Federated and Distributed Learning (Part I)

Machine Learning and Deep Learning

2023/2024

Slides credits: Riccardo Zaccone

Presenter: Riccardo Zaccone

Politecnico di Torino

Roadmap

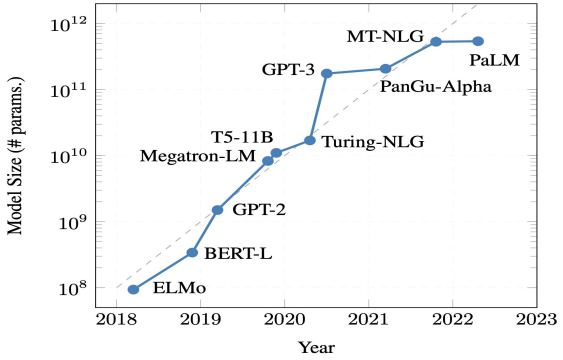
1. Distributed Training
    a. Motivations
    b. Large Batch Optimization
    c. Local Methods

2. Federated Learning
    a. Motivations
    b. Connections with LocalSGD
    c. Challenges in FL
    d. Harnessing heterogeneity: Personalized FL

Distributed Training - Motivations

Modern success of deep learning models relies on **bigger and bigger models**...

› ... but training and serving such models is becoming increasingly more difficult and costly



Evolution of the size of large pre-trained models [Treviso et al., 2022]

› Hardware speedup **scaling laws are not going to cope forever** with the limits of current algorithms and architectures

› The **centralized training** paradigm no longer matches needs of modern applications, which need to learn from **heterogeneous** and **massively decentralized data**

Distributed Training - Flavors of parallelism

- Model parallelism: useful when the model doesn't fit the single accelerator
  - vertical splitting:
    - split the model layerwise
    - operations can be pipelined to increase resource utilization (GPipe)

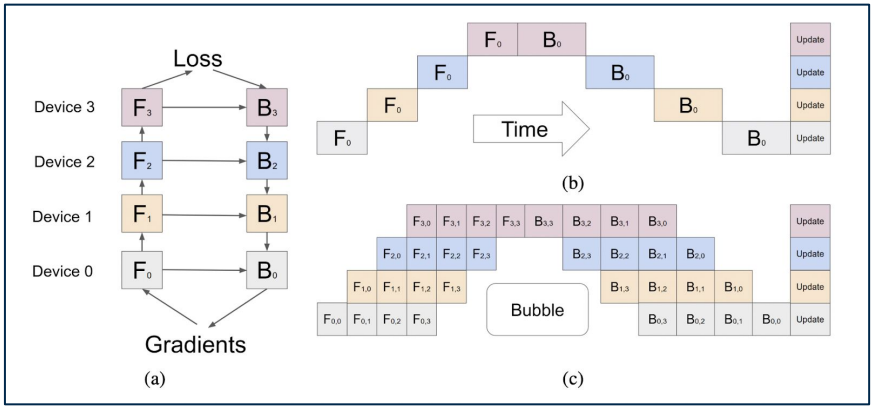


Figure 1: (a) Example of neural network with sequential layers partitioned across four accelerators. F_K is the forward function of the k-th module. B_K is the corresponding backward function. **(b)** Naive model parallelism with severe under-utilization due to the sequential dependency of the network. **(c)** Pipeline parallelism enables different accelerators to work on different micro-batches simultaneously. Gradients are applied synchronously at the end.

Distributed Training - Flavors of parallelism

- Model parallelism: useful when the model doesn't fit the single accelerator
  - horizontal splitting (tensor parallelism):
    - efficient utilization of multiple GPUs
    - requires very fast network, better not to use more than 1 node

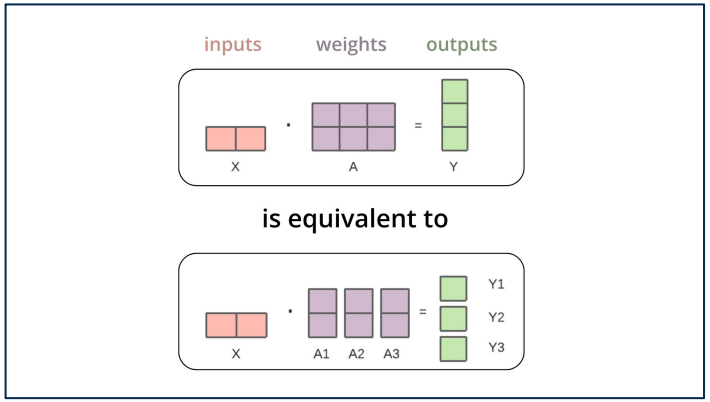


Figure 2: illustration of tensor parallelism: the matrix multiplication is equivalent to splitting the weight tensor column-wise, multiplying each column with the input separately, and then concatenating the separate outputs (source HuggingFace)

Distributed Training - Flavors of parallelism

- Data parallelism: most useful when
  - the batch size doesn't fit the accelerator
  - we aim to speed up the training

- How:
  - Replicate the model on each accelerator
  - Make each replica process a different chunk of data
  - Aggregate the results during the backward phase
  - Distribute the gradients

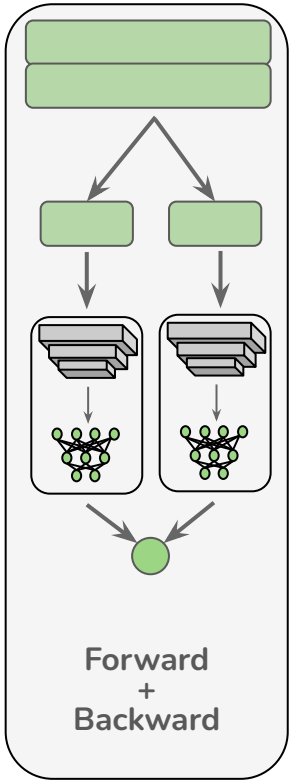


Figure 3: standard data parallel training

Distributed Training - Flavors of parallelism

- Hybrid approaches:

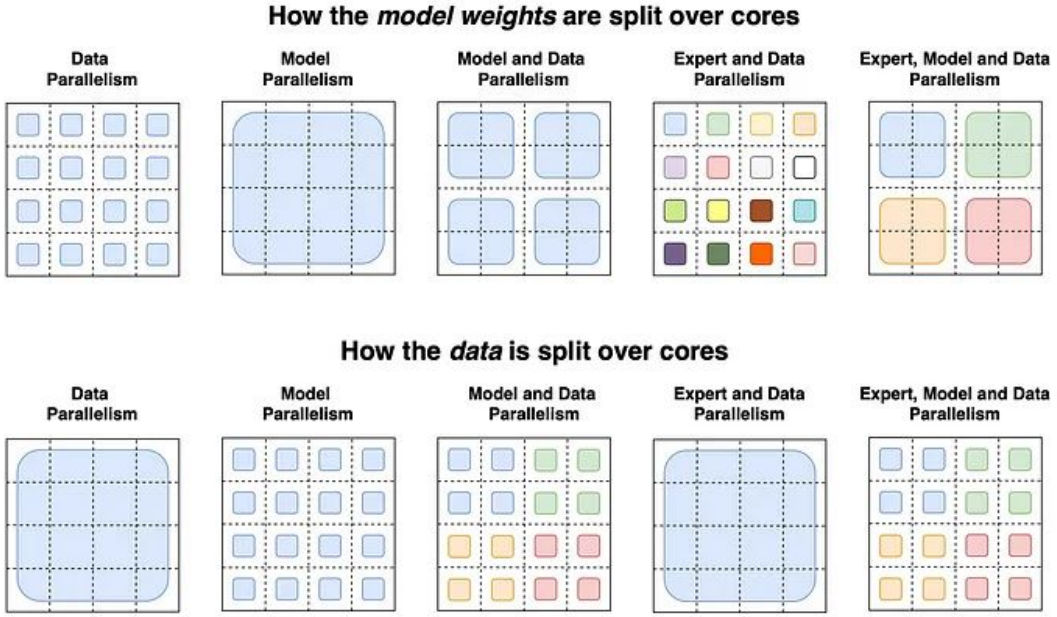


Figure 4: Data and weight partitioning strategies of Google Switch Transformers

Speeding up training with data parallelism

🌏 Large Batch optimizers:

- 💡 **Intuition:** to use N accelerators, increase the batch size xN and process in parallel
- ⚙️ **How:** apply layerwise gradient scaling + clever learning rate warmup

🏠 Local methods:

- 💡 **Intuition:** reduce communication by means of local work
- ⚙️ **How:** perform more than one optimization steps before model's parameters synchronization

Politecnico di Torino

LARGE BATCH OPTIMIZATION FOR DEEP LEARNING: TRAINING BERT IN 76 MINUTES

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, Cho-Jui Hsieh

Ahmad Sidani, Marco Mungai Coppolino, Gluseppe Acquaviva

Outline:

- Introduction and Background
- Challenges with large batch stochastic optimization methods
- Introducing LAMB
- Contributions and Practical Implications
- Preliminaries
- Algorithms
- Experiments
- Conclusion
- References

Introduction and Background

- Training large deep neural networks is computationally challenging.
- Need for new adaptive optimization techniques.
- Interest in large batch stochastic optimization methods as a solution.

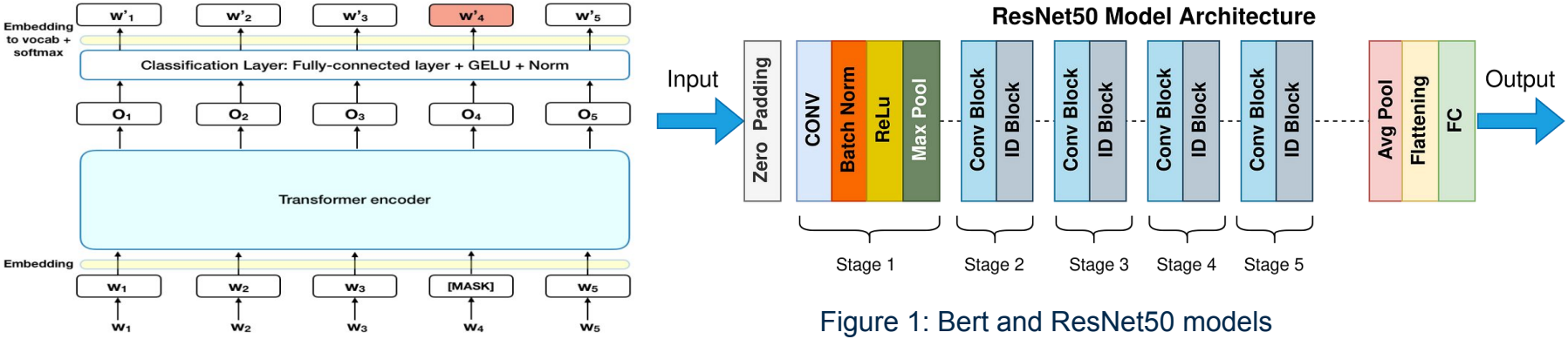


Figure 1: Bert and ResNet50 models

Challenges with large batch stochastic optimization methods

- Current problem with current large batch optimization like generalization problem
- Up to certain batch size linearly scaling the learning rate can speed up the training:
  - Can't scale learning rate too soon: Solution provided **Warmup**
  - When you reach certain batch size, there is no point of using learning rate
- An adaptive learning rate as a solution **LARS algorithm**:
  - Success with ResNet
  - Failure with BERT (wasn't accurate)

Introducing LAMB

- Layerwise Adaptive Large Batch Optimization (LAMB)
- General adaptation strategy for large batch learning
- Convergence analysis of LAMB and LARS
- Superior empirical performance of LAMB
- Improvements for BERT and ResNet-50 training

PRELIMINARIES

- In this paper, they study nonconvex stochastic optimization problems of the form:

$$\min_{x \in \mathbb{R}^d} f(x) := \mathbb{E}_{s \sim \mathbb{P}}[\ell(x, s)] + \frac{\lambda}{2}\|x\|^2$$

  where the goal is to minimize the average value of a function.

- $\lambda/2 \, // \, x \, //$ is the L2 regularization term
- λ is the regularization parameter.
- ℓ is a smooth (possibly non convex) function and P is a probability distribution.Here,x corresponds to model parameters

PRELIMINARIES

They assume function $\ell(x)$ is Li-smooth with respect to x(i), i.e., there exists a constant Li such that

$$\|\nabla_i \ell(x, s) - \nabla_i \ell(y, s)\| \leq L_i \|x^{(i)} - y^{(i)}\|, \quad \forall\, x, y \in \mathbb{R}^d, \text{ and } s \in \mathcal{S}$$

PRELIMINARIES

Stochastic gradient descent (SGD) is one of the simplest first-order algorithms for solving problem in Equation 1. The update at the tth iteration of SGD is of the following form:

$$x_{t+1} = x_t - \eta_t \frac{1}{|\mathcal{S}_t|} \sum_{s_t \in \mathcal{S}_t} \nabla \ell(x_t, s_t) + \lambda x_t,$$

where S_t is set of b random samples drawn from the distribution P.

ALGORITHMS

General Strategy: let's consider an iterative base algorithm *A* (e.g. SGD or ADAM), with the following layerwise update rule:

$$x_{t+1} = x_t + \eta_t u_t,$$

where u_t is the update made by A at time step t.

1. The update is normalized to unit l2-norm.
2. The learning rate is scaled by Φ(xt),such a scaling is done layer wise

$$x_{t+1}^{(i)} = x_t^{(i)} - \eta_t \frac{\phi(\|x_t^{(i)}\|)}{\|g_t^{(i)}\|} g_t^{(i)},$$

LARS VS. LAMB ALGORITHM

Algorithm 1 LARS

Input: $x_1 \in \mathbb{R}^d$, learning rate $\{\eta_t\}_{t=1}^T$, parameter $0 < \beta_1 < 1$, scaling function ϕ, $\epsilon > 0$

Set $m_0 = 0$

for $t = 1$ **to** T **do**

Draw b samples S_t from $\mathbb{P}$

Compute $g_t = \frac{1}{|\mathcal{S}_t|} \sum_{s_t \in \mathcal{S}_t} \nabla \ell(x_t, s_t)$

$m_t = \beta_1 m_{t-1} + (1 - \beta_1)(g_t + \lambda x_t)$

$x_{t+1}^{(i)} = x_t^{(i)} - \eta_t \frac{\phi(\|x_t^{(i)}\|)}{\|m_t^{(i)}\|} m_t^{(i)}$ for all $i \in [h]$

end for

Algorithm 2 LAMB

Input: $x_1 \in \mathbb{R}^d$, learning rate $\{\eta_t\}_{t=1}^T$, parameters $0 < \beta_1, \beta_2 < 1$, scaling function ϕ, $\epsilon > 0$

Set $m_0 = 0$, $v_0 = 0$

for $t = 1$ **to** T **do**

Draw b samples S_t from $\mathbb{P}$.

Compute $g_t = \frac{1}{|\mathcal{S}_t|} \sum_{s_t \in \mathcal{S}_t} \nabla \ell(x_t, s_t)$.

$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

$m_t = m_t / (1 - \beta_1^t)$

$v_t = v_t / (1 - \beta_2^t)$

Compute ratio $r_t = \frac{m_t}{\sqrt{v_t} + \epsilon}$

$x_{t+1}^{(i)} = x_t^{(i)} - \eta_t \frac{\phi(\|x_t^{(i)}\|)}{\|r_t^{(i)} + \lambda x_t^{(i)}\|} (r_t^{(i)} + \lambda x_t^{(i)})$

end for

Experimental Setup

The LAMB optimizer was tested on two large batch training tasks:

- **BERT:** multi-layer bidirectional Transformer encoder
- **RESNET-50:** CNN architecture designed to support hundreds or thousands of convolutional layers, used for image recognition tasks

To test its robustness, only minimal hyperparameter tuning was performed

- **Decaying Parameters:** fixed at $\beta_1 = 0.9$ and $\beta_2 = 0.999$
- **Learning rate:** polynomially decaying learning rate of $\eta_t = \eta_0({}^{1-t}/_{T})$

All the experiments are executed using a TPUv3

Adaptive Learning Rate

- **Square Root Rule:** when the batch size is scaled by a factor k, the learning rate should scale by √k
- **Linear-epoch Warmup Scheduling**: a gradually increase of the learning rate at the start of training

Batch Size	512	1K	2K	4K	8K	16K	32K
Learning Rate	$\frac{5}{2^{3.0}\times 10^3}$	$\frac{5}{2^{2.5}\times 10^3}$	$\frac{5}{2^{2.0}\times 10^3}$	$\frac{5}{2^{1.5}\times 10^3}$	$\frac{5}{2^{1.0}\times 10^3}$	$\frac{5}{2^{0.5}\times 10^3}$	$\frac{5}{2^{0.0}\times 10^3}$
Warmup Ratio	$\frac{1}{320}$	$\frac{1}{160}$	$\frac{1}{80}$	$\frac{1}{40}$	$\frac{1}{20}$	$\frac{1}{10}$	$\frac{1}{5}$
F1 score	91.752	91.761	91.946	91.137	91.263	91.345	91.475
Exact Match	85.090	85.260	85.355	84.172	84.901	84.816	84.939

Table 1: Untuned LAMB for BERT training across different batch sizes

Bert Training: Two Stage Large Batch Optimization

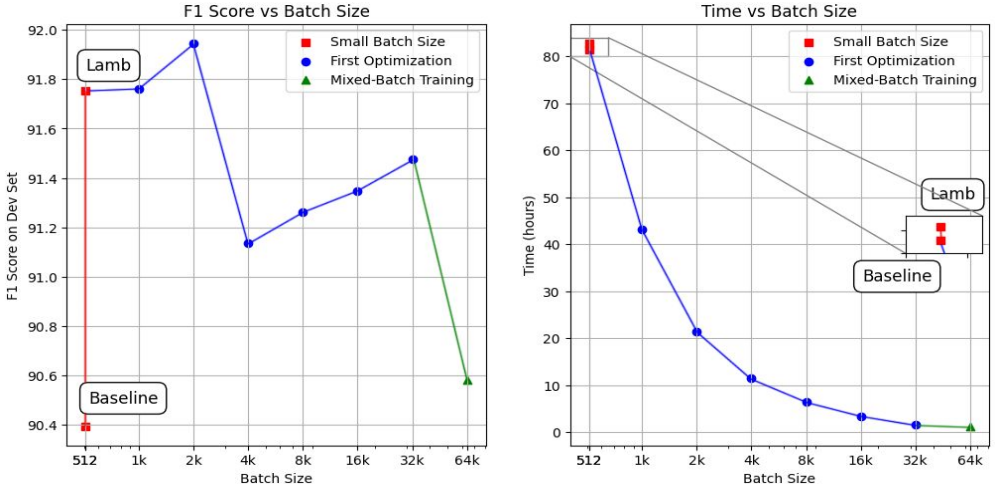


Figure 2: Comparison with ADAMW and LARS

Batch Size	512	1K	2K	4K	8K	16K	32K
LARS	90.717	90.369	90.748	90.537	90.548	89.589	diverge
LAMB	91.752	91.761	91.946	91.137	91.263	91.345	91.475

Table 2: Comparison with ADAMW and LARS

ImageNet Training With ResNet-50

- Training a ResNet-50 model on the ImageNet dataset is an industry-standard task used to evaluate the performance of machine learning systems
- By employing layerwise adaptive learning rates, LARS is able to train ResNet in few minutes, achieving 76.3% accuracy top-1 accuracy in 90 epochs

optimizer	adagrad/adagrad+	adam/adam+	adamw/adamw+	momentum	lamb
Accuracy	0.5538/0.7201	0.6604/0.7348	0.6727/0.7307	0.7520	0.7666

Table 3: Comparison between optimizers: batch size of 16K

Conclusions

- Large batch techniques are essential tools for accelerating deep neural network training
- LAMB is an optimizer designed to support adaptive element-wise updating, and layer-wise learning
- LAMB has demonstrated superior efficiency compared to other optimizers
- Research Updates: ResNet STrikes Back

References

1. Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, Cho-Jui Hsieh. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. arXiv preprint arXiv:1904.00962
2. Machine Learning and Deep Learning Slides
3. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270
4. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

Politecnico di Torino

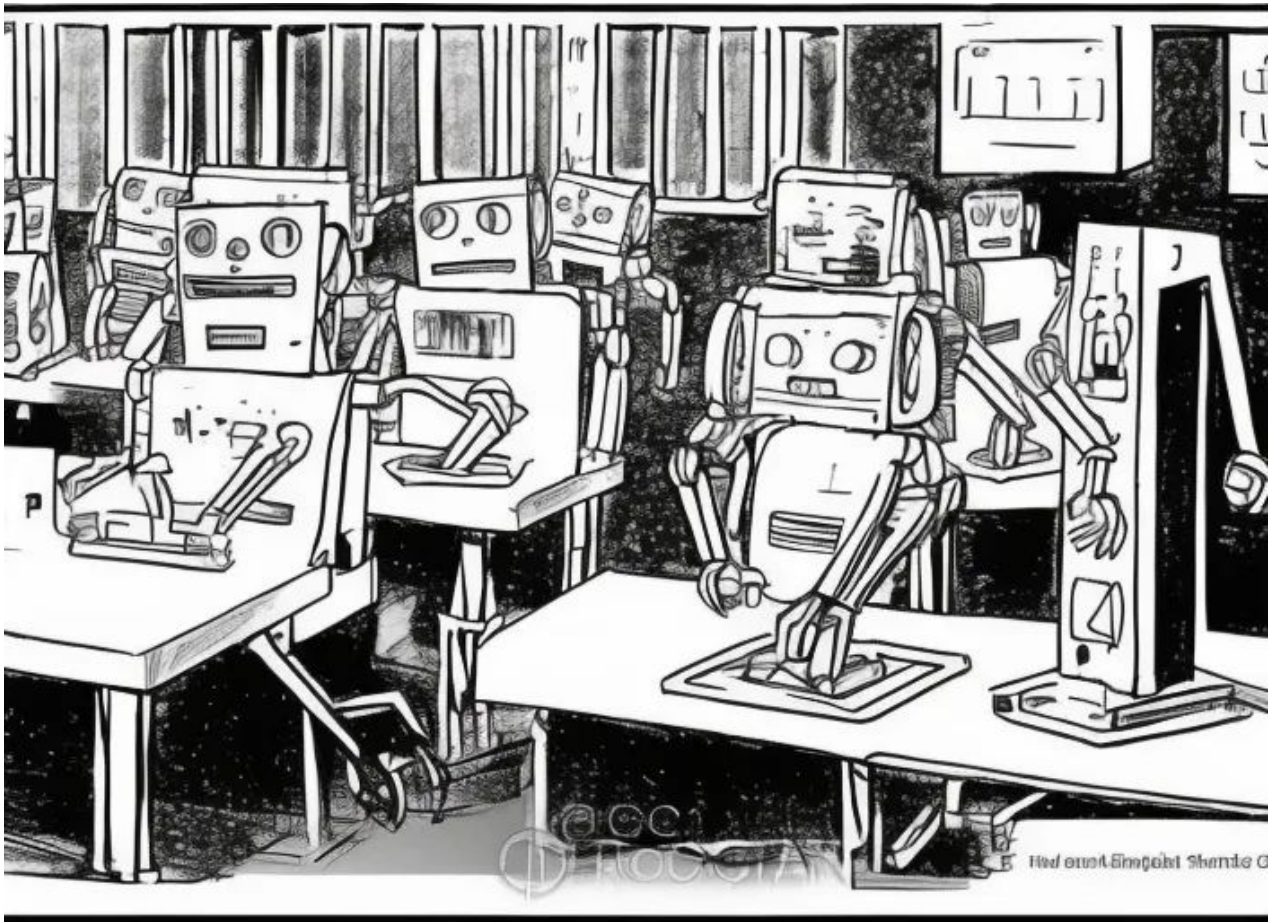
Thank you for your attention!

Politecnico di Torino

Don’t Use Large Mini-Batches, Use Local SGD

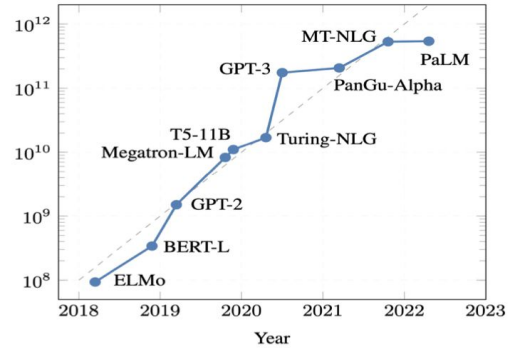
Tao Lin, Sebastian U. Stich, Kumar Kshitij Patel, Martin Jaggi

Erfan Bayat, Elia Faure-Rolland, Valeria Petrianni



Why use Distributed Learning?

1. Data volume
2. Model complexity
3. Speed and efficiency
4. Scalability
5. Resource utilization



Enable **parallelization** while being communication efficient

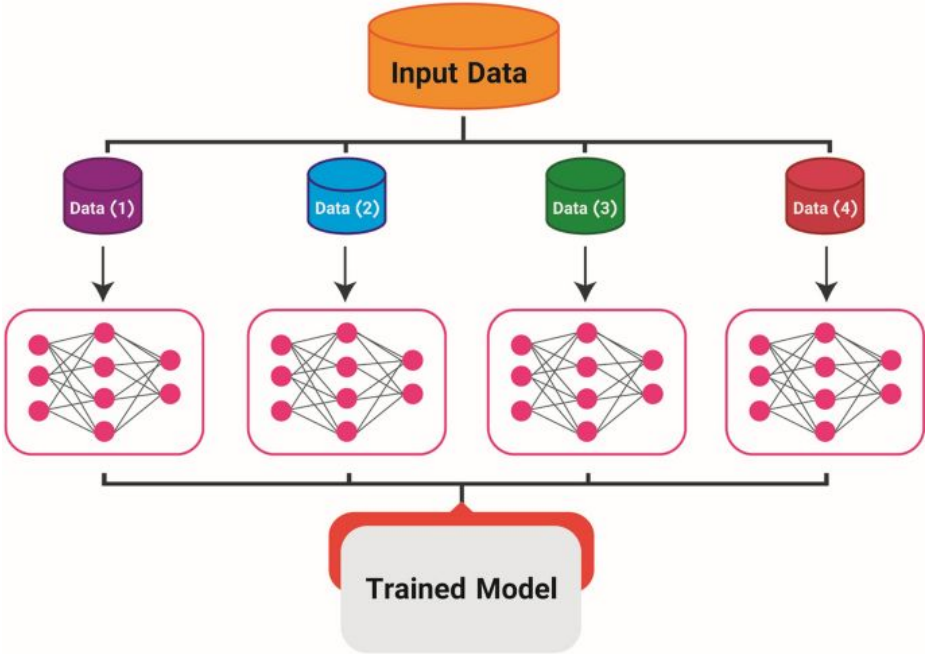
Exhibit good **generalization** behaviour

What is **mini-batch SGD**?

Gradient computed on small subsets (mini-batches) of data.

Process:

- Data is divided into mini-batches
- Each worker computes gradients on assigned mini-batch
- Gradients are aggregated and model parameters are updated



Distributed learning approaches

Large batch SGD

To use N accelerators, increase the batch size xN and process in parallel

- It uses a significantly larger batch size
- Scaling Training
- Aggregates gradients frequently
- Suitable for stable convergence

But what is the problem?

Local SGD

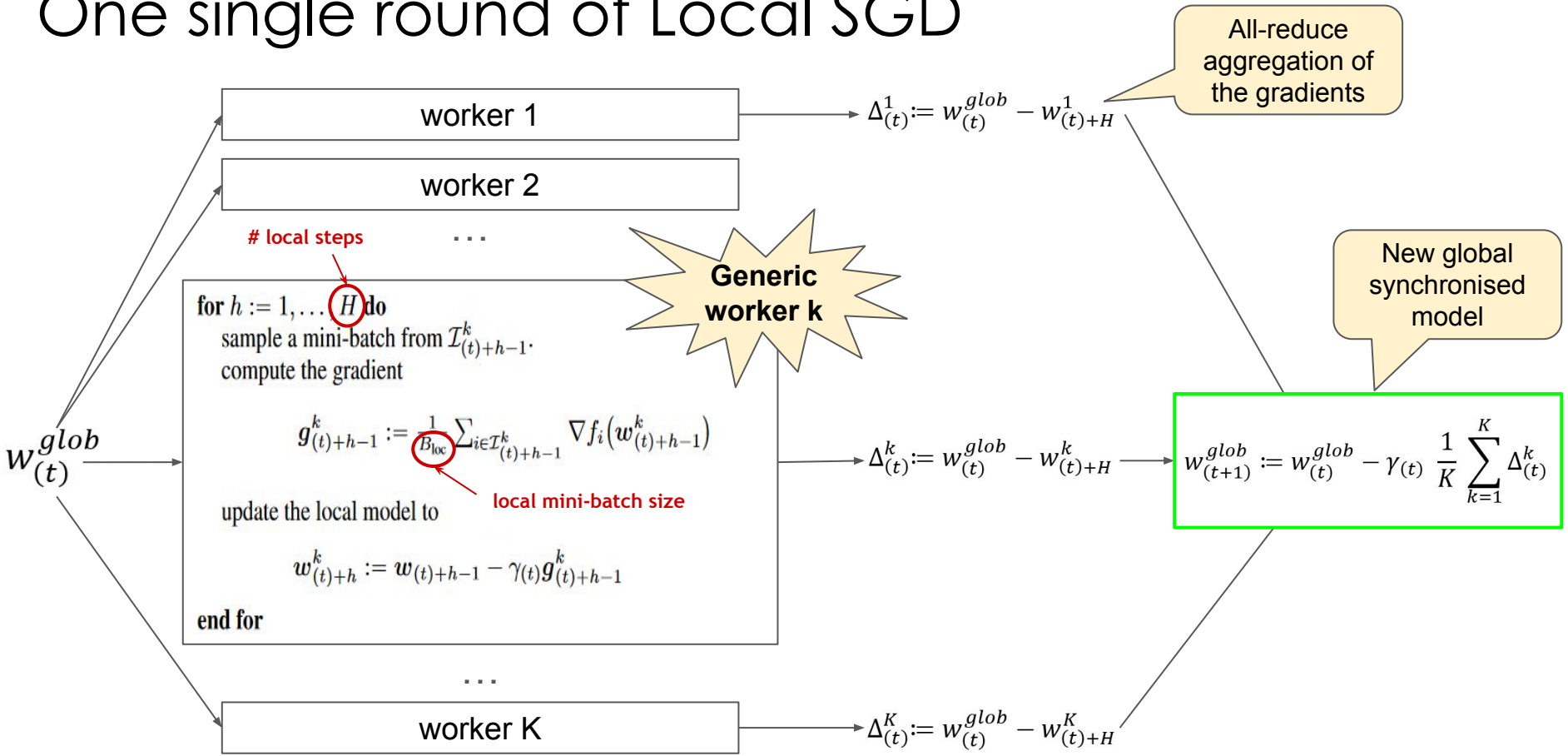
- Reduces communication overhead
- Performs multiple local updates before synchronization

LOCAL SGD IN DETAIL

Main **idea** of Local SGD

LOCAL SGD NEEDS H TIMES FEWER COMMUNICATION ROUNDS THAN MINI BATCH SGD

One single round of Local SGD



Mini-batch SGD as a **special case** of Local SGD

Loop inside each worker before updating the global model

for $h := 1, \ldots, H$ **do**

sample a mini-batch from $\mathcal{I}^k_{(t)+h-1}$.

compute the gradient

$$g^k_{(t)+h-1} := \frac{1}{B_{\text{loc}}} \sum_{i \in \mathcal{I}^k_{(t)+h-1}} \nabla f_i\left(w^k_{(t)+h-1}\right).$$

update the local model to

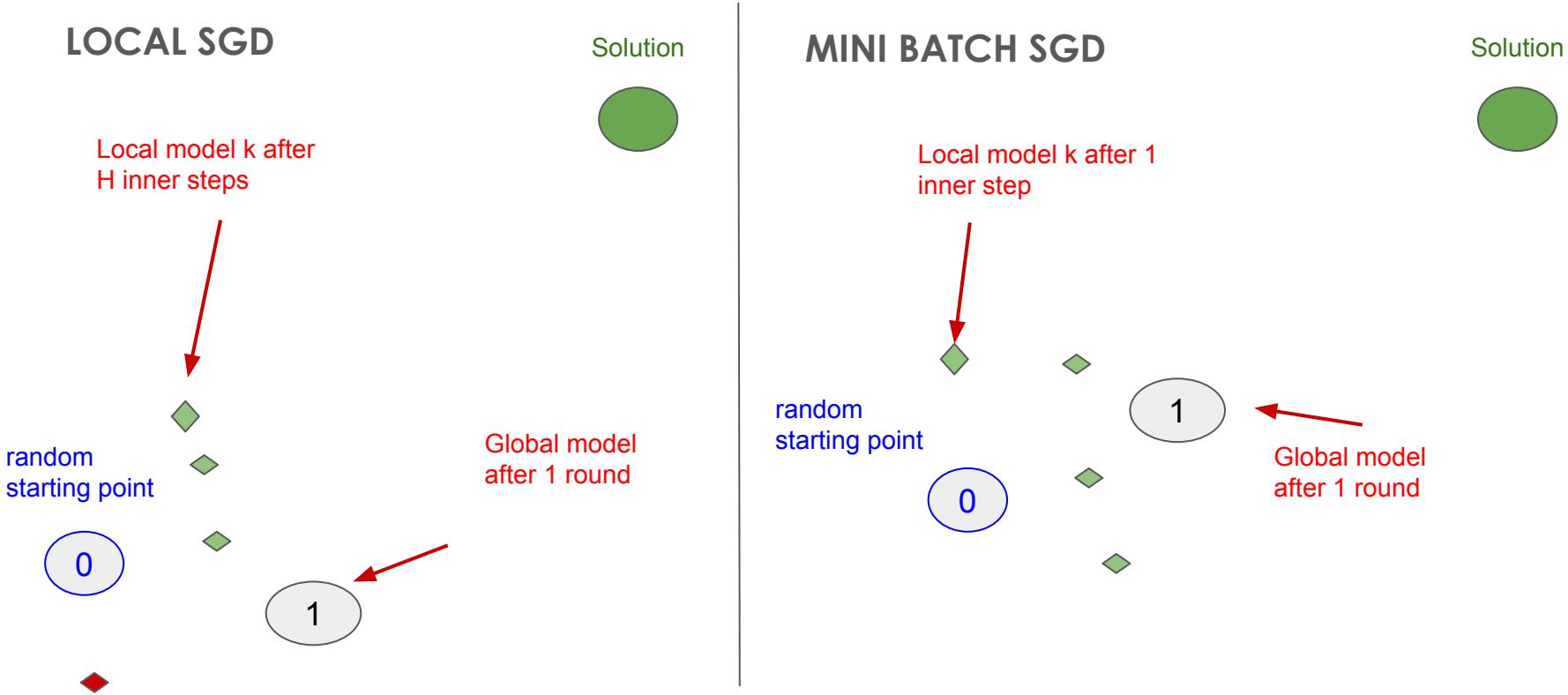
$$w^k_{(t)+h} := w_{(t)+h-1} - \gamma_{(t)} g^k_{(t)+h-1}.$$

end for

If **H=1** (update immediately)

MINI BATCH SGD!

Issues of Local SGD



Issues of Local SGD

HOW TO SOLVE THEM?

Post-local SGD

Two phases of training:

1. t' steps with mini-batch SGD

2. Local SGD

Number of local steps

$$H_{(t)} = \begin{cases} 1, & \text{if } t \leq t' \\ H, & \text{if } t > t' \end{cases}$$

iteration dependent

EXPERIMENTAL RESULTS

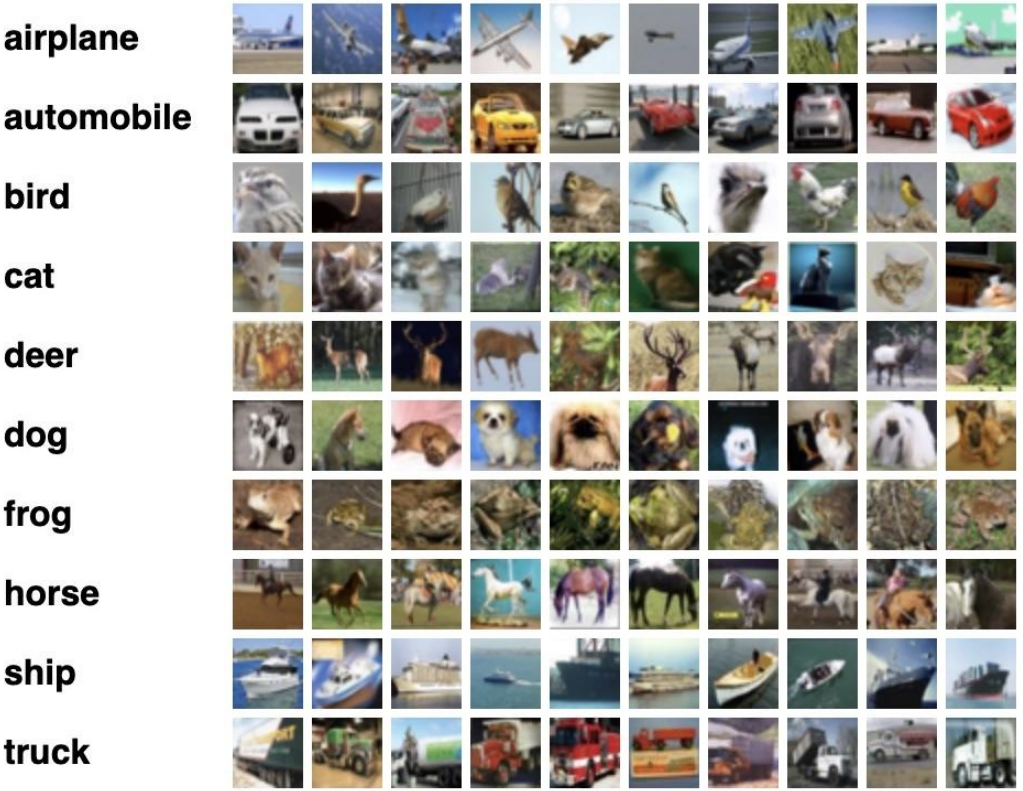
Experimental results: **setup**

Datasets

- CIFAR-10
- CIFAR-100

Network Architectures

- ResNet-20



Better scalability on CIFAR, in terms of **time-to-accuracy**

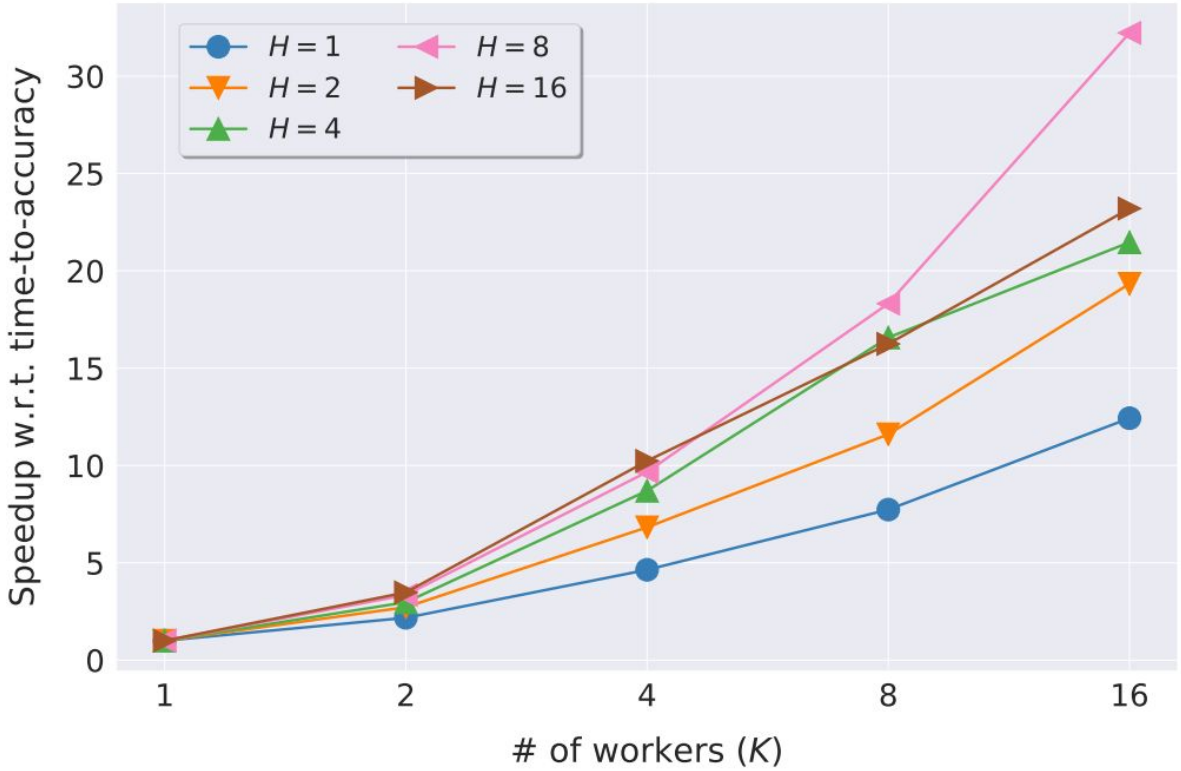


Figure 1: Scaling behavior of local SGD in clock-time for increasing number of workers K, for different number of local steps H.

Local SGD outperforms mini-batch SGD at the **same effective batch size and communication ratio**

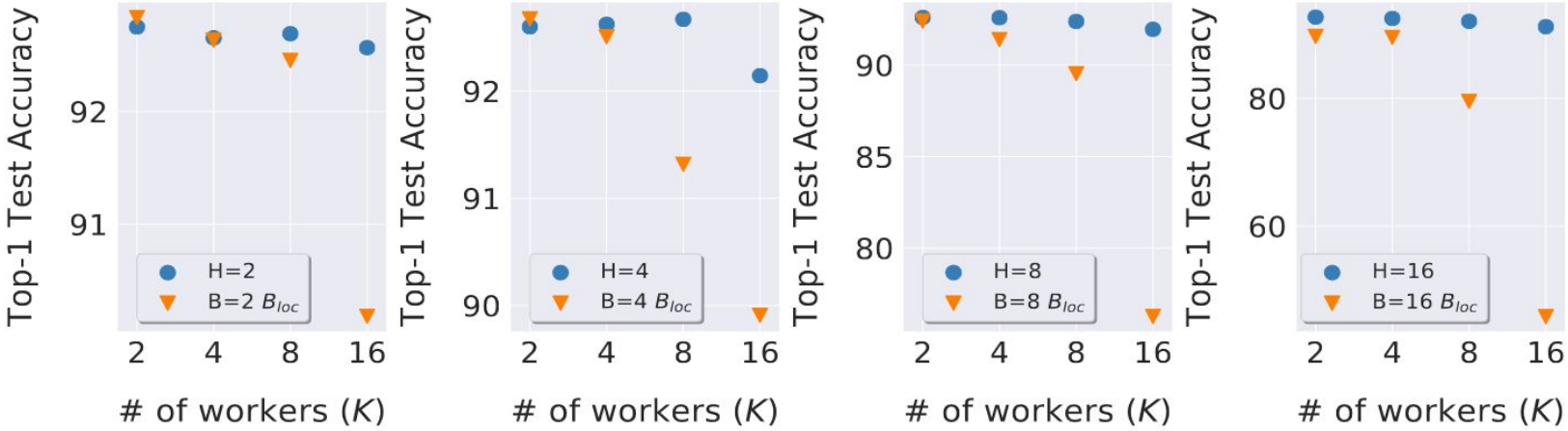


Figure 2: Top-1 test accuracy of local SGD (**circle**) vs. mini-batch SGD (**triangle**), with same effective batch size B = H*B_loc per worker.

(Post)-local SGD closes the generalization gap of large-batch training

Algorithm	Top-1 acc.	Effect. batch size	Algorithm	Top-1 acc.	Effect. batch size
Mini-batch SGD ($K=4$)	92.6%	$KB=512$	Mini-batch SGD ($K=16$)	92.5%	$KB=2048$
Mini-batch SGD ($K=4$)	89.5%	$KB=8192$	Mini-batch SGD ($K=16$)	76.3%	$KB=16384$
Mini-batch SGD ($K=4$)	92.5%	$KB=512 \rightarrow 8192$	Mini-batch SGD ($K=16$)	92.0%	$KHB_{\text{loc}}=2048 \rightarrow 16384$
Local SGD ($H=16$, $K=4$)	92.5%	$KHB_{\text{loc}}=8192$	Local SGD ($H=8$, $K=16$)	92.0%	$KHB_{\text{loc}}=16384$
Post-local SGD ($H=16$, $K=4$)	**92.7%**	$KHB_{\text{loc}}=512 \rightarrow 8192$	Post-local SGD ($H=8$, $K=16$)	**92.9%**	$KHB_{\text{loc}}=2048 \rightarrow 16384$

Table 1: Test performance (generalization) for local SGD variants and mini-batch SGD.

Post-local SGD training for different **H** and **K**

Figure 3(a): Performance of post-local SGD **for different number of local steps H** for K=16. H=1 is mini-batch SGD.

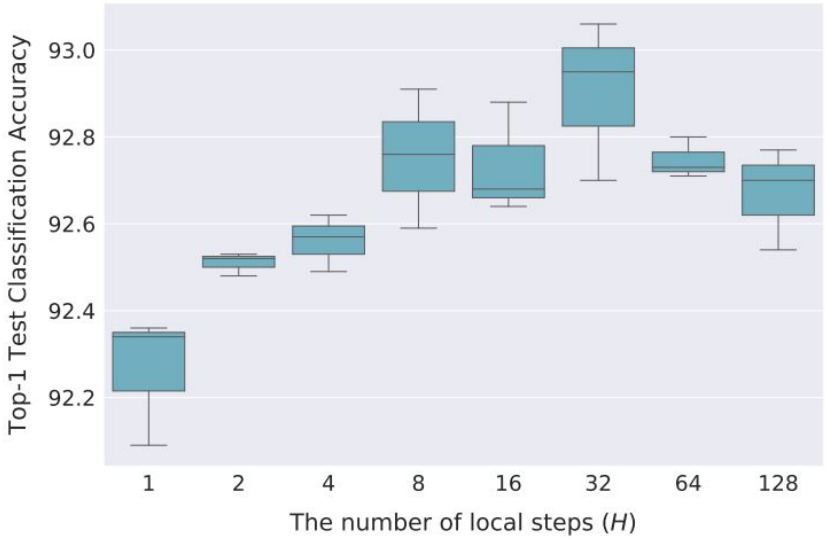
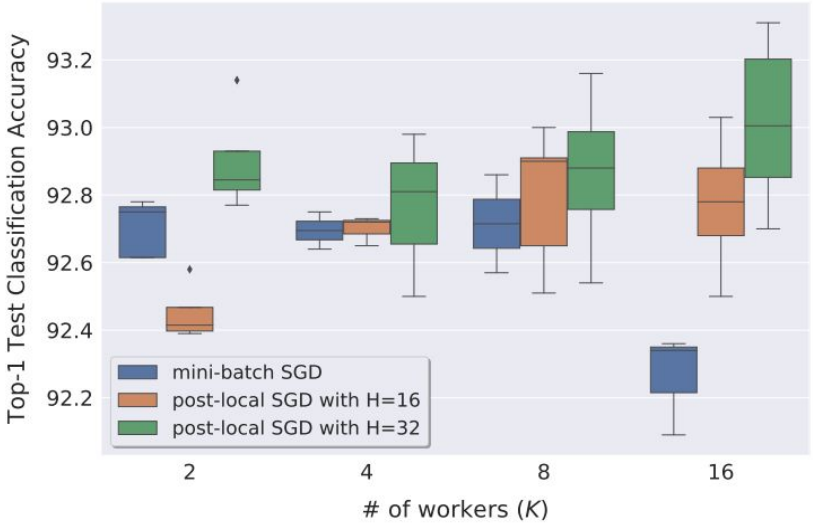


Figure 3(b): Performance of post-local SGD **for different number of workers K** for H=16 and H=32.



Post-local SGD converges to **flatter minima**

- Flatter minima are preferred for better generalization.

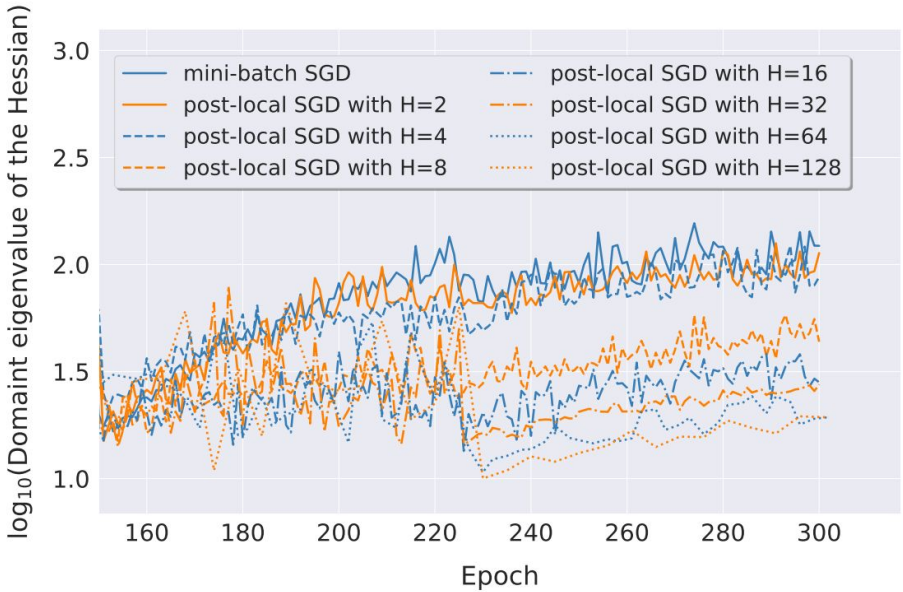
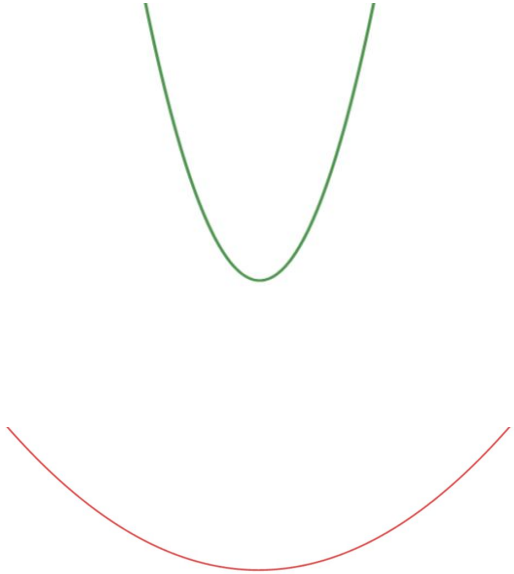


Figure 4: The dominant eigenvalue of the Hessian for different local minima, for model trained from different schemes.



Why Local SGD?

- Better generalization performance
- Significant speed-up
- Easier approach

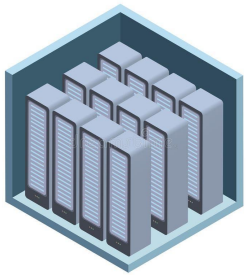
Politecnico di Torino

Thank you for your attention!

Distributed and Federated Learning

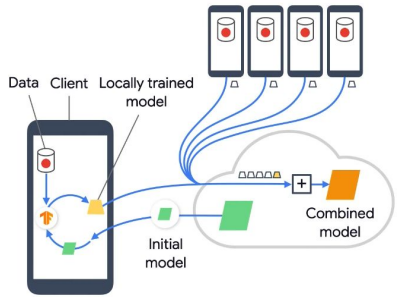
Distributed Learning:

- 💪 Nodes are powerful servers in a data center
- ⚡ Interconnections are fast and reliable
- 🔀 Data can be exchanged between nodes



Federated **L**earning:

- 📱 Nodes are usually heterogeneous, low-powered and decentralized edge devices
- 🐢 Interconnections are slow and unreliable
- 🔒 Additional constraints on privacy and efficiency



FedAvg = **Fed**erated **Av**era**g**ing

Main Challenges in FL

Statistical heterogeneity:

📊 **non-iidness** of local datasets hampers convergence of FedAvg[1]
⌛ **increased number of rounds** to reach a target accuracy [2]

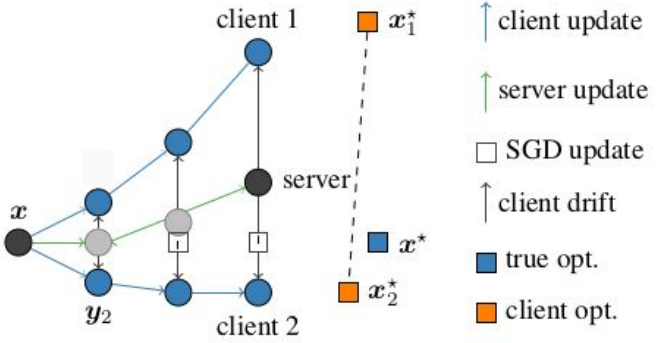


Figure 1: Client-drift in FedAvg illustrated for 2 clients with 3 local steps, from [5]

System heterogeneity:

📱 **edge devices** characterized by [3]:
- 🔋 being **transient**
- 🧮 **limited** computing **resources**
- 🚧 **slow**, **expensive**, and **unreliable communication links**

How to handle statistical heterogeneity?

Use proximal updates [FedProx]

💡 **penalize** the **divergence** from the **global model**

- 💥 **ineffective** in addressing heterogeneity

Use Stochastic Variance Reduction [SCAFFOLD]

💡 use **server** and **client control variates** to estimate the client drift, correct locally

- 💯 principled in theory, works in most cases
- ☹️ requires **doubling the communication**!

Use Alternating Direction Method of Multipliers [FedDyn]

💡 **regularize the (local) loss function** to converge to **stationary points** of the **global loss**

- 💯 same theoretical guarantees of SCAFFOLD
- 🎖️ communication-efficient
- 💣 **issues in dealing with pathological non-iid settings** [8]:
  - 🚫 in practical cases it can diverge 😱

Use Momentum [FedAvgM, Mime]:

💥 **Naive use in FL fails** at addressing extreme non-iidness

🫣 More sophisticated solutions [Mime] require **much more communication** (2x-3x)

Politecnico di Torino

Communication-Efficient Learning of Deep Networks from Decentralized Data

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Aguera y Arcas

Ivan Ludvig Tereshko
Riccardo Moroni

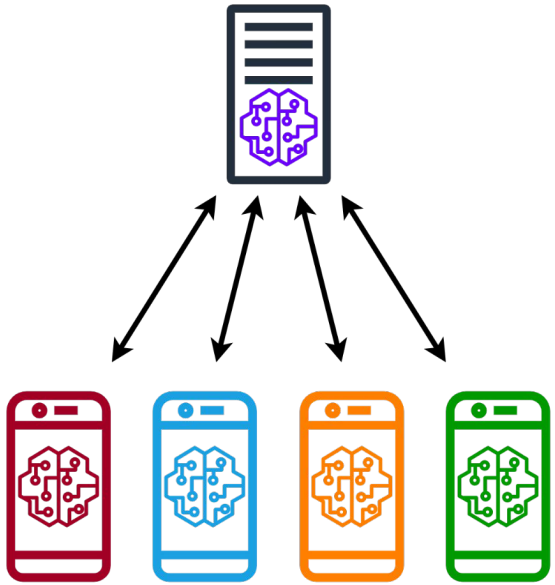
Introduction

Federated learning:

- Federation of clients lead by central server
- Clients update global model
- Data stays on clients privacy preserving

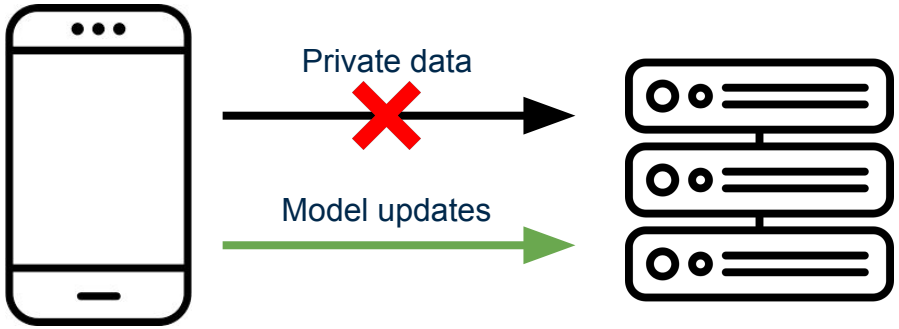
Use case examples:

- Next-word-prediction
- Image classification
- Voice recognition



Privacy

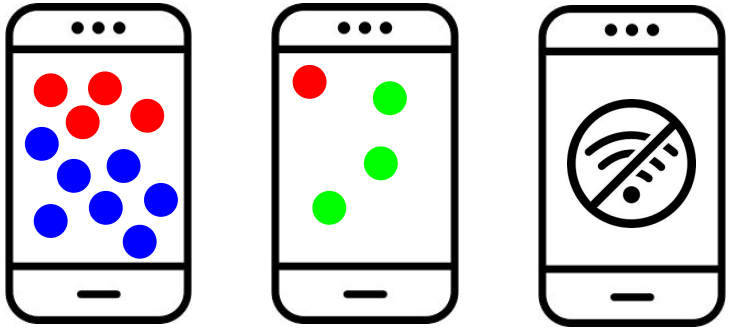
- Data stays on clients
- Updates to model contain less information
- Source of updates can remain unknown
- Stronger guarantees:
  - Differential privacy
  - Multiparty computation



Federated Optimization Properties

Data depends on usage by particular client:

- Non-IID distributed
- Unbalanced
- Massively distributed
- Limited communication with clients



Objective

- Data is partitioned among K clients
- $\mathcal{P}_k$ – dataset of k-th client with size $n_k = |\mathcal{P}_k|$
- Overall loss is a weighted average of client losses

Overall loss: $f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w)$

Client loss: $F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w)$

IID case: $\mathbb{E}_{\mathcal{P}_k}[F_k(w)] = f(w)$

FedAvg

- Synchronous updates in rounds
- Fraction of clients **C** is selected
- Global model is sent to selected clients
- Selected clients train for **E** epochs on batches of size **B**
- Updates are sent to server
- Server averages results

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**

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

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

  $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$

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

FedAvg

- Train on clients
- Average on server

Key parameters

C - fraction of clients per round

B - local minibatch size

E - local epochs

FedSGD: *C*=1, *B*=∞, *E*=1

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**

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

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

$w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$

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

Averaging Models

- Models w and w' are trained with SGD on different partitions of MNIST
- General non-convex objective
- Model averaging: $\theta w + (1 - \theta) w'$
- Different initialization leads to bad averaging model

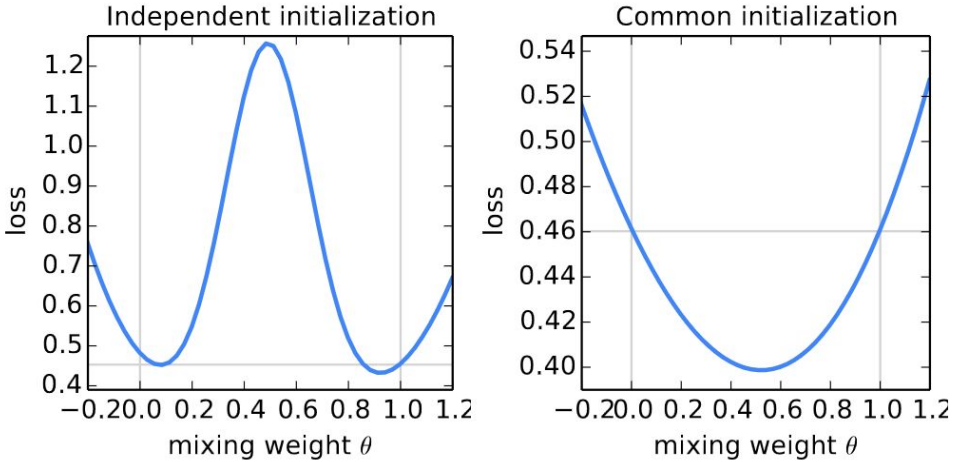


Figure 1: Loss on the full MNIST Dataset

Datasets

MNIST

IID Split

Non-IID Split *(Pathological)*
Each client sees pictures about just 2 digits.

CIFAR-10

IID Split

Image Classification Task

The Complete Works of William Shakespeare
(Pathological)

Each client is a speaking role in each play with at least two lines.

Language Modeling Task

Models

2NN : simple multilayer-Perceptron with 2-hidden layers with 200 units each using ReLu activations

CNN : convolutional neural network with two 5x5 convolution layers, a fully connected layer with 512 units and ReLu activation, and a final softmax output layer

Image Classification Task

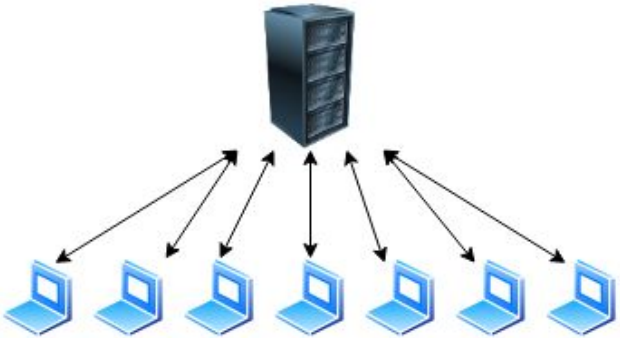
LSTM : stacked character-level LSTM language model which after reading each character in a line, predicts the next character

Language Modeling Task

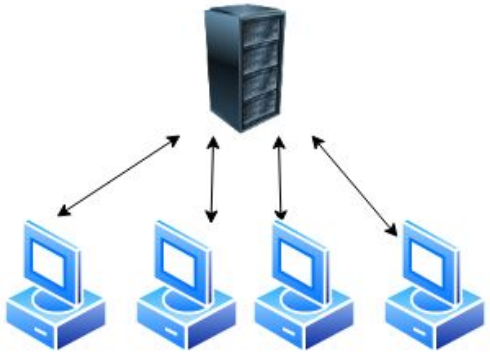
How to decrease the rounds of communication?

2 possible strategies:

1) Increasing Parallelism



2) Increasing Computation on each client



*server-client connections depicted in the images are intended **per round**

Results
1) Increasing Parallelism

2NN	IID		NON-IID	
C	$B = \infty$	$B = 10$	$B = \infty$	$B = 10$
0.0	1455	316	4278	3275
0.1	1474 (1.0×)	87 (3.6×)	1796 (2.4×)	664 (4.9×)
0.2	1658 (0.9×)	77 (4.1×)	1528 (2.8×)	619 (5.3×)
0.5	— (—)	75 (4.2×)	— (—)	443 (7.4×)
1.0	— (—)	70 (4.5×)	— (—)	380 (8.6×)
CNN, $E = 5$				
0.0	387	50	1181	956
0.1	339 (1.1×)	18 (2.8×)	1100 (1.1×)	206 (4.6×)
0.2	337 (1.1×)	18 (2.8×)	978 (1.2×)	200 (4.8×)
0.5	164 (2.4×)	18 (2.8×)	1067 (1.1×)	261 (3.7×)
1.0	246 (1.6×)	16 (3.1×)	— (—)	97 (9.9×)

Each table entry gives the **number of rounds** of communication necessary to achieve a test-set accuracy of 97%-99% along with the speedup relative to the *C=0* baseline.

Table 1: Increasing *C* achieves satisfactory accuracy in lower time, especially in Non-IID scenarios.

Results

2) Increasing Computation on each client

MNIST CNN, 99% ACCURACY					
CNN	E	B	u	IID	NON-IID
FEDSGD	1	∞	1	626	483
FEDAVG	5	∞	5	179 (3.5×)	1000 (0.5×)
FEDAVG	1	50	12	65 (9.6×)	600 (0.8×)
FEDAVG	20	∞	20	234 (2.7×)	672 (0.7×)
FEDAVG	1	10	60	34 (18.4×)	350 (1.4×)
FEDAVG	5	50	60	29 (21.6×)	334 (1.4×)
FEDAVG	20	50	240	32 (19.6×)	426 (1.1×)
FEDAVG	5	10	300	20 (31.3×)	229 (2.1×)
FEDAVG	20	10	1200	18 (34.8×)	173 (2.8×)

SHAKESPEARE LSTM, 54% ACCURACY					
LSTM	E	B	u	IID	NON-IID
FEDSGD	1	∞	1.0	2488	3906
FEDAVG	1	50	1.5	1635 (1.5×)	549 (7.1×)
FEDAVG	5	∞	5.0	613 (4.1×)	597 (6.5×)
FEDAVG	1	10	7.4	460 (5.4×)	164 (23.8×)
FEDAVG	5	50	7.4	401 (6.2×)	152 (25.7×)
FEDAVG	5	10	37.1	192 (13.0×)	41 (95.3×)

Table 2: Adding more local SGD updates per round can produce a **dramatic decrease in communication costs** for both IID and Non-IID scenarios.

Results

2) Increasing Computation on each client

ACC.	80%	82%	85%
SGD	18000 (—)	31000 (—)	99000 (—)
FEDSGD	3750 (4.8×)	6600 (4.7×)	N/A (—)
FEDAVG	280 (64.3×)	630 (49.2×)	2000 (49.5×)

Table 3: Experiments on the Cifar-10 dataset (which is more complex) confirm previous results. (*C = 0.1, E = 5, B = 50*)

Cifar-10 results were achieved using a different CNN architecture.

Results

2) Increasing Computation on each client

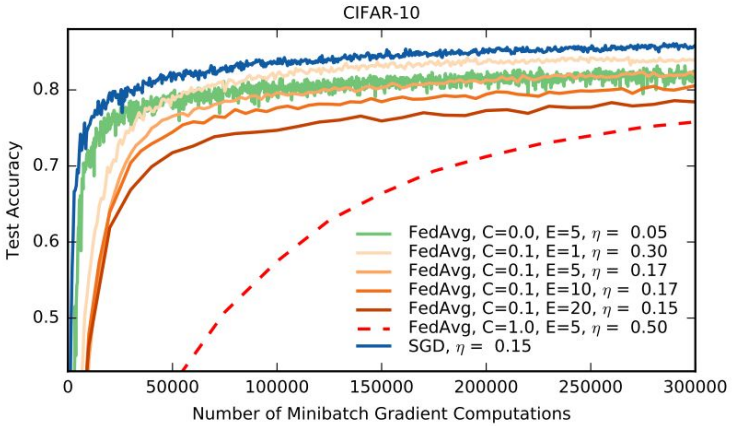
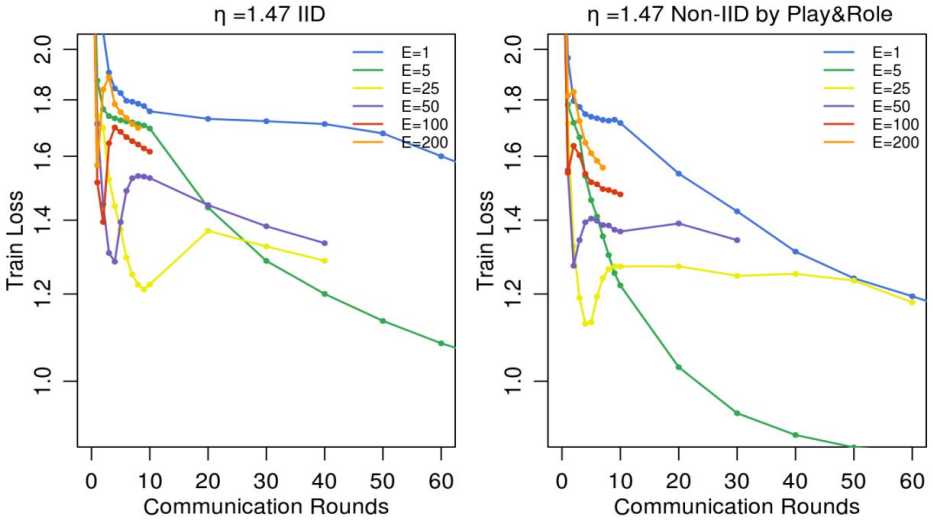


Figure 2: More SGD local updates lead to more stable models.

Moreover…



…but

Figure 3: Large *E* (local steps) for the Shakespeare LSTM (*B=10* and *C=0.1*)

Real-World Problem

Dataset (Non-IID):

- 10M public posts
- 500k clients

Model:
A large-scale language model word LSTM on a vocabulary of 10,000 words.

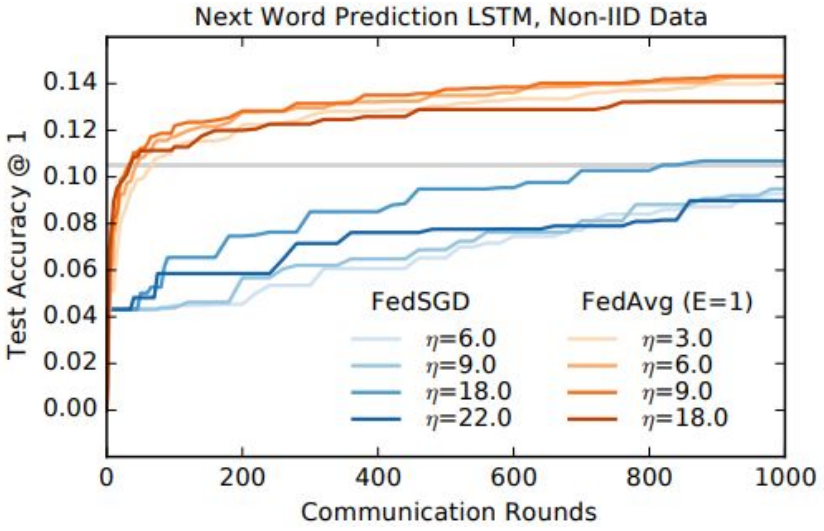


Figure 4

More Practical Issues

- Change of client datasets
- Correlation between client availability and data distribution
- Non-responding or corrupted clients

Beyond the scope

Politecnico di Torino

Thank you for your attention!

Politecnico di Torino

Adaptive Federated Optimization

Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konec˘ný, Sanjiv Kumar, H. Brendan McMahan

Abdirashid Chorshanbiyev, Ikromjon Nishonov, Mirkomil Egamberdiev

Federated Learning (FL)

A distributed machine learning approach where multiple clients collaboratively learn a model without sharing their raw training data.

Two main settings:

- **Cross-silo FL**: Large institutions (e.g., hospitals) collaborate.
- **Cross-device FL:** Edge devices (e.g., smartphones) collaborate.

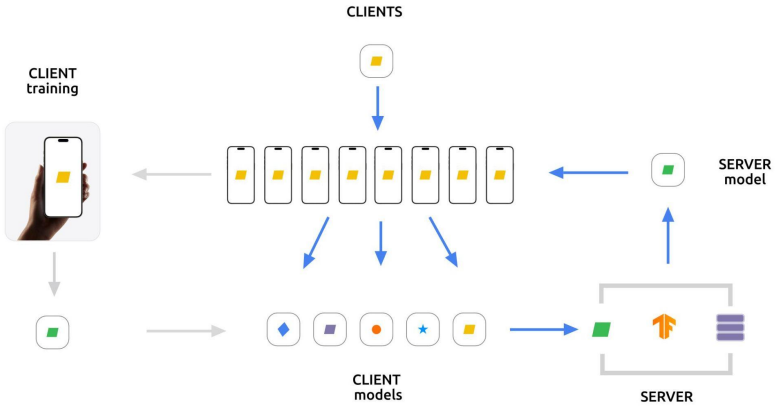


Figure 1

Federated Averaging (FedAvg)

A standard FL algorithm where clients perform local updates (multiple SGD epochs) on their data.
The server averages the client models to update the global model. Limitations:

- ***Convergence issues:*** Can struggle with heterogeneous data due to client drift (local models diverging from the optimal global model).
- ***Lack of adaptivity:*** Similar to SGD, it may not be ideal for scenarios with noisy gradients, common in language models.

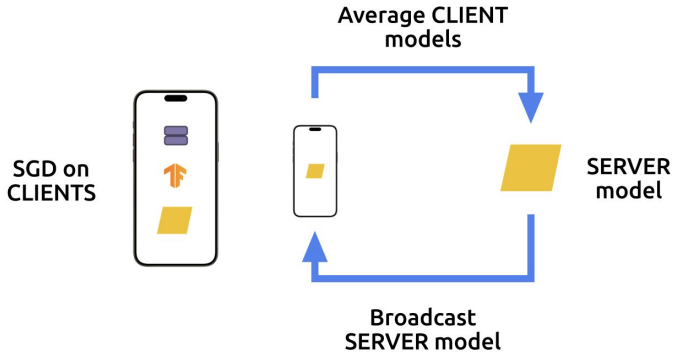


Figure 2

Adaptive Federated Optimization (FedOpt)

A general framework that addresses FedAvg's limitations.

Key components:

- ***Client optimizer*** (ClientOpt): Minimizes loss on local data (e.g., **SGD**).
- ***Server optimizer*** (ServerOpt): Optimizes the global model based on aggregated client updates (e.g., **Adagrad**, **Adam**, **Yogi**).

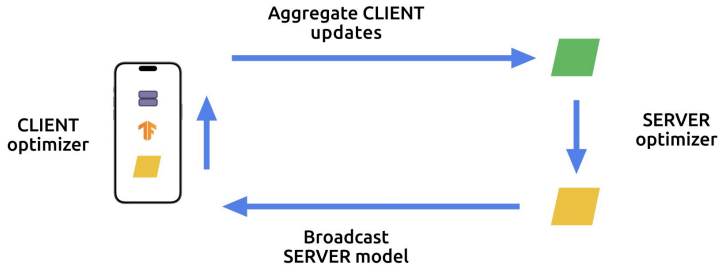


Figure 3

Federated Optimization Framework

Server optimization:

- Centralized aggregation
- Maintaining state
- Adaptive Optimization

Client optimization:

- Clients may participate at most once.
- Clients perform a small number of steps.
- The server can maintain state throughout the algorithm.

Algorithm 1 FEDOPT

1: Input: x_0, CLIENTOPT, SERVEROPT
2: **for** $t = 0, \dots, T-1$ **do**
3: Sample a subset of S of clients
4: $x_{i,0}^t = x_t$
5: **for** each client $i \in S$ **in parallel do**
6: **for** $k = 0, \dots, K-1$ **do**
7: Compute an unbiased estimate $g_{i,k}^t$ of $\nabla F_i(x_{i,k}^t)$
8: $x_{i,k+1}^t = \text{CLIENTOPT}(x_{i,k}^t, g_{i,k}^t, \eta_l, t)$ SGD
9: $\Delta_i^t = x_{i,K}^t - x_t$
10: $\Delta_t = \frac{1}{|S|}\sum_{i \in S} \Delta_i^t$
11: $x_{t+1} = \text{SERVEROPT}(x_t, -\Delta_t, \eta, t)$ Adaptive Optimizer

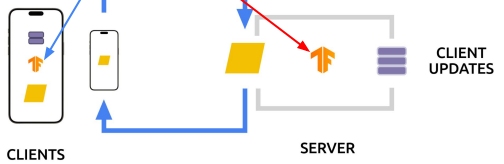


Figure 5

Differences and applications of FedOpt and FedAvg

Steps	FedOpt	FedAvg
Client Optimizer	Any Specified Optimizer	SGD
Server Optimizer	Any Specified Optimizer	SGD
Client Selection	Randomly sample subset S of clients	Randomly sample subset S of clients
Local Updates	Perform K local updates using CLIENTOPT	Perform multiple local epochs using SGD
Learning Rate Adaptation	Adaptive(depends on specified optimizer)	Fixed Learning Rate(SGD)
Main Advantage	Flexibility with different optimizers	Simplicity
Primary Use Case	General Federated Optimization	Standard Federated Optimization

Table 1: Comparison between FedOpt and FedAvg

Incorporating Adaptivity

Adaptive methods have had nearly unparalleled success in centralized optimization. We set out to bring the insights and power of these methods to the federated domain.
Our approach merges theory and practice of FL:

- Theoretically justified algorithms.
- Extensive empirical studies on a wide swath of models and tasks.
- Easy-to-use-open-source implementations.
- Reproducible benchmarks showcasing the methods.

Our goal is to enable more accurate and heterogeneity-aware FL models.

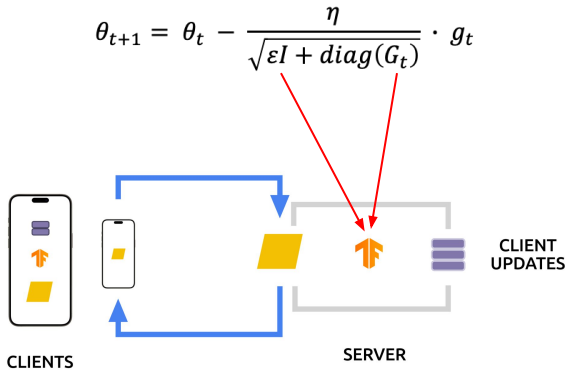


Figure 6

Adaptive Federated Optimization (FedOpt)

Strategy: Adaptive server optimization + client SGD

- Incorporates adaptivity without increasing client computation or communication!

Theoretical results:

- Convergence guarantees for non-convex losses, asymptotically optimal.
- Justify the use of pseudo-gradients.
- Interplay between client heterogeneity and communication efficiency

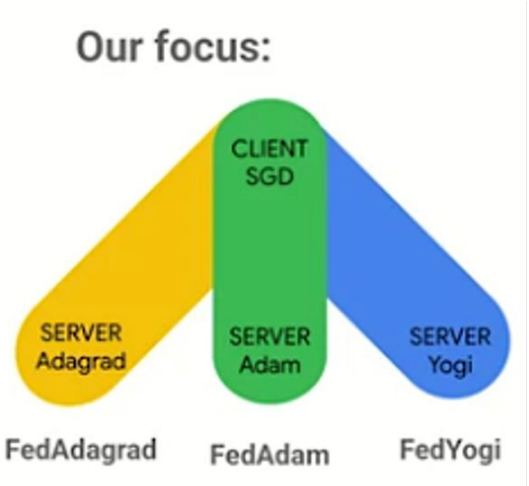


Figure 4

FedAdagrad, FedYogi, FedAdam

A specific instantiation of FEDOPT where the server uses the ADAGRAD optimizer, which adjusts learning rates based on historical gradients.

- the parameter $\boldsymbol{\tau}$ controls the algorithms' *degree of adaptivity* (e.g., with smaller values represents higher adaptivity)
- Handles heavy-tail distributions with adaptive rates

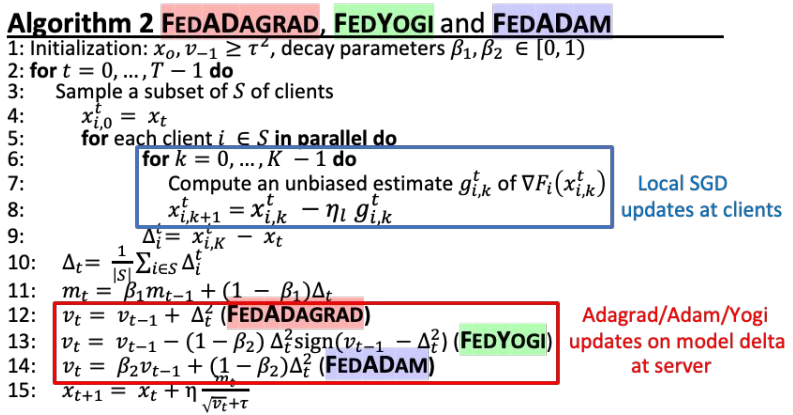


Figure 7

Datasets, Tasks, Methods

DATASET	MODEL	TASK SUMMARY	Dataset Reasons for Selection
CIFAR-10	ResNet-18	Image classification	CIFAR-10 is a widely-used benchmark for image classification. Data is not distributed identically across clients , to reflect heterogeneous client data distributions for testing federated learning methods
CIFAR-100	ResNet-18	Digit recognition	
EMNIST	Bottleneck network	Autoencoder	Extends MNIST to include handwritten characters, suitable for character recognition tasks, demonstrating adaptability.
Shakespeare	RNN with 2 LSTM layers	Next-character prediction	Simulates language modeling with each client representing a role in Shakespeare's works, providing substantial data variability.
Stack Overflow	RNN with 1 LSTM layers	Next-word prediction	Represents a real-world application with diverse text data from numerous clients, testing scalability and efficiency.
	Logistic Regression classifier	Tag prediction	

Table 2: Summary of Datasets, Models, and Task Details

Empirical results

- Adaptive methods generally outperform the non-adaptive baseline across all tasks, demonstrating their effectiveness in improving model performance
- By averaging the performance over the last 100 rounds, it is depicted that the results reflect stable and reliable performance rather than transient spikes

FEDERATED	ADAGRAD	ADAM	YOGI	AVGM	AVG
CIFAR-10	72.1	77.4	**78.0**	77.4	72.8
CIFAR-100	47.9	**52.5**	**52.4**	52.4	44.7
EMNIST CR	85.1	**85.6**	**85.5**	**85.2**	84.9
SHAKESPEARE	**57.5**	57	**57.2**	**57.3**	56.9
SO NWP	23.8	**25.2**	**25.2**	23.8	19.5
SO LR	**67.1**	65.8	65.9	36.9	30.0
EMNIST AE	4.20	1.01	**0.98**	1.65	6.47

Table 3: Average validation performance over the last 100 rounds: % accuracy for rows 1–5; Recall@5 (x100) for Stack Overflow LR; and MSE (x1000) for EMNIST AE. Performance within 0.5% of the best result for each task are shown in bold.

Empirical results

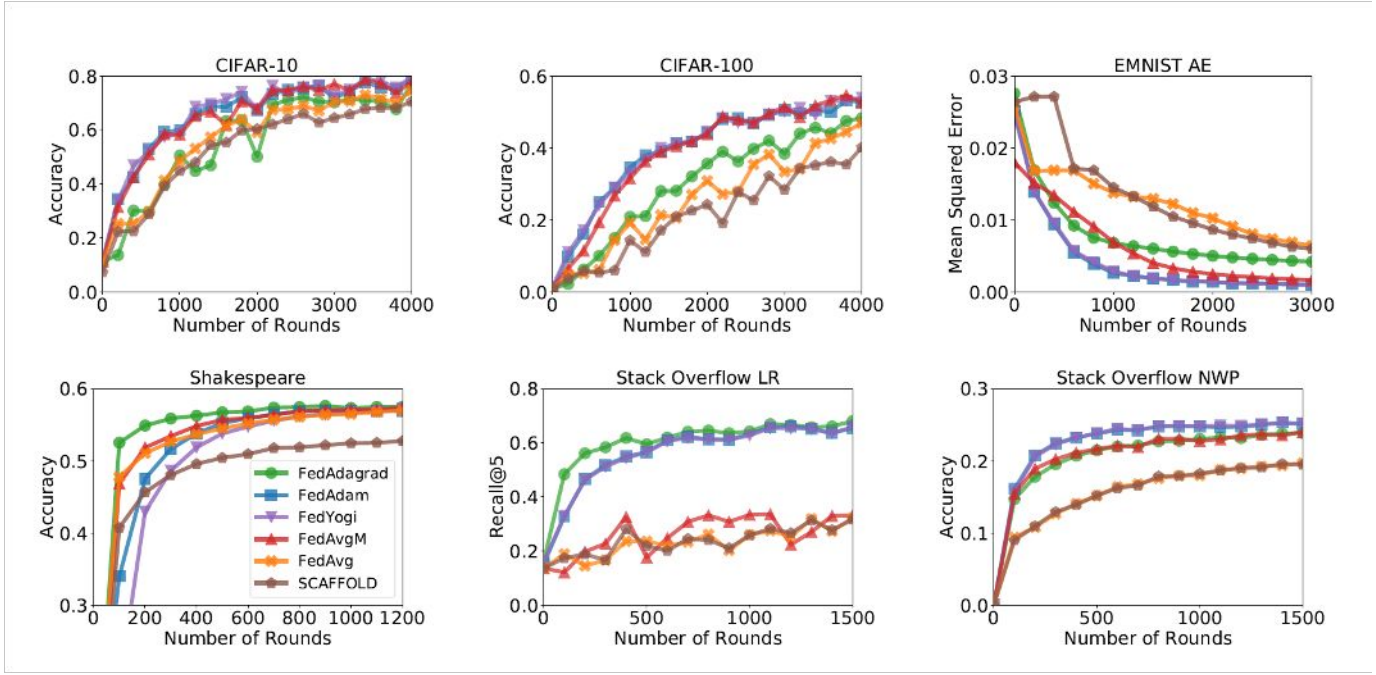


Figure 8: Validation accuracy of adaptive and non-adaptive methods, as well as SCAFFOLD, using constant learning

Future work

- So far , this paper focuses on adaptive server optimization for communication reasons. However, we can also **leverage client optimization**. Because Adaptive client optimization allows each client to learn a better model for their own data.

- Conduct extensive scalability studies to evaluate the performance and communication efficiency of the FEDOPT framework in **large-scale federated learning environments** with thousands or even millions of clients.

- Investigate the performance of **other adaptive optimizers** within the FEDOPT framework to determine if they offer additional benefits in federated learning scenarios.

Conclusion

- The FEDOPT framework provides significant **flexibility** by allowing the use of various optimizers on both the client and server sides. This adaptability makes it possible to tailor the federated learning process to specific needs and challenges, enhancing overall performance and convergence.

- Adaptive optimizers like ADAGRAD, used in Algorithm 2 (FEDADAGRAD), demonstrate **improved performance** in federated settings. They **adjust learning rates** based on historical gradient information, which can handle diverse data distributions and client heterogeneity more effectively than standard SGD.

- Empirical results show that using the FEDOPT framework, particularly with adaptive optimizers on the server side, leads to **better convergence rates** and **model accuracy** compared to traditional federated learning methods like FEDAVG. This improvement is especially noticeable in scenarios with non-i.i.d. (non-independent and identically distributed) data across clients.

Politecnico di Torino

Thank you for your attention!

Politecnico di Torino

PerFedMask: Personalized Federated Learning with Optimized Masking Vectors

Mehdi Setayesh, Xiaoxiao Li, and Vincent W.S. Wong

Fatemeh Ahmadpour s289594
Mohammadmahdi Gheisari s289595
Ali Abbasi s290007

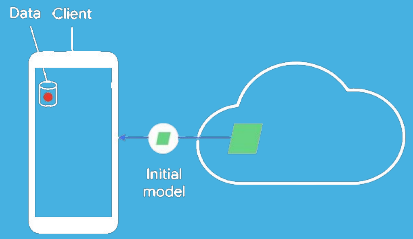
Introduction

Federated Learning

01

Model Transfer

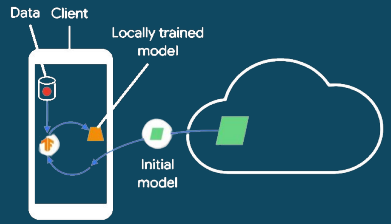
Edge devices download the latest model from the server to be used as their local model



02

Local Updates

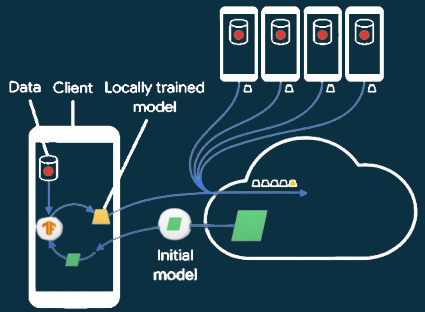
Each device performs multiple local update iterations for updating the local model based on its local dataset



03

Model Upload

Devices upload their updated local models to the server



04

Server Aggregation

The server computes the new model by aggregating the local models

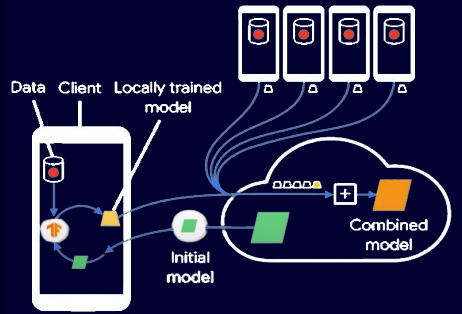


Fig.1: Federated learning steps

Problems

Data Heterogeneity

Local dataset

May be different in size

May contain Non-IID data samples

Device Heterogeneity

Diverse and limited capabilities in

Computation

Communication

storage

Personalized FL

Modifies traditional FL to address heterogeneity problems through:

Model Separation: global and local head model

Collaborative Training

Fine-Tuning

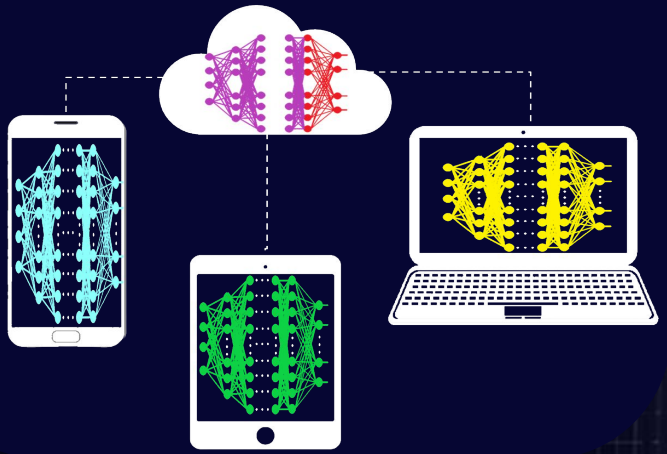


Fig.2: Personalized FL

Masking

Masking vectors are used in FL to selectively enable or disable updates to certain parts of the model during local training.

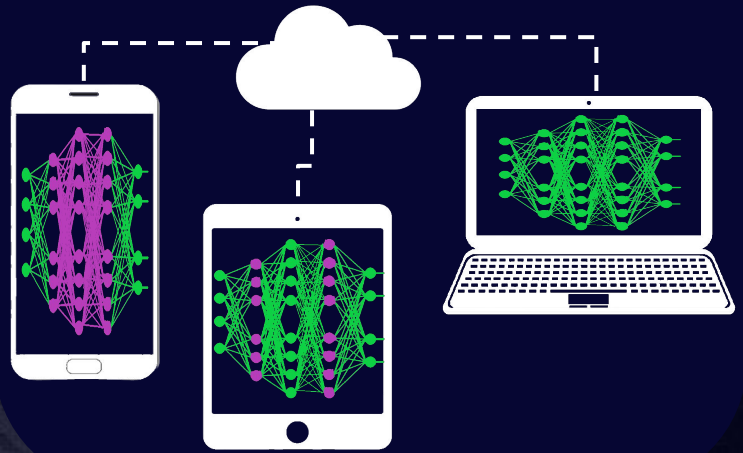


Fig.3: Masking vectors

Masking with pruning and freezing

Pruning

Using masking vectors to keep only important parameters and remove unimportant ones

Cons:

- Additional computational overhead
- Different model architectures
- Accuracy reduction

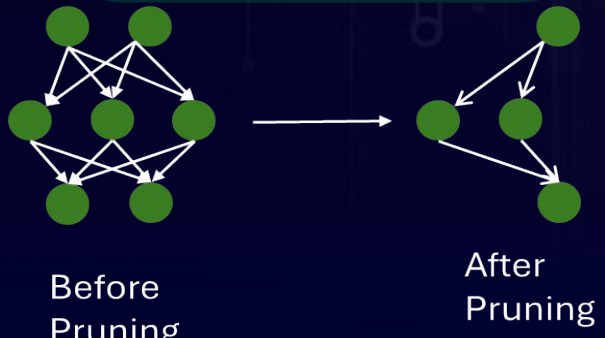


Fig.4: Pruning

Freezing

Using masking vectors to freeze some parts of the learning model for each device

Pros:

- More stable FL without changing model architecture
- Reduce computational and communicational resources

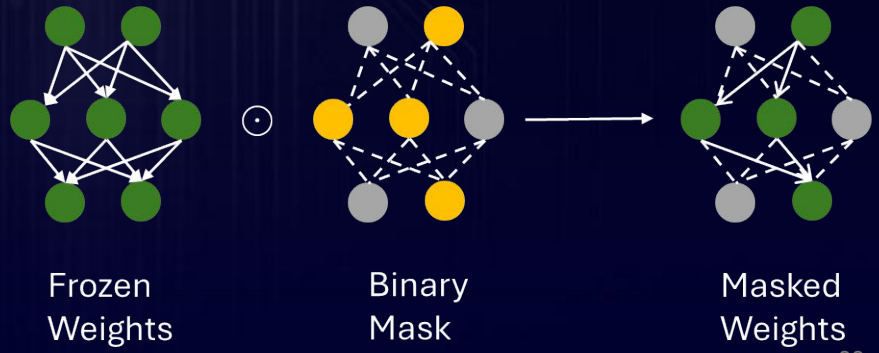


Fig.5: Freezing

Bias

- Masking introduces bias in model training
- Some model parts get updated more frequently than others
- This imbalance leads to a less accurate and fair model

Layer-wise Masking

- Freezes entire model layers based on device capability
- Balances training and reducing bias
- Fine-tuning after training improves accuracy

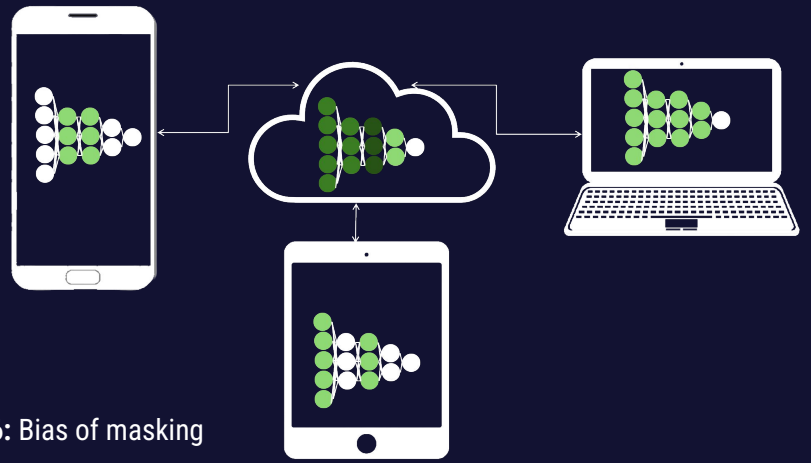
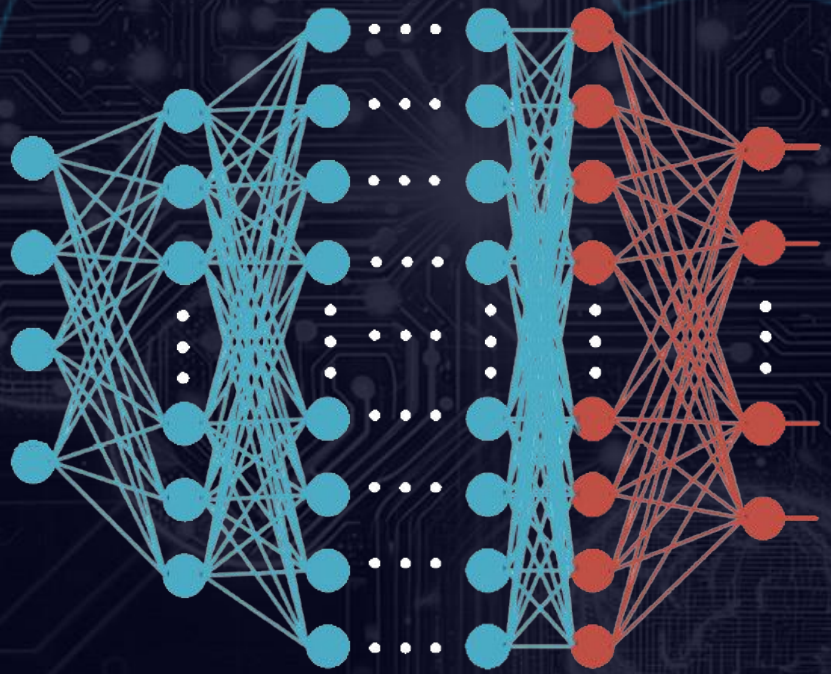


Fig.6: Bias of masking

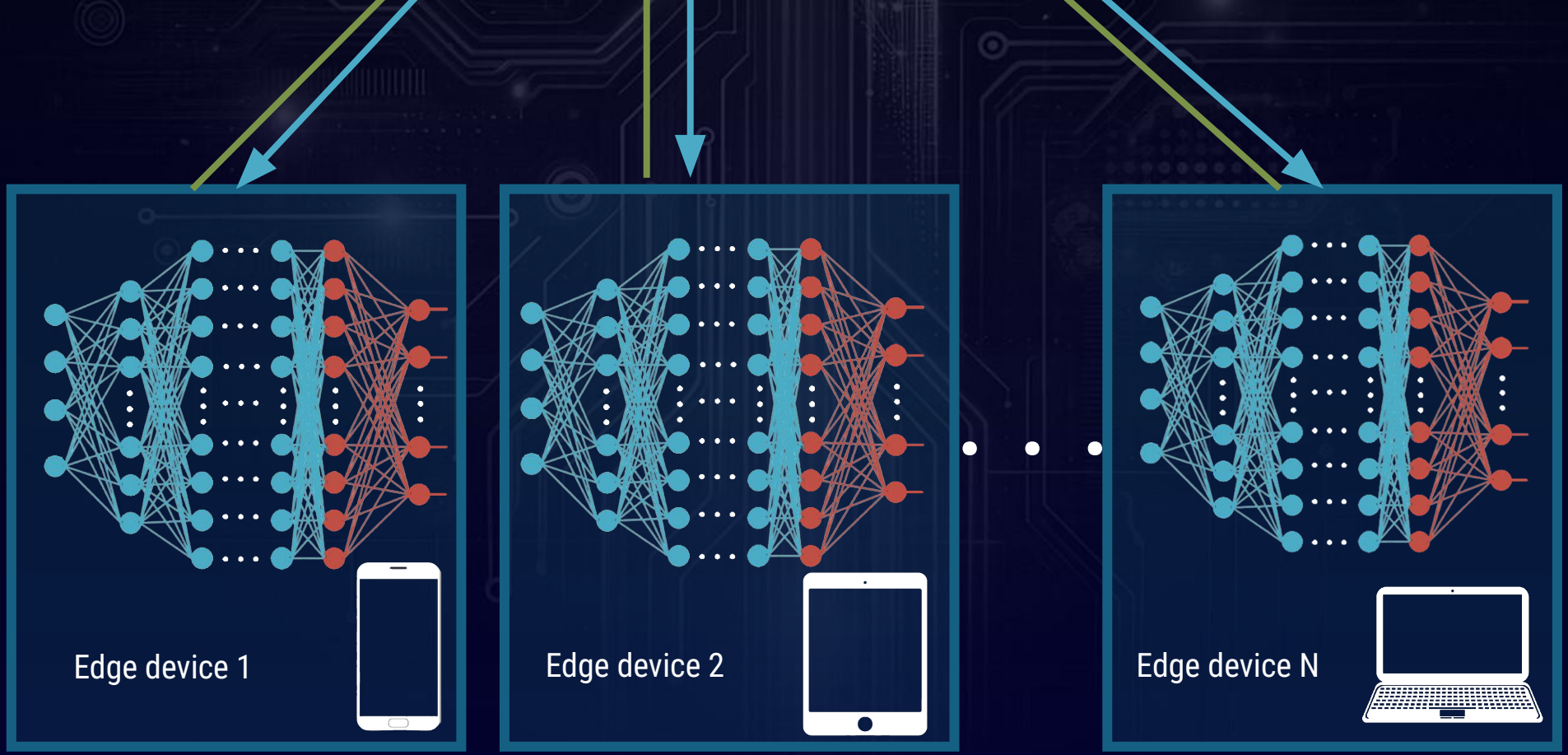
PerFedMask

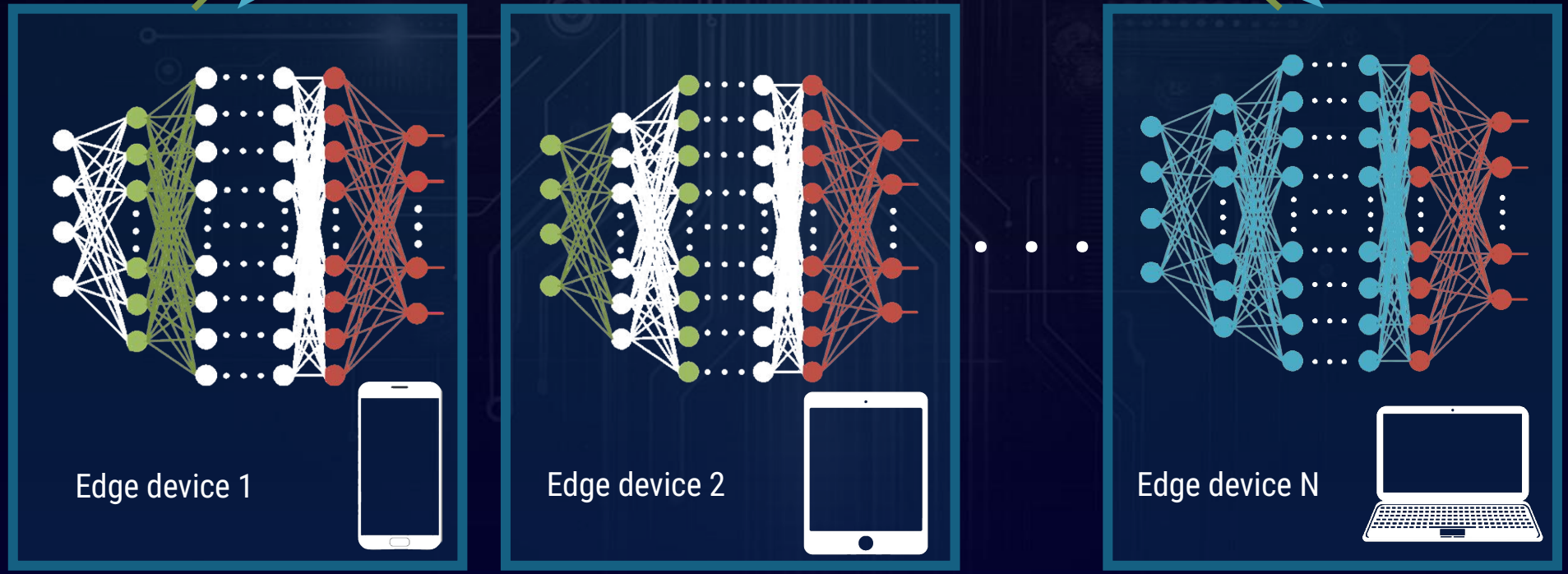
Method

Server



- Global model
- Local head model


Edge device 1
Edge device 2
Edge device N


Edge device 1
Edge device 2
Edge device N

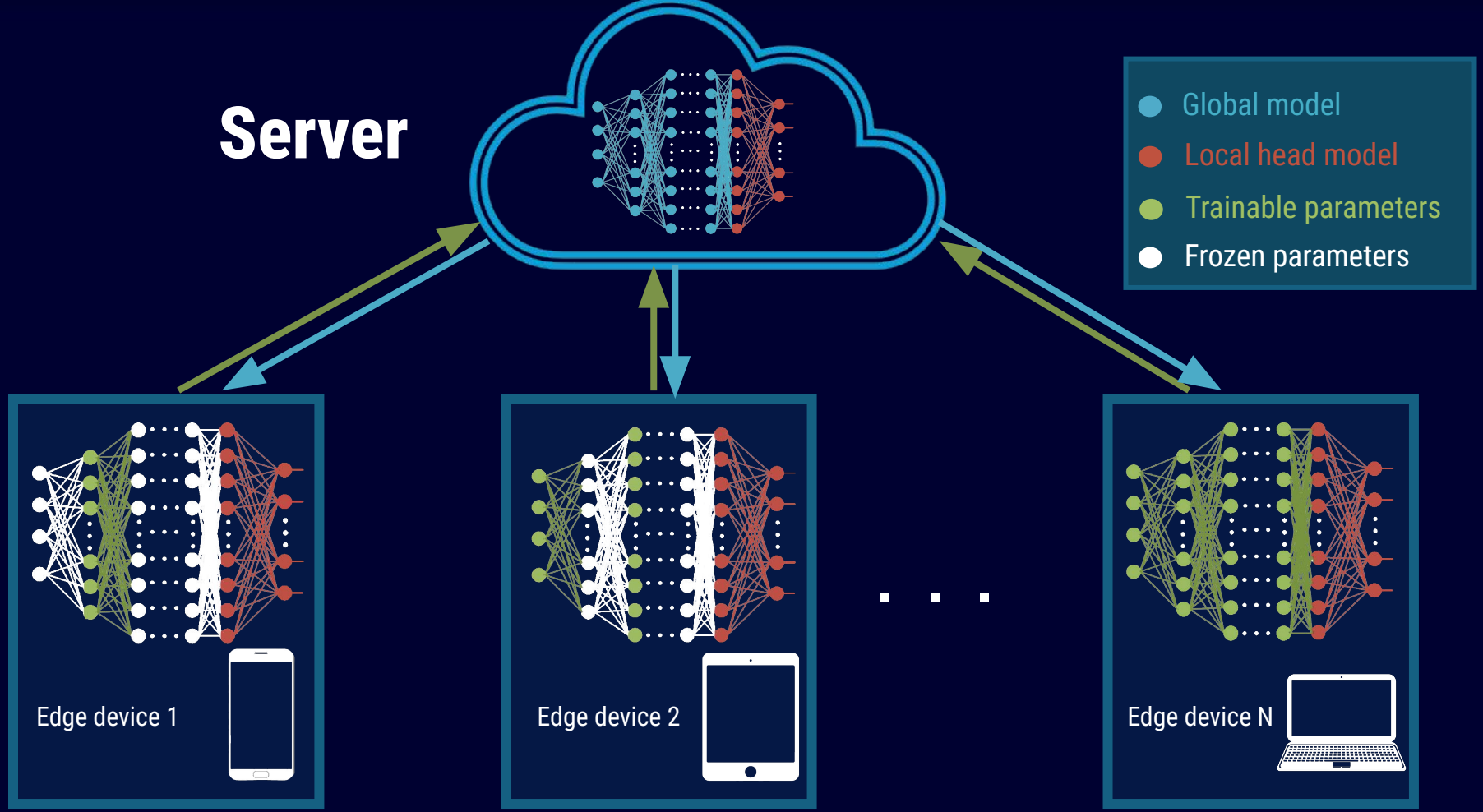


Fig.7: Illustration of an FL system using PerFedMask

Advantages of PerFedMask

Generalization

Extends FedBABU for varying computational capabilities.

Flexibility

Compatible with other FL algorithms like HeteroFL and Split-Mix FL.

Consistency

Addresses objective inconsistency without modifying device optimizers.

Datasets and Model Architectures

CIFAR-10

ResNet model
100 devices:
- 450 training samples
- 50 validation samples
- 100 test samples

50 batch size
Maximum 320 communication round
5 local iteration for fine-tuning
3 classes per device

CIFAR-100

MobileNet model
100 devices:
- 450 training samples
- 50 validation samples
- 100 test samples

50 batch size
Maximum 320 communication round
5 local iteration for fine-tuning
10 classes per device

Metrics

- Test accuracy before and after fine-tuning
- Average number of FLOPs
- Average trainable parameters

Performance Comparison

Test Accuracy After Fine-tuning

Dataset	C	PerFedMask	FedBABU	FedProx	FedNova	HeteroFL	Split-Mix FL	FedAvg
CIFAR-10	1	88.43	88.20	84.96	84.26	87.33	85.56	84.99
CIFAR-10	0.1	83.60	84.27	74.55	71.88	73.34	77.76	71.19
CIFAR-100	1	72.40	69.01	64.63	65.24	68.65	65.95	65.27
CIFAR-100	0.1	67.47	66.32	59.36	60.42	65.87	62.35	59.12

Table 1: Comparison of the test accuracy after fine-tuning with 6 state of the art in FL

Number of Trainable Parameters

Dataset	PerFedMask	FedBABU	FedProx	FedNova	HeteroFL	Split-Mix FL	FedAvg
CIFAR-10	6.13M	11.167M	11.172M	11.172M	5.674M	0.793M	11.17M
CIFAR-100	1.803M	3.207M	3.209M	3.309M	1.774M	0.223M	3.309M

Table 2: Comparison the number of trainable parameters with 6 state of the art in FL

Combination of PerFedMask with HETERO FL and SPLIT MIX FL

Algorithm	Test Accuracy Before Fine-tuning	Test Accuracy After Fine-tuning	# of Trainable Parameters	# of FLOPs Forward	# of FLOPs Backward
PerFedMask + Split-Mix FL	51.88	87.74	0.691M	0.178G	0.514G
PerFedMask + HeteroFL	69.44	87.79	5.473M	1.111G	1.721G
PerFedMask	70.14	88.43	6.138M	2.182G	2.697G
Split-Mix FL	57.96	85.56	0.793M	0.178G	0.541G
HeteroFL	62.58	87.33	5.674M	1.111G	1.749G
FedBABU	69.27	88.20	11.167M	2.182G	3.466G

Table 3: Comparison of performance, number of trainable parameters, and number of FLOPs, when PerFedMask combined with HETERO FL and SPLIT MIX FL

Effect of Fine-tuning Steps

		Fine-tuning Steps					
Dataset	Algorithm	0	2	4	6	8	10
CIFAR-10	PerFedMask	70.14	88.78	88.69	88.35	88.25	88.30
	FedBABU	69.27	88.63	88.38	88.43	88.12	87.92
CIFAR-100	PerFedMask	32.04	72.62	72.84	72.48	72.11	72.40
	FedBABU	29.70	69.03	69.15	68.86	68.83	68.81

Table 4: Effect of fine-tuning step in PerFedMask and FedBaBU

Effect of increasing number of devices with maximum computational capabilities

Algorithm	v	Test Accuracy		# of Trainable Parameters	# of Backward FLOPs
		Before Fine-tuning	After Fine-tuning		
PerFedMask	0.2	29.29	72.07	0.941M	0.617G
PerFedMask	0.4	32.31	74.33	1.518M	0.675G
PerFedMask	0.6	32.79	72.82	2.095M	0.741G
PerFedMask	0.8	33.59	72.64	2.647M	0.803G
PerFedMask	1.0	34.73	73.76	3.207M	0.863G

Table 5: Effect of increasing number of devices with maximum computational capabilities in PerFedMask

Effect of masking vector design

Design Approach	Bias	Training Loss	Training Accuracy	Test Accuracy Before Fine-tuning
Sequential Masking	3.36M	2.324	50.11%	27.02%
Random Masking	0.609M	1.706	64.40%	28.88%
Optimized Masking	0.204M	1.648	66.63%	29.84%

Table 6: Effect of masking vector design

Conclusion

Overview

- Flexible and easy-to-implement personalized FL algorithm

Key Techniques

- Optimized masking vectors
- Fine-tuning

Benefits

- Higher accuracy
- Fewer trainable parameters
- Lower computational costs

Future Directions

- Explore freezing priorities
- Apply masking vector designs in pruning methods.
- ZooPFL algorithm

Thanks for your attention