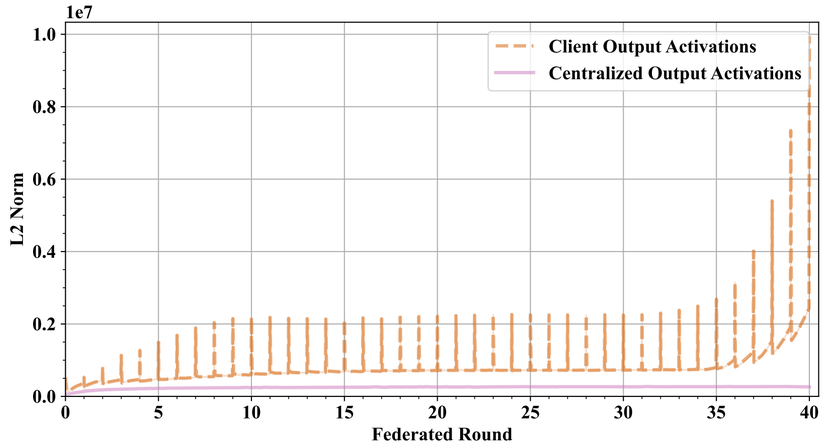


characterizing the blow-up and varying server learning rate

75M
64
clients
6.25% s.r.







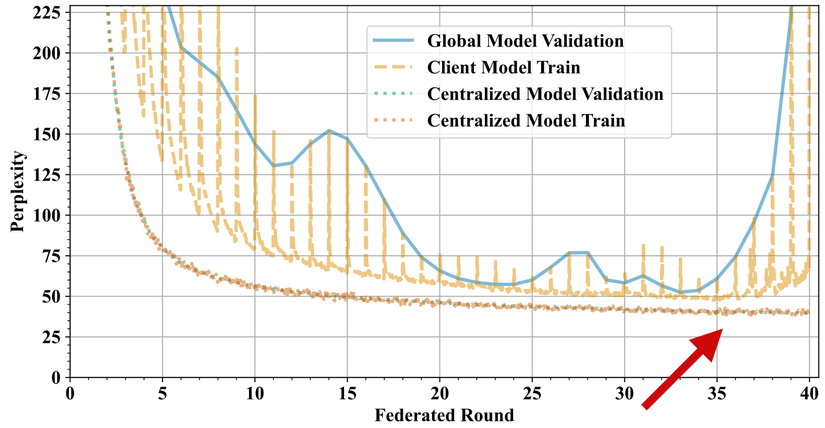
characterizing the blow-up and varying server learning rate

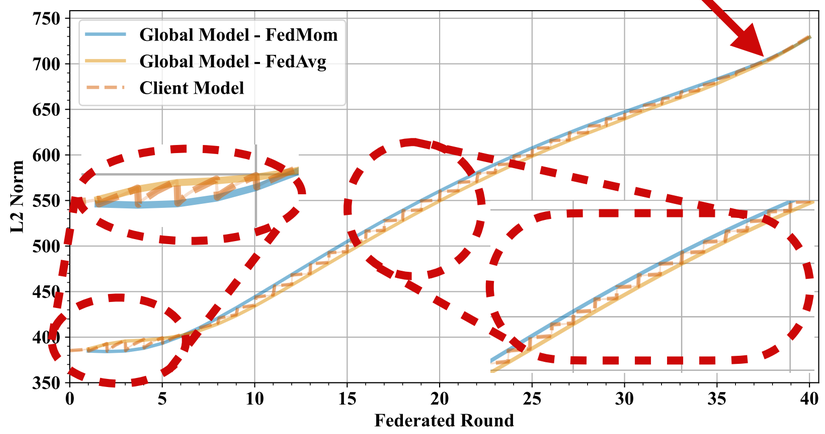
Takeaways:

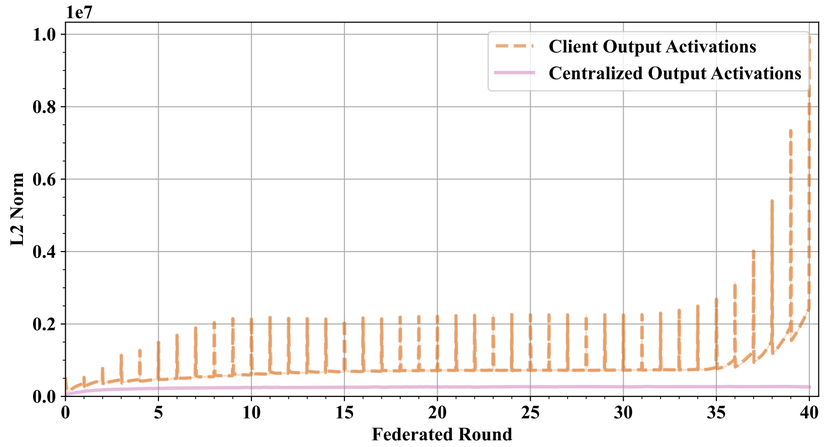
1. observe global model norms after aggregation

2. conservative server LRs help by slowing down

75M
64
clients
6.25% s.r.







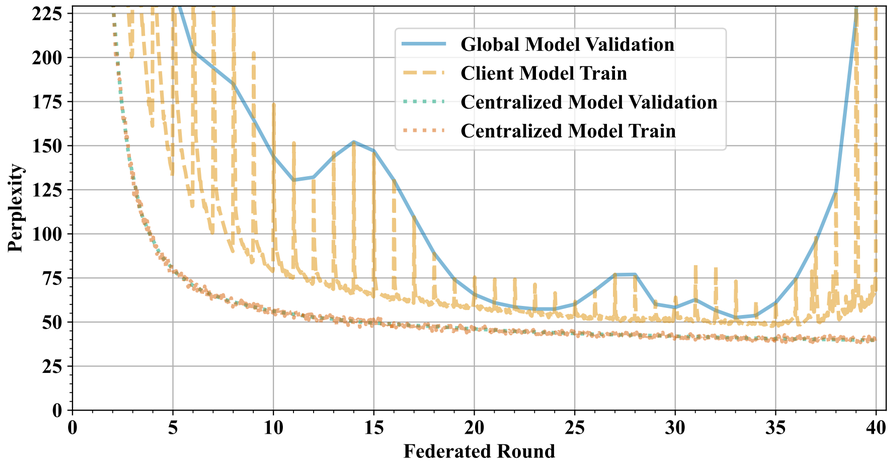
characterizing the blow-up and varying server learning rate

Takeaways:

1. observe global model norms after aggregation

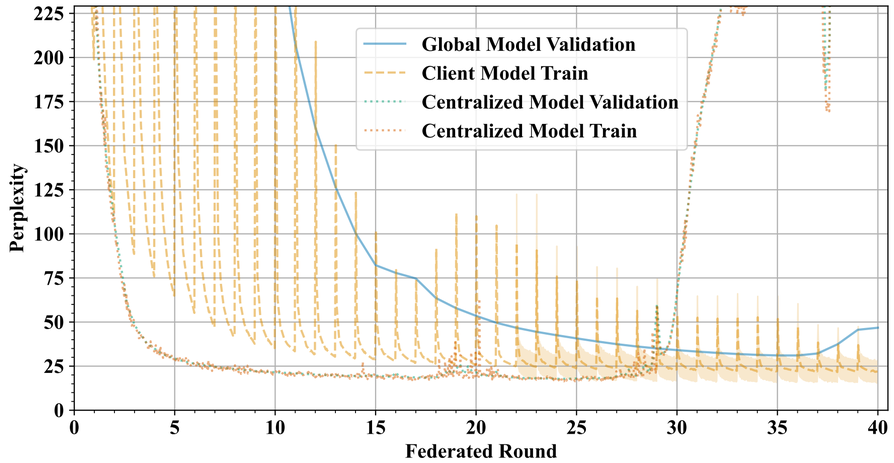
2. conservative server LRs help by slowing down

Heterogeneous Datasets

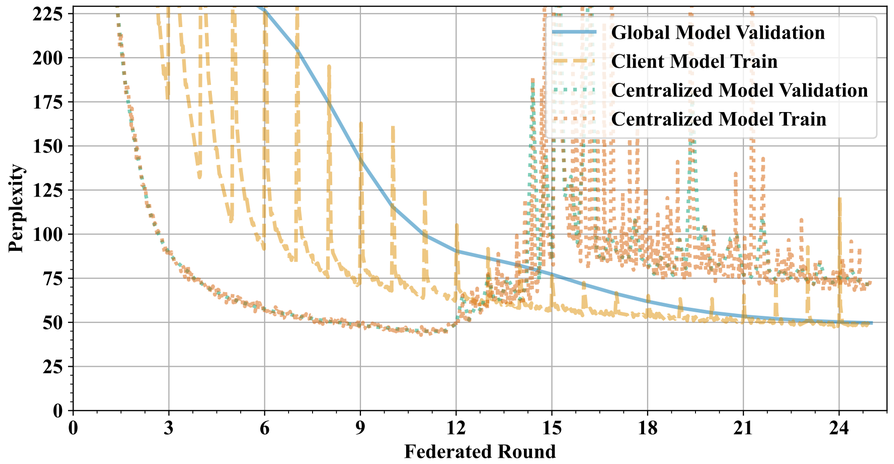


C4 (IID) - 64 clients - 6.25% sampling rate

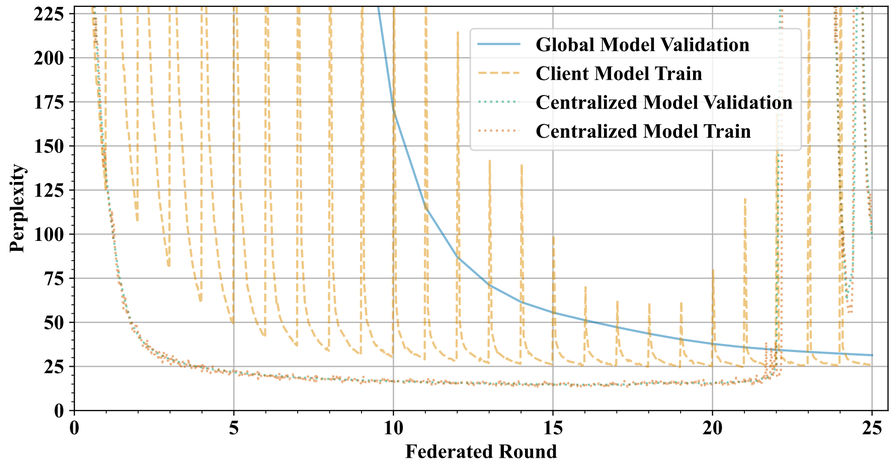
75M



The Pile (non-IID) - 64 clients - 6.25% sampling rate



125M

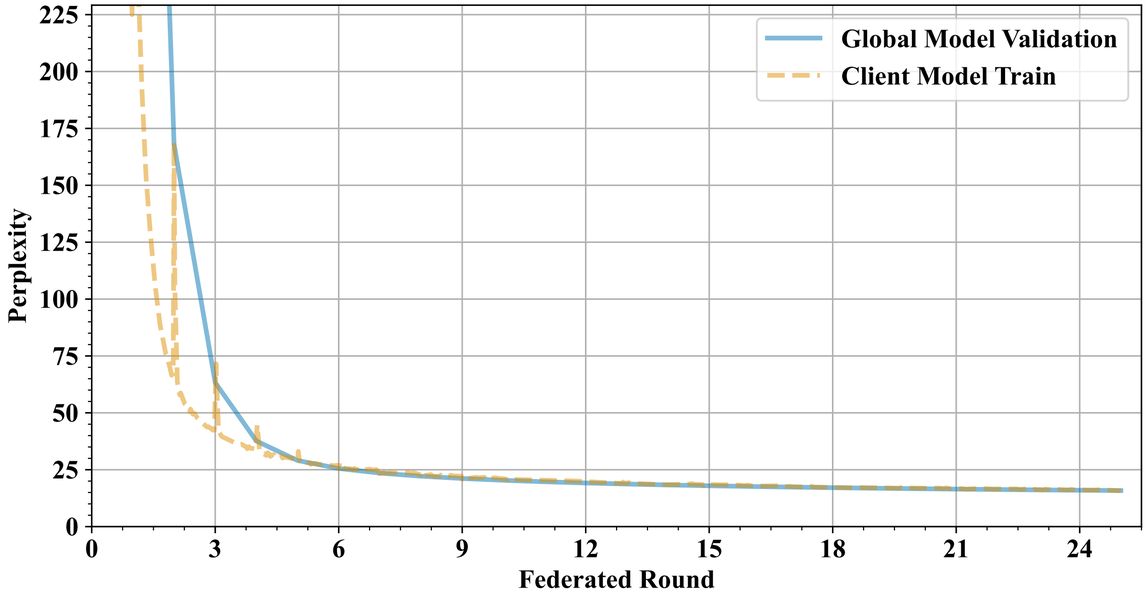


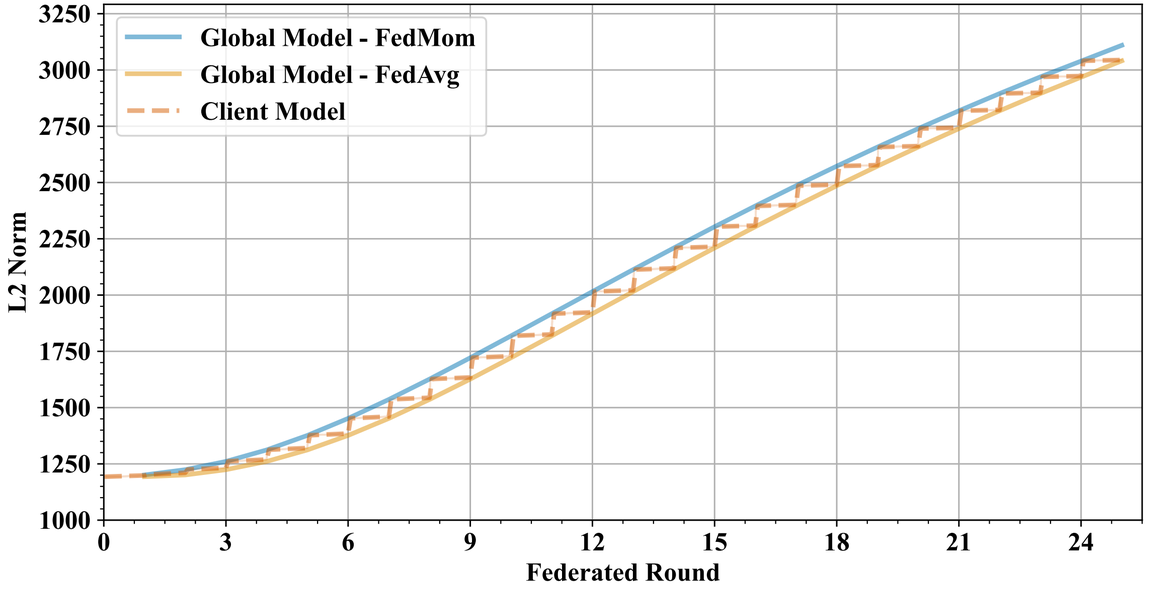
… and the 3B too!

C4 (IID) - 64 clients - 6.25% sampling rate

ServerOpt ⇒ SGD + Nestorov

ClientOpt ⇒ AdamW + cos LR with warmup





Next Steps and Open Questions

- **Distributed/Federated mixture,** what's best?
  ⇒ investigating different system configurations

  ⇒ how bandwidth and topology can impact our recipe

- More structured training recipe along the lines of **MoEs**
  ⇒ should we treat blocks and embedding layers the same way?

  ⇒ interplay between **multilingual** pre-training and FL?

- Fine-tuning and downstream performance of LLMs pre-trained on a federated population
  ⇒ more thorough **evaluation harness**

  ⇒ assess **meta-learning capabilities**

- **Privacy and memorization**

Thank you!

Any question?

reach me offline at: ls985@cam.ac.uk