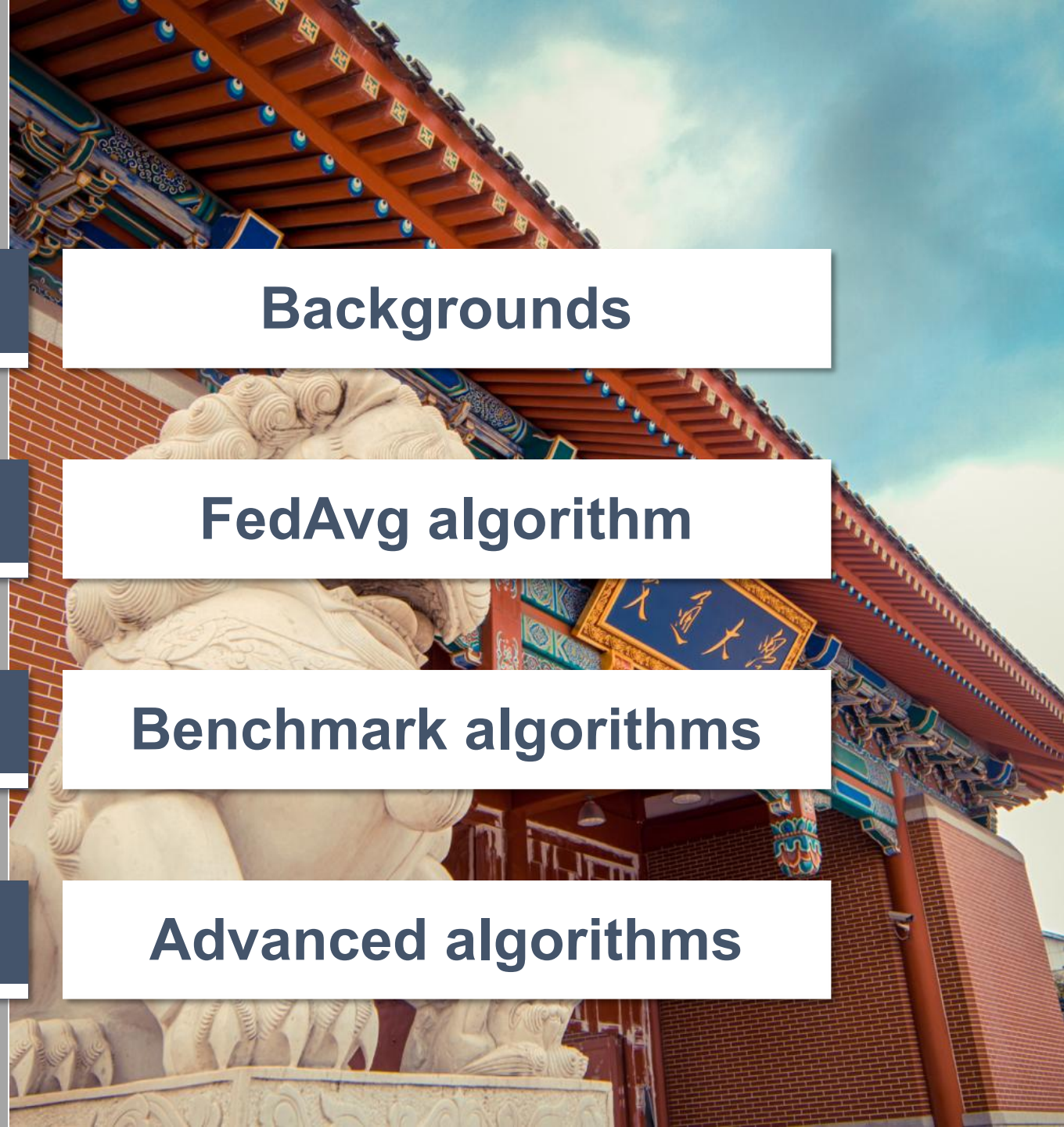
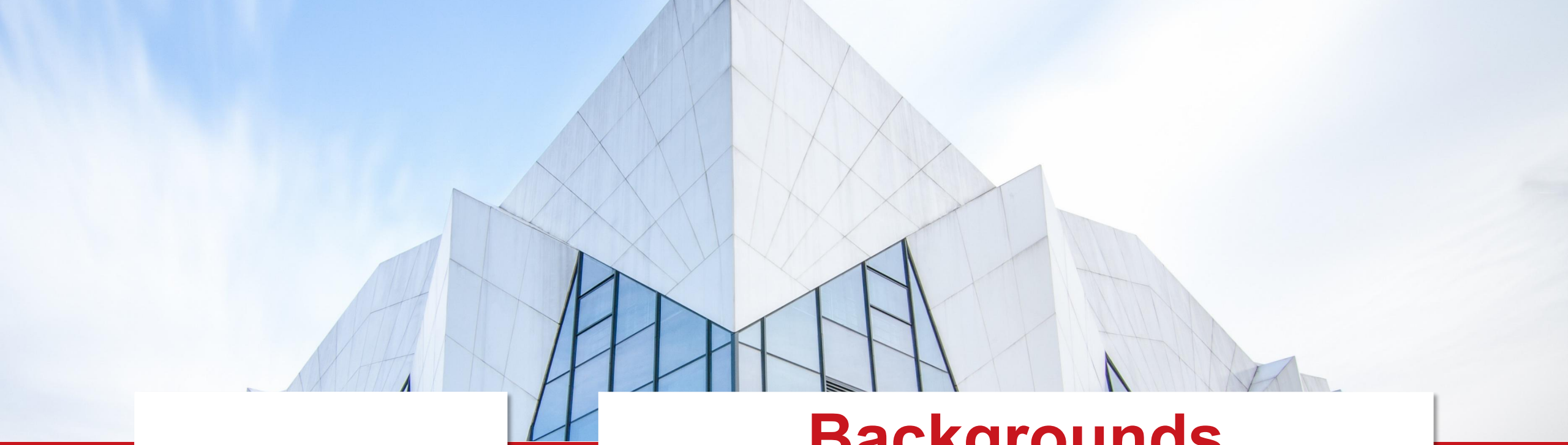
Federated Learning (FL) Lab

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录





00

Backgrounds

- Governmental support
- Develop trend

隐私计算及联邦学习

- 数据最高管理部门——国家数据局成立
- 负责协调推进数据基础制度建设，统筹数据资源整合共享和开发利用，统筹推进数字中国、数字经济、数字社会规划和建设等，由国家发展和改革委员会管理。
- 国家数据局的成立或将促进以下行业的发展：
  - 数据收集（各行业数字化转型）
  - 数据储存(国资云，底层算力基础设施)
  - 数据处理（数据安全与隐私计算）
  - 数据定价（数据资产化）
  - 数据流转(数据交易机构、公共数据交易)

隐私计算及联邦学习

	安全多方计算	联邦学习	同态加密	可信执行环境	合成数据	差分隐私	秘密共享
波士顿女性劳动力	■						
欧洲统计系统	■	■	■				
欧盟统计局				■			
印度尼西亚旅游部				■			
意大利国家统计研究所	■						
英国国家统计局					■	■	
三星SDS（韩国）	■						
加拿大统计局	■		■		■		
韩国统计局	■		■			■	
荷兰统计局	■	■	■				■
联合国欧洲经济委员会		■	■			■	
美国人口普查局						■	
美国教育部	■						

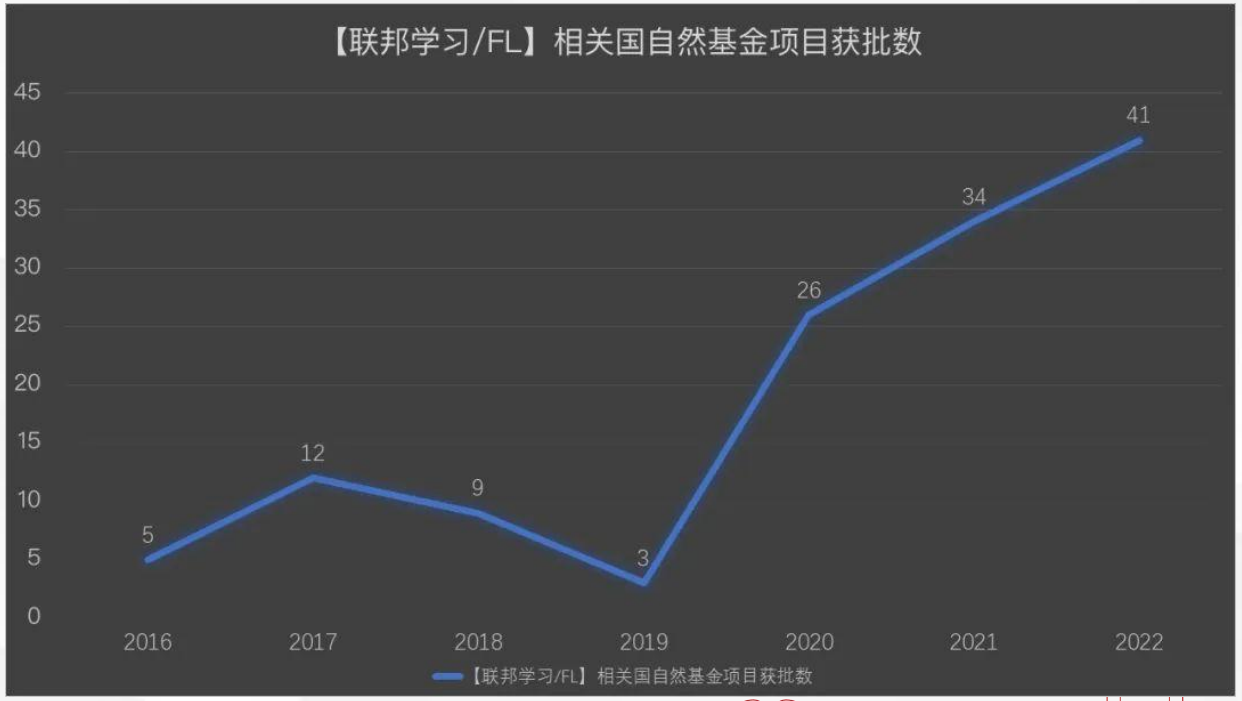
隐私计算及联邦学习

- 联邦学习相关国自然基金项目获批数呈现出逐年增加的趋势，这表明联邦学习在学术界和工业界的应用越来越受到重视。联邦学习是多个领域的融合，触及到的研究方向众多，应用场景非常丰富。

- 金融领域有复旦大学柴洪峰团队获批的重大研究计划《大数据背景下基于联邦学习的小微企业信用风险评估研究》

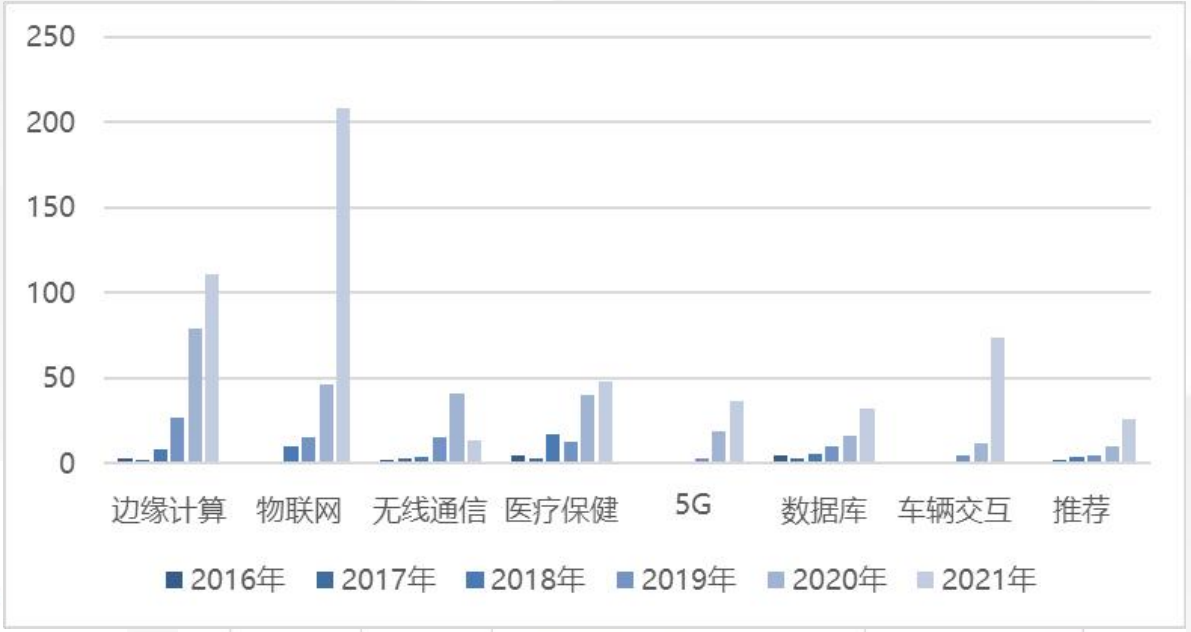
- 通信领域有深圳市大数据研究院朱光旭团队获批的青年科学基金项目《面向联邦式边缘学习的高效通信技术研究》

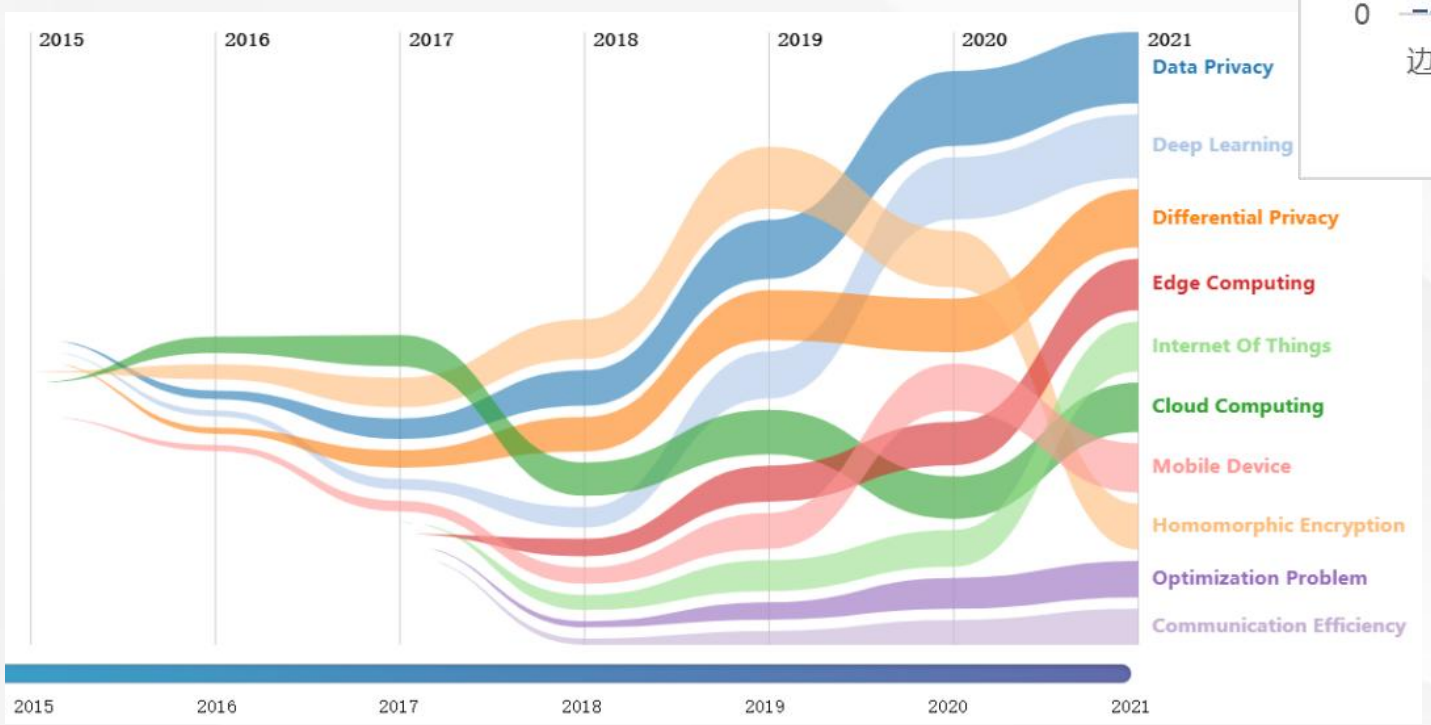
- 医疗领域有上海大学武星团队的面上项目《多中心胶囊内窥镜影像联邦主动学习》

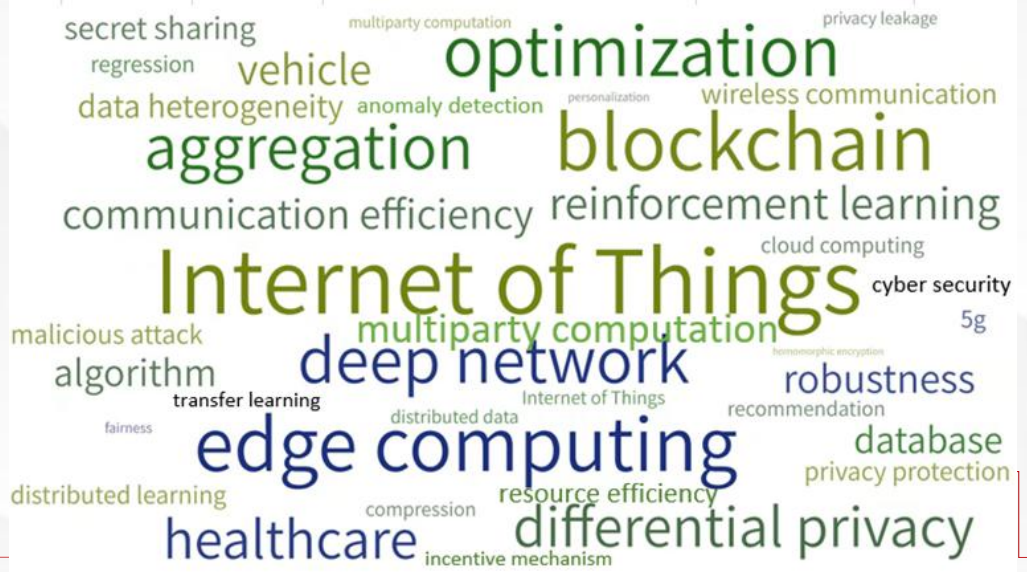


隐私计算及联邦学习

- 涵盖多个领域，主要以边缘计算、物联网和车辆交互为主
- 各领域联邦学习研究逐年增长，趋势良好







隐私计算及联邦学习

- 联邦学习性能上以聚合、优化方法和异构为主
- 各方法研究量逐年增长

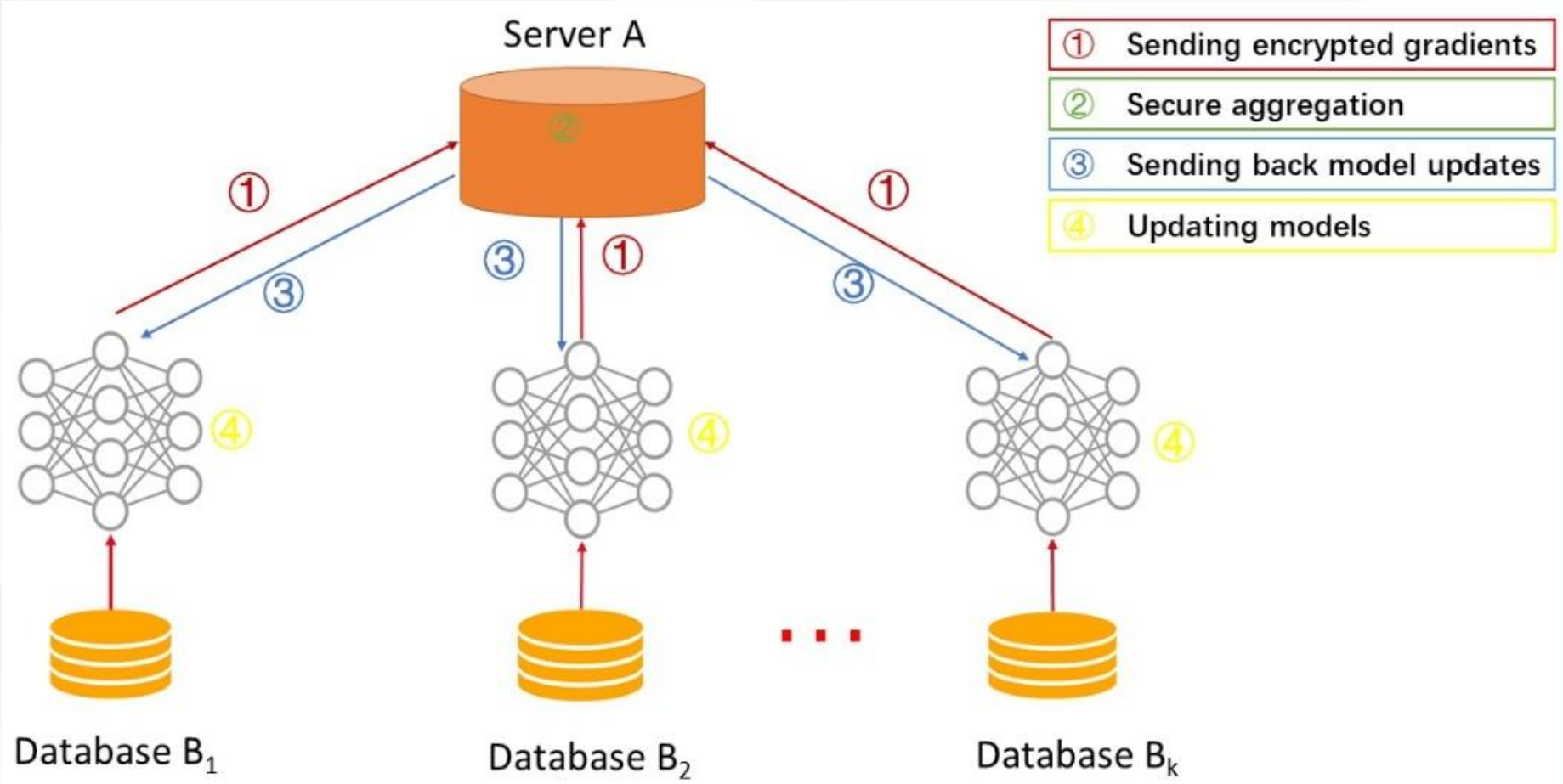
- 联邦学习安全上以区块链、差分隐私和多方计算为主
- 各方法研究量逐年增长

开源框架

Github热度	发布方	系统名称	开源时间	系统特点
8000	OpenMined	PySyft	2017.7	• 一个用于安全和私有深度学习的 Python 库 • 基于 PyTorch，使用 Unity Game Engine • 安全多方计算 • 联合学习、差异隐私
4100	微众银行	FATE	2019.2	• 工业级框架，采用 Python开发，底层计算存储基于 EGGROLL、Spark 等高性能计算引擎 • 提供一站式的联邦模型企业级服务解决方案。提供多插件支持联邦学习企业和科研应用 • 支持主流的分类、回归、聚类和迁移学习的联邦化算法 • 提供多种安全计算协议支撑上层应用，支持同态加密协议、秘密共享协议、不经意传输协议和 DH密钥交换算法等 • 提供20多个联邦算法组件
1800	谷歌	TensorFlow Federated	2019.3	• 可以选择 ML 模型架构 • 模型设计理念以数据为主

联邦学习介绍

- 联邦学习是一种分布式机器学习技术，或机器学习框架。
- 在保证**数据隐私安全**及**合法合规**的基础上，实现**共同建模**，提升AI模型的效果。
- 本质上是通过多个用户设备共同训练一个代表所有用户设备的全局模型
- 训练过程不需要用户数据的交换，更强调隐私性。



01

FedAvg algorithm

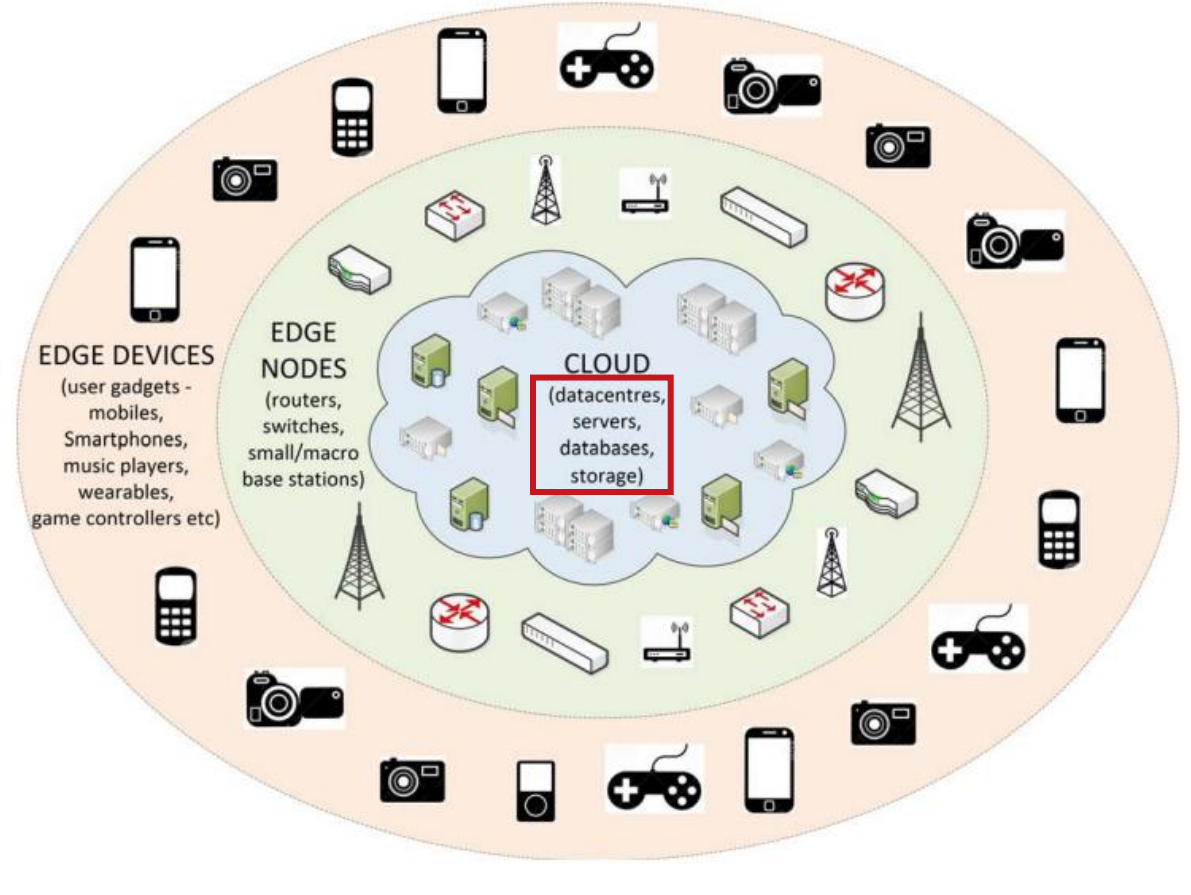
- Background and contributions
- Federated learning review
- Federated settings
- Federated Averaging
- Experiments

FedAvg algorithm

- Background
  - Recently, attributed to the collection of **massive data** from users or organizations, AI has been thriving for years.
  - However, some **private data** is also collected during data collection, such as the shopping behaviors, facial images, house locations, etc.
  - Data breach is becoming more and more severe, and many governments have issued **data privacy protecting** laws.

FedAvg algorithm

- Background
  - AI model training based on distributed computing and edge computing is required.



FedAvg algorithm

- Background
  - Different from traditional settings, data cannot be collected and naturally exists locally.

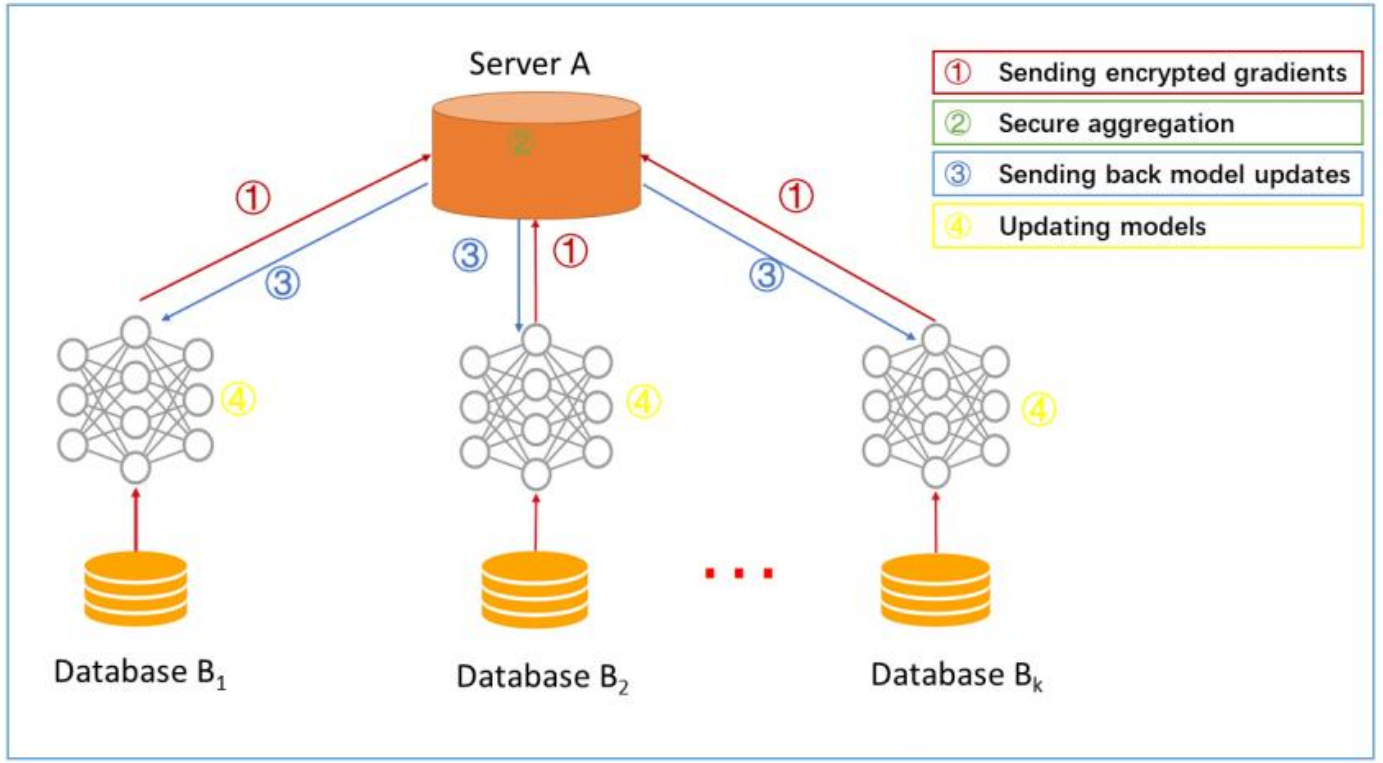


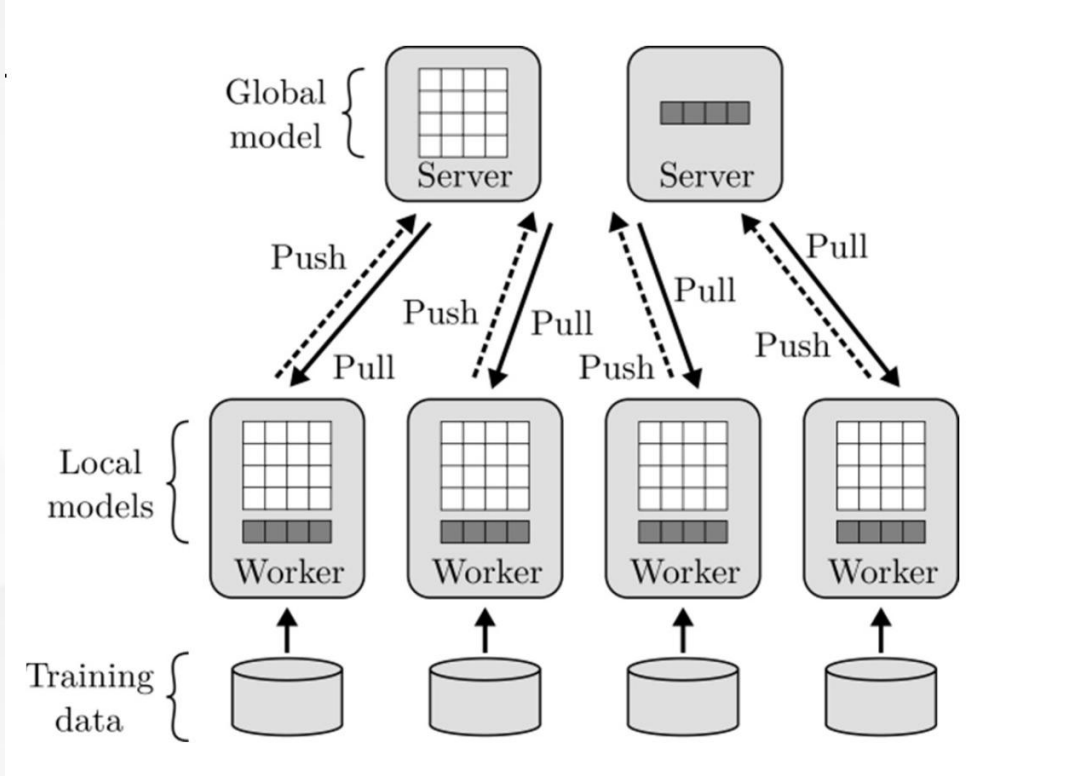
Fig. 3. Architecture for a horizontal federated learning system

FedAvg algorithm

- Contributions
  - Consider the problem of training dispersed data from mobile devices as an important research direction.
  - Propose a simple and practical algorithm for Federated Averaging.
  - An extensive empirical evaluation of the proposed algorithms shows that they are robust to non-independently identically distributed (Non-IID) and unbalanced data.

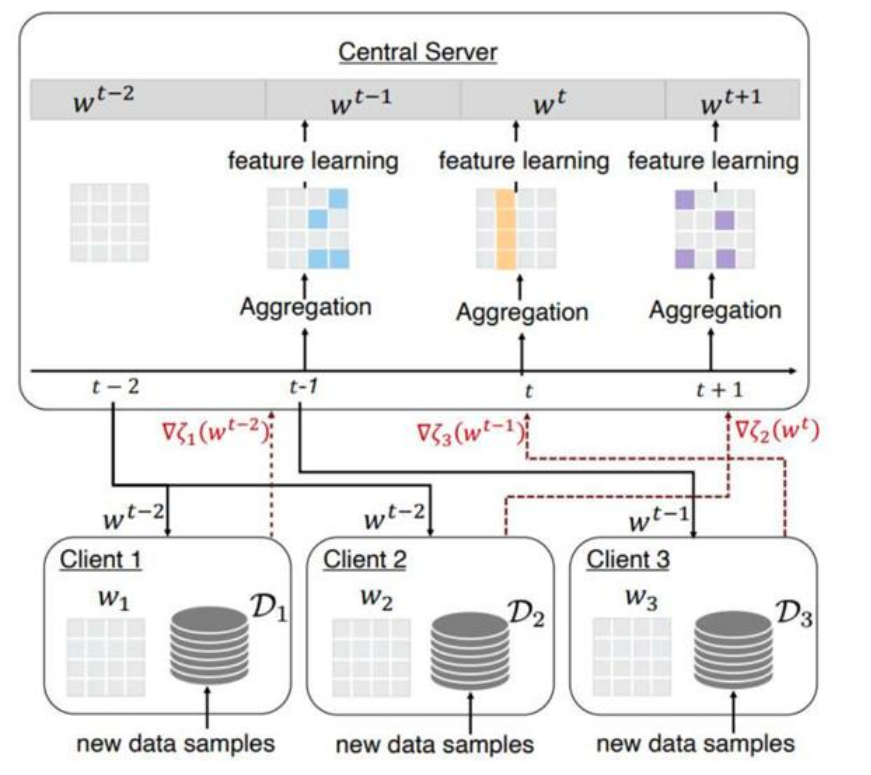
FedAvg algorithm

- Federated learning review
  - Learning tasks are handled by a loose federation of participating devices (clients) coordinated by a central server.



FedAvg algorithm

- Federated learning review
  - Each client has a local training data set that does not need to be uploaded to the server and only sends local model parameters for each update.



FedAvg algorithm

- Federated learning (FL) review: ideal FL
  - Training on **real-world data** from mobile devices has distinct advantages over the proxy data that is ubiquitous in data centers.
  - This data is **privacy-sensitive** or large (compared to the size of the model) to avoid recording it to the data center for model training.
  - For supervised tasks, labels on data can be **naturally inferred** from user interactions.

FedAvg algorithm

- Federated learning (FL) review: privacy
  - In the traditional distributed training setting, even if an "anonymous" data set is held, users' privacy will be threatened through the **connection with other data**.
  - In contrast, the information transmitted in FL is the **minimum update** (all/part of the model parameters) needed to improve a particular model, and less information means a lower risk of privacy disclosure.
  - Combining FL with **secure multi-party computing and differential privacy**.

FedAvg algorithm

- Federated settings
  - Non-IID data: Training data on a given client is usually based on mobile device usage by a particular user, so any local data set for a particular user **does not represent a group distribution**.
  - Imbalance: Some users will use the service or application more than others, resulting in **different amounts of local training data**.
  - Massive clients: We expect the number of clients participating in the FL to be **much larger than** the average number of instances per client.
  - Limited communication: Mobile devices are often **offline or in a slow or expensive connection**.

FedAvg algorithm

- Objective

$$\min_{w \in \mathbb{R}^d} f(w) \qquad \text{where} \qquad f(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w). \qquad (1)$$

$$f(w) = \sum_{k=1}^{K} \frac{n_k}{n} F_k(w) \quad \text{where} \quad F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w).$$

$$f_i(w) = l(x_i, y_i; w)$$

K number of clients

P_k the distribution of local data

n_k the number of local data samples

FedAvg algorithm

- Limited communication
  - In data center optimization, communication costs are relatively small while computing costs dominate, and recent emphasis has been on using GPUs to reduce these costs. In contrast, in joint optimization, **communication costs dominate** -- we're typically limited by upload bandwidth of 1MB/s or less.
  - Clients typically volunteer for optimization only when charging, plugging in, and using a non-billable Wi-Fi connection, and we expect each client to **participate in a small number of update sessions** per day.
  - Modern smartphones have **plenty of local computing power and small datasets** on a single device.

FedAvg algorithm

- Reduce communication
  - Increasing **parallelism**, that is, using more clients to work independently between rounds of communication.
  - Add computations per client, that is, **perform more complex computations** (such as cumulative training) between rounds of communication.

FedAvg algorithm

Federated Averaging

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**

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

$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

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

K： Client amount

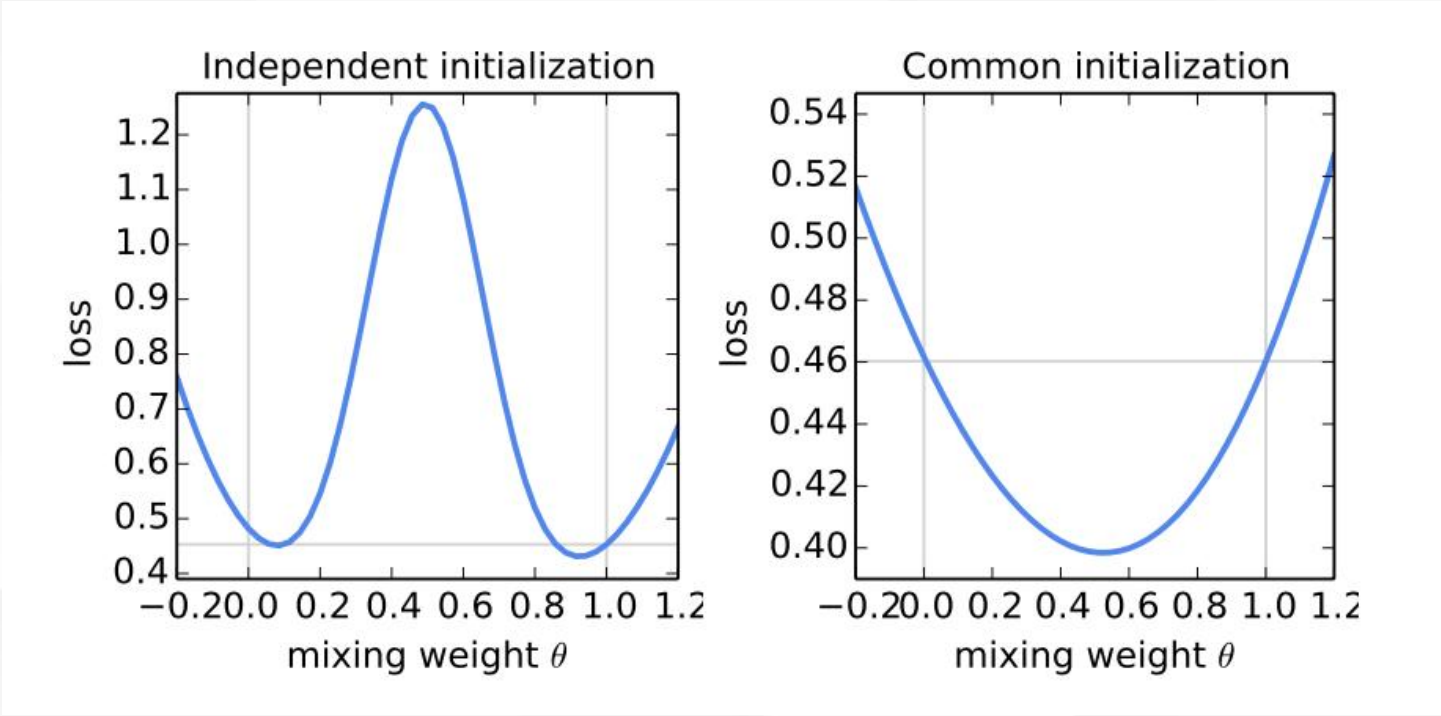
B： Batch size

E： Local training rounds

η： Local learning rate

FedAvg algorithm

- Federated Averaging
  - It is beneficial to initialize local models with the common start point
  - Mixed weight = θw + (1 - θ)w′



FedAvg algorithm

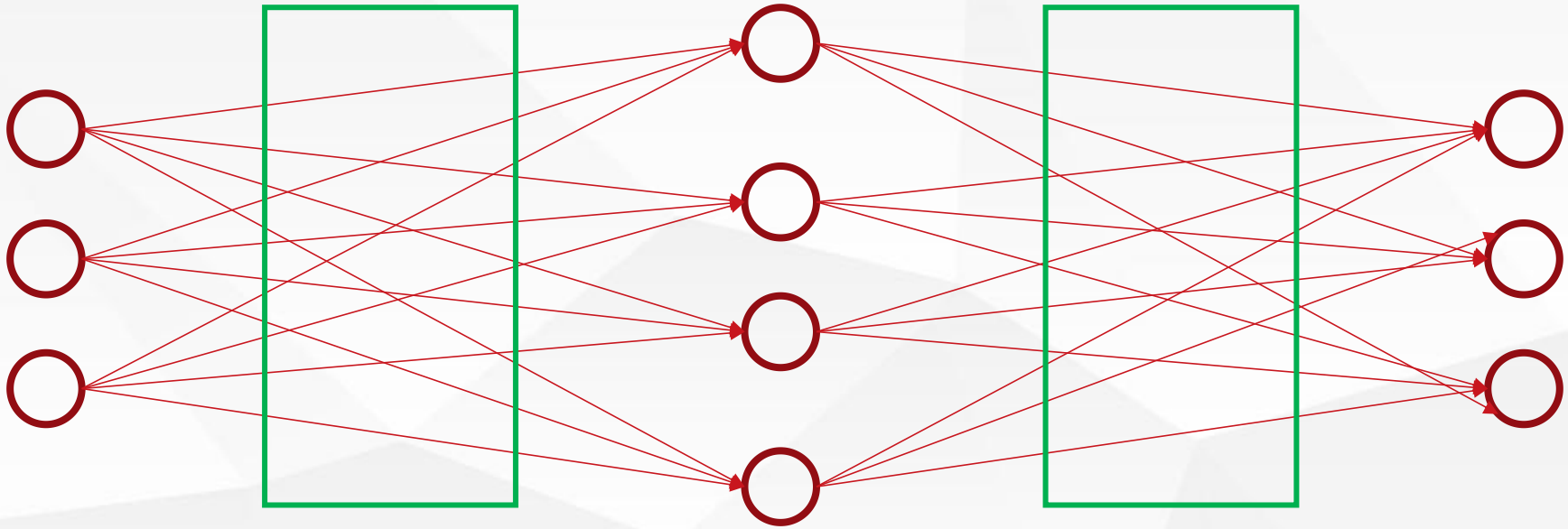
- Experiments
  - Image classification: MNIST handwritten digital recognition
  - Language modeling: Dataset based on the Complete Works of William Shakespeare

FedAvg algorithm

- Experiments (MNIST)
  - IID: shuffling the data and dividing it into 100 clients, each receiving 600 examples.
  - Non-IID: (1) sort the data by number label; (2) divide it into 200 shards of size 300; (3) assign 2 shards to each of the 100 clients, most clients will have only two number examples.

FedAvg algorithm

- Experiments (MNIST)
  - 1) A simple multilayer-perceptron with 2-hidden layers with 200 units each using ReLU activations (199,210 total parameters), which we refer to as the MNIST 2NN.



FedAvg algorithm

- Experiments (MNIST)
  - 1) A simple multilayer-perceptron with 2-hidden layers with 200 units each using ReLU activations (199,210 total parameters), which we refer to as the MNIST 2NN.

```
class FedAvgMLP(nn.Module):
    def __init__(self, in_features=784, num_classes=10, hidden_dim=200):
        super().__init__()
        self.fc1 = nn.Linear(in_features, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, num_classes)
        self.act = nn.ReLU(inplace=True)

    def forward(self, x):
        if x.ndim == 4:
            x = x.view(x.size(0), -1)
        x = self.act(self.fc1(x))
        x = self.fc2(x)
        return x
```

FedAvg algorithm

- Experiments (MNIST)
  - 2) A CNN with two 5x5 convolution layers (the first with 32 channels, the second with 64, each followed with 2x2 max pooling), a fully connected layer with 512 units and ReLU activation, and a final softmax output layer (1,663,370 total parameters).

```
class FedAvgCNN(nn.Module):
    def __init__(self, in_features=1, num_classes=10, dim=1024):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_features,
                      32,
                      kernel_size=5,
                      padding=0,
                      stride=1,
                      bias=True),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=(2, 2))
        )
```

FedAvg algorithm

- Experiments (Key hyperparameters)
  - C: Proportion of clients participating in calculation: 1 indicates that all clients participate in training
  - E: Number of training cycles per client between two communications
  - B: Mini-batch size of each client. ∞ indicates full-batch
  - $u_k = E\frac{n_k}{B}$: The total number of updates in each iteration on client k

FedAvg algorithm

- Experiments (communication rounds)

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×)

C = 0.1 is the best

C: Proportion of clients participating in calculation
E: Number of training cycles per client between two communications
B: Mini-batch size of each client. ∞ indicates full-batch
$u_k = E\frac{n_k}{B}$: The total number of updates in each iteration on client k

FedAvg algorithm

- Experiments (communication rounds)

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×)

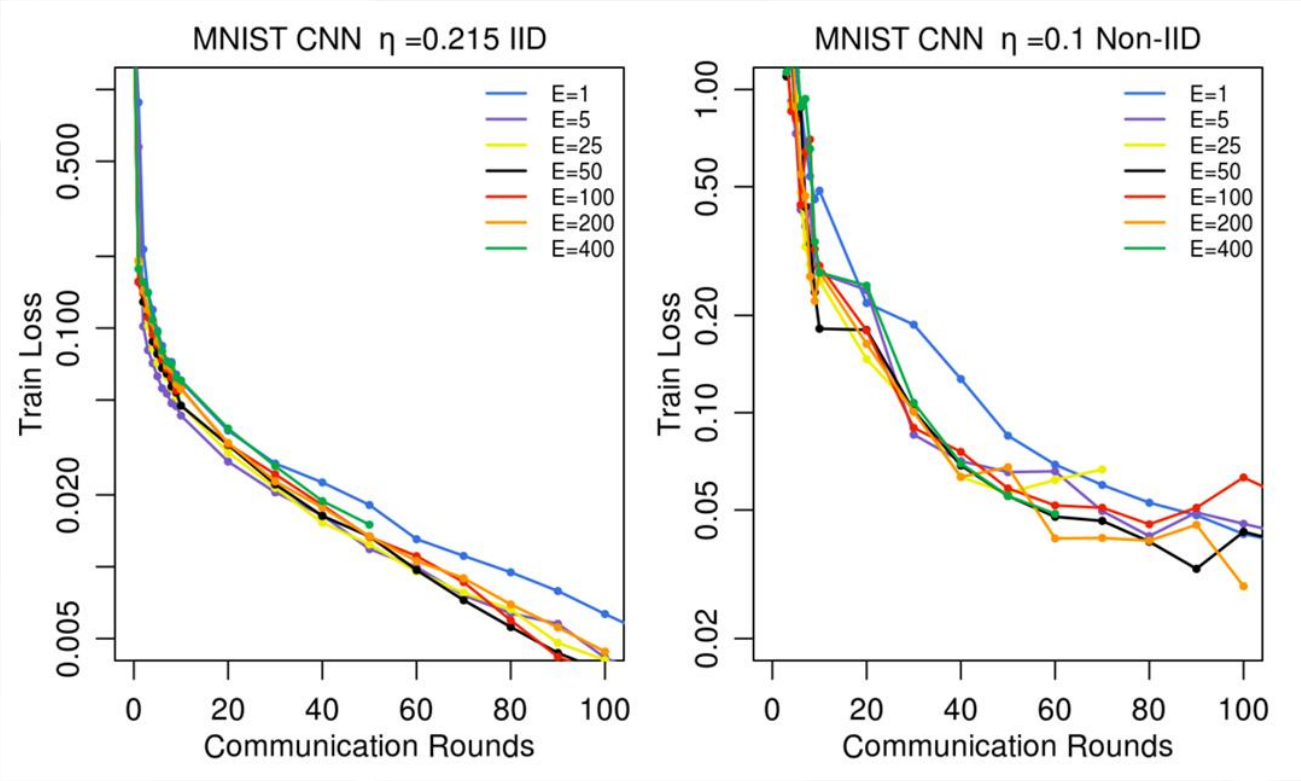
FedSGD: C = 1 and Full-Batch Optimization

FedAvg: C = 0.1

C: Proportion of clients participating in calculation
E: Number of training cycles per client between two communications
B: Mini-batch size of each client. ∞ indicates full-batch
$u_k = E\frac{n_k}{B}$: The total number of updates in each iteration on client k

FedAvg algorithm

- Experiments (training loss curves)



C: Proportion of clients participating in calculation, C=0.1

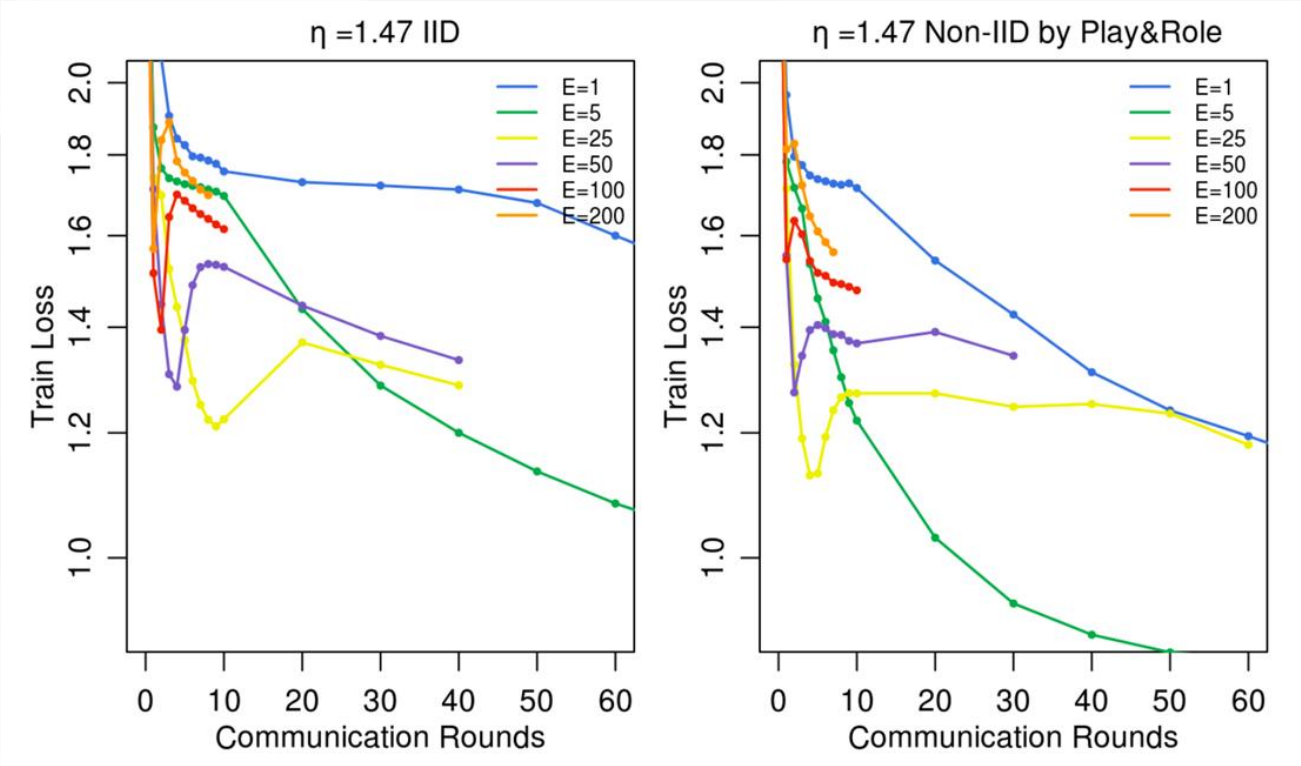
E: Number of training cycles per client between two communications

B: Mini-batch size of each client. B=10.

$u_k = E\frac{n_k}{B}$: **The total number of updates in each iteration on client k**

FedAvg algorithm

- Experiments (training loss curves)



LSTM

C: Proportion of clients participating in calculation, C=0.1
E: Number of training cycles per client between two communications
B: Mini-batch size of each client. B=10.
$u_k = E\frac{n_k}{B}$: The total number of updates in each iteration on client k

02

Benchmark algorithms

- MOON
- FedDyn
- KT-pFL
- FedMA

MOON

MOON algorithm

Observation

- The global model trained on a whole dataset can learn a better representation than the local model trained on a skewed subset.

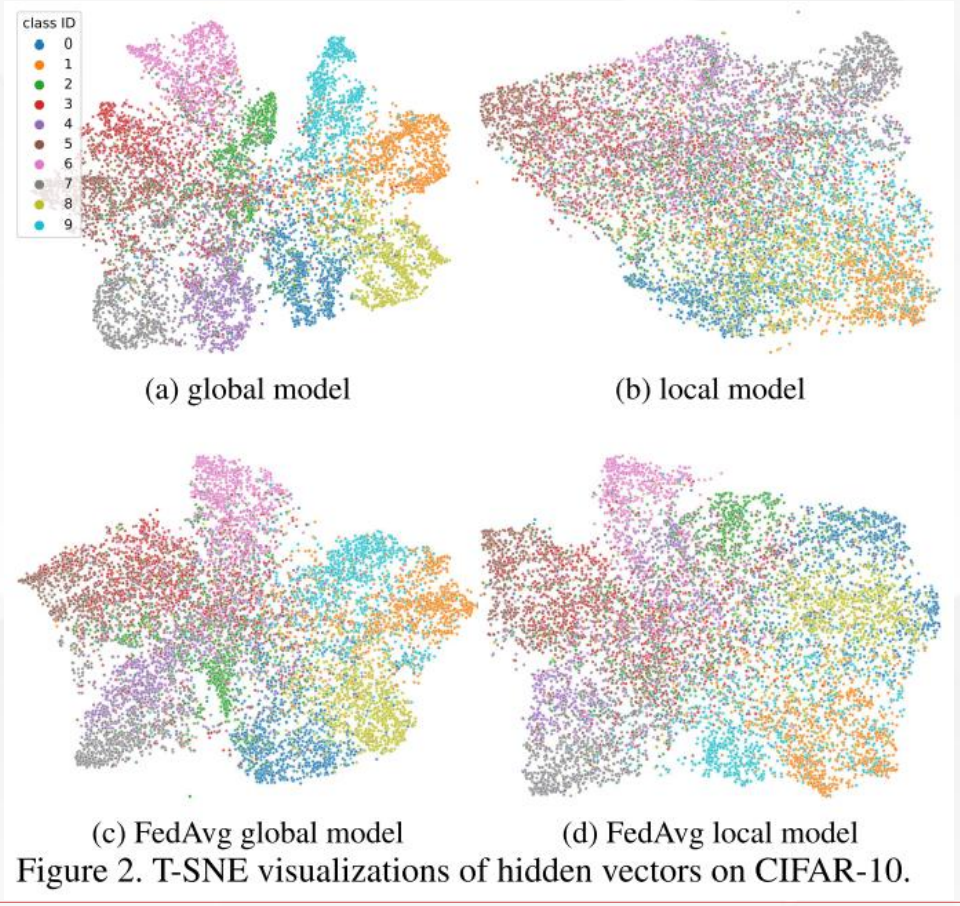


Figure 2. T-SNE visualizations of hidden vectors on CIFAR-10.

MOON algorithm

- Observation
  - The global model trained on a whole dataset can learn a better representation than the local model trained on a skewed subset.
  - Propose model-contrastive learning (MOON), which corrects the local updates by maximizing the agreement of representation learned by the current local model and the representation learned by the global model.

MOON algorithm

- Contrastive learning
  - The key idea of contrastive learning is to reduce the distance between the representations of different augmented views of the same image (i.e., positive pairs), and increase the distance between the representations of augmented views of different images (i.e., negative pairs)

$$l_{i,j} = -\log \frac{\exp(\text{sim}(x_i, x_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} \exp(\text{sim}(x_i, x_k)/\tau)}$$

MOON algorithm

- MOON
  - Global representation $z_{glob} = R_{w^t}(x)$
  - Local representation w_i^{t-1} (i.e., $z_{prev} = R_{w_i^{t-1}}(x)$)
  - Current representation $z = R_{w_i^t}(x)$
  -

$$\ell_{con} = -\log \frac{\exp(\text{sim}(z, z_{glob})/\tau)}{\exp(\text{sim}(z, z_{glob})/\tau) + \exp(\text{sim}(z, z_{prev})/\tau)}$$

$$\ell = \ell_{sup}(w_i^t; (x, y)) + \mu \ell_{con}(w_i^t; w_i^{t-1}; w^t; x),$$

$$\min_{w_i^t} \mathbb{E}_{(x,y)\sim D^i} [\ell_{sup}(w_i^t; (x, y)) + \mu \ell_{con}(w_i^t; w_i^{t-1}; w^t; x)].$$

MOON algorithm

MOON

Algorithm 1: The MOON framework

Input: number of communication rounds T, number of parties N, number of local epochs E, temperature τ, learning rate η, hyper-parameter μ

Output: The final model w^T

Server executes:
initialize w^0
for $t = 0, 1, ..., T-1$ **do**
  for $i = 1, 2, ..., N$ **in parallel do**
    send the global model w^t to P_i
    $w_i^t \leftarrow$ **PartyLocalTraining**(i, w^t)
  $w^{t+1} \leftarrow \sum_{k=1}^{N} \frac{|\mathcal{D}^i|}{|\mathcal{D}|} w_k^t$
return w^T

PartyLocalTraining(i, w^t):
$w_i^t \leftarrow w^t$
for epoch $i = 1, 2, ..., E$ **do**
  for each batch $\mathbf{b} = \{x, y\}$ of $\mathcal{D}^i$ **do**
    $\ell_{sup} \leftarrow CrossEntropyLoss(F_{w_i^t}(x), y)$
    $z \leftarrow R_{w_i^t}(x)$
    $z_{glob} \leftarrow R_{w^t}(x)$
    $z_{prev} \leftarrow R_{w_i^{t-1}}(x)$
    $\ell_{con} \leftarrow -\log \frac{\exp(\text{sim}(z, z_{glob})/\tau)}{\exp(\text{sim}(z, z_{glob})/\tau) + \exp(\text{sim}(z, z_{prev})/\tau)}$
    $\ell \leftarrow \ell_{sup} + \mu \ell_{con}$
    $w_i^t \leftarrow w_i^t - \eta \nabla \ell$
return w_i^t to server

FedDyn

FedDyn algorithm

- Intuition
  - Training models on local data that minimize local empirical loss appears to be meaningful, but yet, doing so is fundamentally inconsistent with minimizing the global empirical loss.
  - Dynamically modify the device objective with a penalty term so that, in the limit, when model parameters converge, they do so to stationary points of the global empirical loss.

FedDyn algorithm

FedDyn

Algorithm 1: Federated Dynamic Regularizer - (FedDyn)

Input: $T, \boldsymbol{\theta}^0, \alpha > 0, \nabla L_k(\boldsymbol{\theta}_k^0) = \mathbf{0}$.

for $t = 1, 2, \ldots T$ **do**

  Sample devices $\mathcal{P}_t \subseteq [m]$ and transmit $\boldsymbol{\theta}^{t-1}$ to each selected device,

  for *each device* $k \in \mathcal{P}_t$, *and in parallel* **do**

    Set $\boldsymbol{\theta}_k^t = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} L_k(\boldsymbol{\theta}) - \langle \nabla L_k(\boldsymbol{\theta}_k^{t-1}), \boldsymbol{\theta} \rangle + \frac{\alpha}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{t-1}\|^2$,

    Set $\nabla L_k(\boldsymbol{\theta}_k^t) = \nabla L_k(\boldsymbol{\theta}_k^{t-1}) - \alpha \left(\boldsymbol{\theta}_k^t - \boldsymbol{\theta}^{t-1}\right)$,

    Transmit device model $\boldsymbol{\theta}_k^t$ to server,

  end for

  for *each device* $k \notin \mathcal{P}_t$, *and in parallel* **do**

    Set $\boldsymbol{\theta}_k^t = \boldsymbol{\theta}_k^{t-1}, \nabla L_k(\boldsymbol{\theta}_k^t) = \nabla L_k(\boldsymbol{\theta}_k^{t-1})$,

  end for

  Set $\boldsymbol{h}^t = \boldsymbol{h}^{t-1} - \alpha \frac{1}{m} \left(\sum_{k \in \mathcal{P}_t} \boldsymbol{\theta}_k^t - \boldsymbol{\theta}^{t-1}\right)$,

  Set $\boldsymbol{\theta}^t = \left(\frac{1}{|\mathcal{P}_t|} \sum_{k \in \mathcal{P}_t} \boldsymbol{\theta}_k^t\right) - \frac{1}{\alpha} \boldsymbol{h}^t$

end for

FedDyn algorithm

- Analysis

$$\boldsymbol{\theta}_k^t = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\mathfrak{R}_k(\boldsymbol{\theta}; \boldsymbol{\theta}_k^{t-1}, \boldsymbol{\theta}^{t-1}) \triangleq L_k(\boldsymbol{\theta}) - \langle \nabla L_k(\boldsymbol{\theta}_k^{t-1}), \boldsymbol{\theta} \rangle + \frac{\alpha}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}^{t-1}\|^2 \right]$$

- The first order condition

$$\nabla L_k(\boldsymbol{\theta}_k^t) - \nabla L_k(\boldsymbol{\theta}_k^{t-1}) + \alpha(\boldsymbol{\theta}_k^t - \boldsymbol{\theta}^{t-1}) = \mathbf{0}$$

- If local device models converge, they converge to the server model, and the convergence point is a stationary point of the global loss.

if $\boldsymbol{\theta}_k^t \to \boldsymbol{\theta}_k^\infty$, it generally follows that, $\nabla L_k(\boldsymbol{\theta}_k^t) \to \nabla L_k(\boldsymbol{\theta}_k^\infty)$, and as a consequence, we have $\boldsymbol{\theta}^t \to \boldsymbol{\theta}_k^\infty$. In turn this implies that $\boldsymbol{\theta}_k^\infty \to \boldsymbol{\theta}^\infty$, i.e., is independent of k.

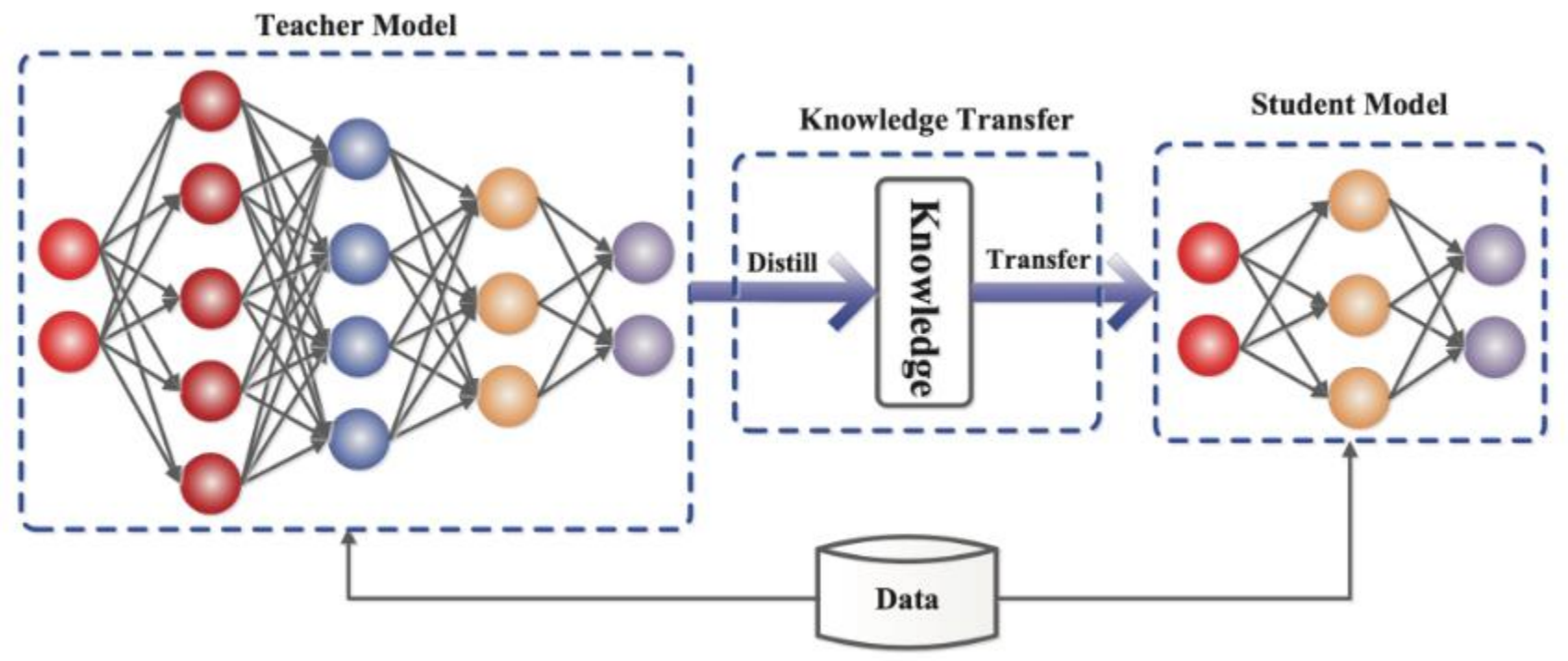
KT-pFL

KT-pFL algorithm

- Intuition
  - Main idea is to allow each client to maintain a personalized soft prediction at the server that can be updated by a linear combination of all clients local soft predictions using a knowledge coefficient matrix.
  - Regardless of model structures

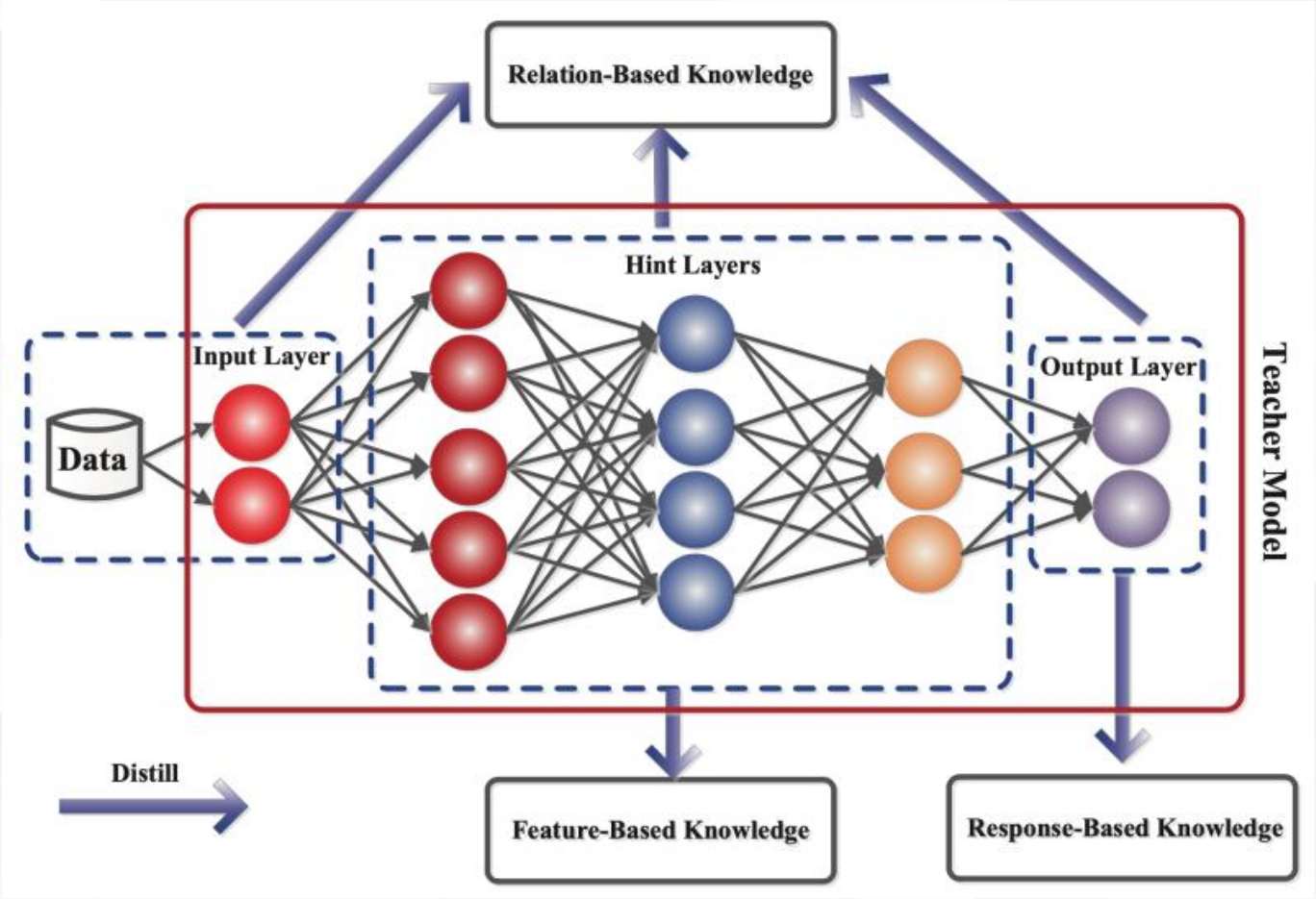
KT-pFL algorithm

- Knowledge Distillation (KD)
  - Transfer knowledge from well-learned teacher model to student model



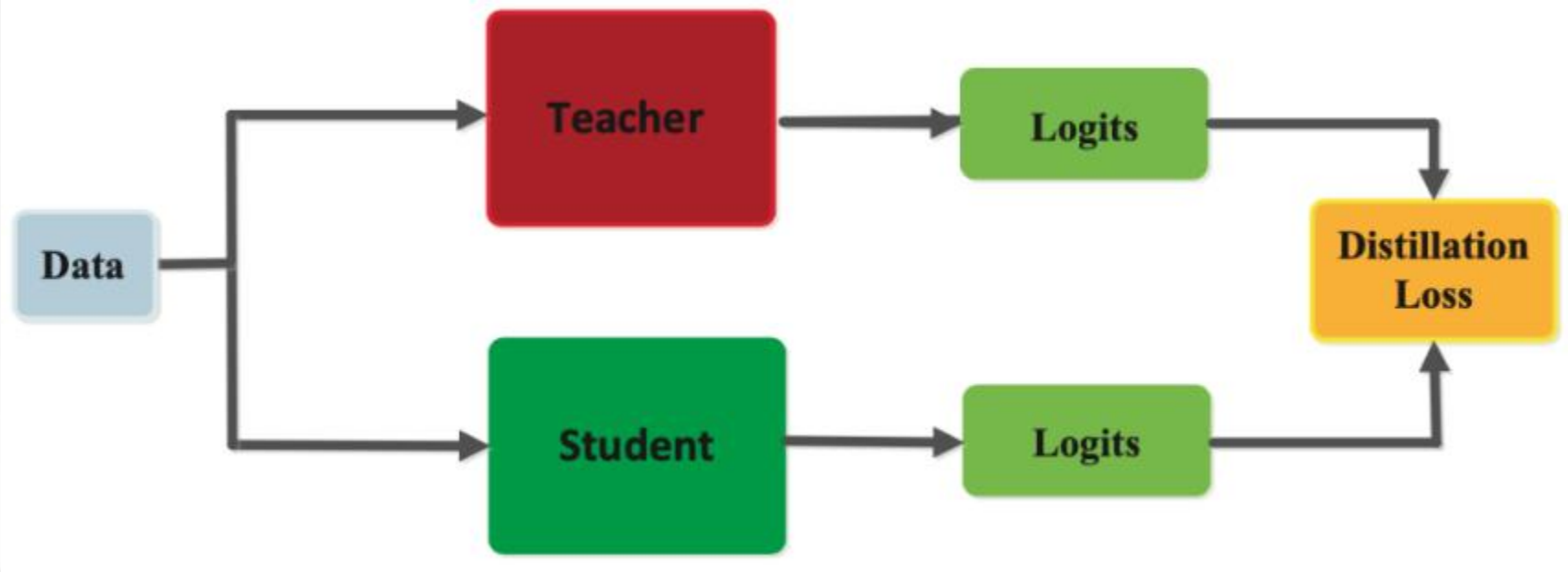
KT-pFL algorithm

- Knowledge Distillation (KD)
  - Classification



KT-pFL algorithm

- Knowledge Distillation (KD)
  - Response-based KD



KT-pFL algorithm

- KT-pFL
  - Objective

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) := \sum_{n=1}^{N} \frac{D_n}{D} \mathcal{L}_n(\mathbf{w}), \text{ where } \mathcal{L}_n(\mathbf{w}) = \frac{1}{D_n} \sum_{i=1}^{D_n} \mathcal{L}_{CE}(\mathbf{w}; x_i, y_i)$$

$$\min_{\mathbf{w}^1,\cdots,\mathbf{w}^N} \mathcal{L}(\mathbf{w}^1, \cdots, \mathbf{w}^N) := \sum_{n=1}^{N} \frac{D_n}{D} L_n(\mathbf{w}^n)$$

KT-pFL algorithm

- KT-pFL
  - Personalized loss function
    - Kullback–Leibler (KL) Divergence
    - c_{mn} is the knowledge coefficient which is used to estimate the contribution from client m to n.
    - $s(w^n, \hat{x})$ can be deemed to be a soft prediction of the client n

$$\mathcal{L}_{per,n}(\mathbf{w}^n) := \mathcal{L}_n(\mathbf{w}^n) + \lambda \sum_{\hat{x} \in \mathbb{D}_r} \mathcal{L}_{KL}\left(\sum_{m=1}^{N} c_{mn} \cdot s(\mathbf{w}^m, \hat{x}), s(\mathbf{w}^n, \hat{x})\right)$$

$$s(\mathbf{w}^n, \hat{x}) = \frac{\exp(z_c^n/T)}{\sum_{c=1}^{C} \exp(z_c^n/T)},$$

KT-pFL algorithm

- KT-pFL
  - Knowledge coefficient matrix

$$\mathbf{c} = \begin{Bmatrix} c_{11} & c_{12} & \cdots & c_{1N} \\ c_{21} & c_{22} & \cdots & c_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ c_{N1} & c_{N2} & \cdots & c_{NN} \end{Bmatrix}.$$

KT-pFL algorithm

- KT-pFL
  - Objective

$$\min_{\mathbf{w},\mathbf{c}} \mathcal{L}(\mathbf{w},\mathbf{c}) := \sum_{n=1}^{N} \frac{D_n}{D} \mathcal{L}_{per,n}(\mathbf{w}^n) + \rho \|\mathbf{c} - \frac{\mathbf{1}}{N}\|^2$$

$$\mathbf{w} = [\mathbf{w}^1, \cdots, \mathbf{w}^N] \in \mathbb{R}^{\sum_{n=1}^{N} d_n}$$

KT-pFL algorithm

- KT-pFL
  - Training
    - Update w
      - Local Training
      - Distillation
    - Update c

$$\mathbf{w}^n \leftarrow \mathbf{w}^n - \eta_1 \nabla_{\mathbf{w}^n} \mathcal{L}_n(\mathbf{w}^n; \xi_n),$$

$$\mathbf{w}^n \leftarrow \mathbf{w}^n - \eta_2 \nabla_{\mathbf{w}^n} \mathcal{L}_{KL}\left(\sum_{m=1}^{N} \mathbf{c}_m^{*,T} \cdot s(\mathbf{w}^m, \xi_r), s(\mathbf{w}^n, \xi_r)\right)$$

$$\mathbf{c} \leftarrow \mathbf{c} - \eta_3 \lambda \sum_{n=1}^{N} \frac{D_n}{D} \nabla_{\mathbf{c}} \mathcal{L}_{KL}\left(\sum_{m=1}^{N} \mathbf{c}_m \cdot s(\mathbf{w}^{m,*}, \xi_r), s(\mathbf{w}^{n,*}, \xi_r)\right) - 2\eta_3 \rho(\mathbf{c} - \frac{\mathbf{1}}{N})$$

KT-pFL algorithm

- KT-pFL

Algorithm 1 KT-pFL Algorithm

Input: $\mathbb{D}, \mathbb{D}_r, \eta_1, \eta_2, \eta_3$ and T
Output: $\mathbf{w} = [\mathbf{w}^1, \cdots, \mathbf{w}^N]$

Initialize $\mathbf{w}_0$ and $\mathbf{c}_0$
procedure SERVER-SIDE OPTIMIZATION
    Distribute $\mathbf{w}_0$ and $\mathbf{c}_0$ to each client
    for each communication round $t \in \{1, 2, ..., T\}$ **do**
        for each client n **in parallel do**
            $\mathbf{w}_{t+1}^n \leftarrow ClientLocalUpdate(n, \mathbf{w}_t^n, \mathbf{c}_{t,n})$
        Update knowledge coefficient matrix $\mathbf{c}$ via (7)
        Distribute $\mathbf{c}_{t+1}$ to all clients
procedure CLIENTLOCALUPDATE$(n, \mathbf{w}_t^n, \mathbf{c}_{t,n})$
    Client n receives $\mathbf{w}_t^n$ and $\mathbf{c}_n$ from the server
    for each local epoch i from 1 to E **do**
        for mini-batch $\xi_t \subseteq \mathbb{D}_n$ **do**
            Local Training: update model parameters on private data via (5)
    for each distillation step j from 1 to R **do**
        for mini-batch $\xi_{r,t} \subseteq \mathbb{D}_r$ **do**
            Distillation: update model parameters on public data via (6)
    return local parameters $\mathbf{w}_{t+1}^n$

KT-pFL algorithm

KT-pFL

- Illustration

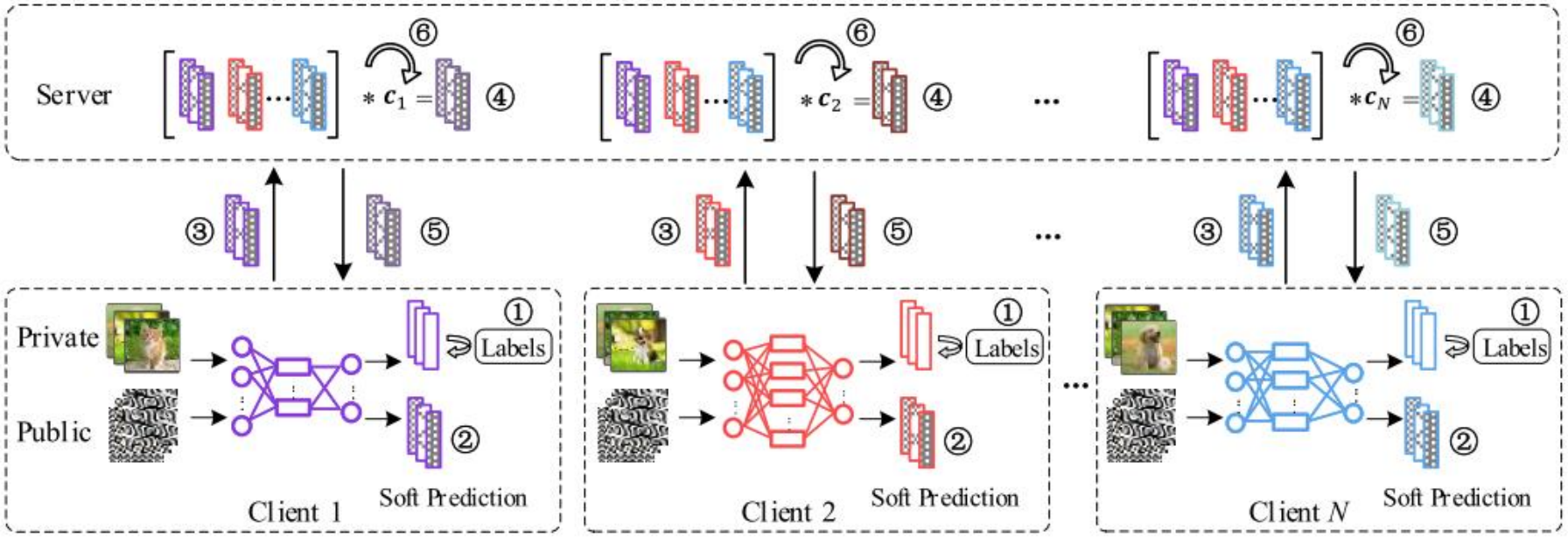
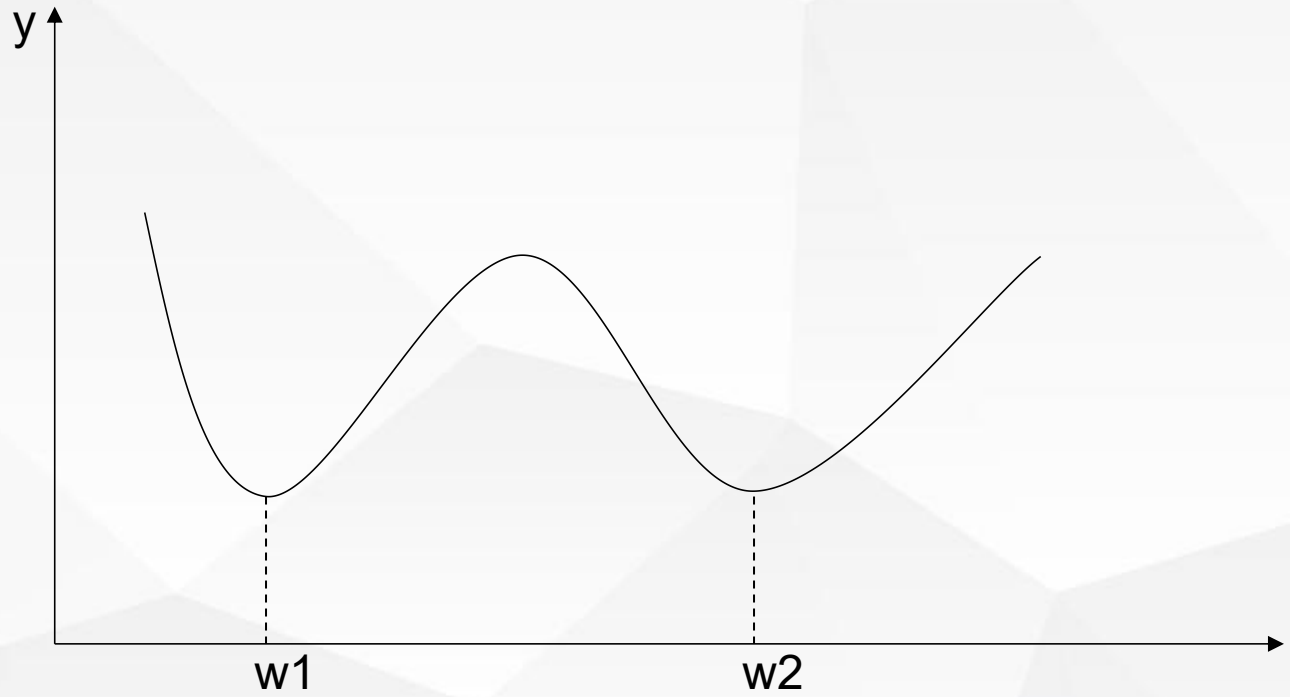


Figure 1: Illustration of the KT-pFL framework. The workflow includes 6 steps: ① local training on private data; ②, ③ each client outputs the local soft prediction on public data and sends it to the server; ④ the server calculates each client's personalized soft prediction via a linear combination of local soft predictions and knowledge coefficient matrix; ⑤ each client downloads the personalized soft prediction to perform distillation phase; ⑥ the server updates the knowledge coefficient matrix.

FedMA

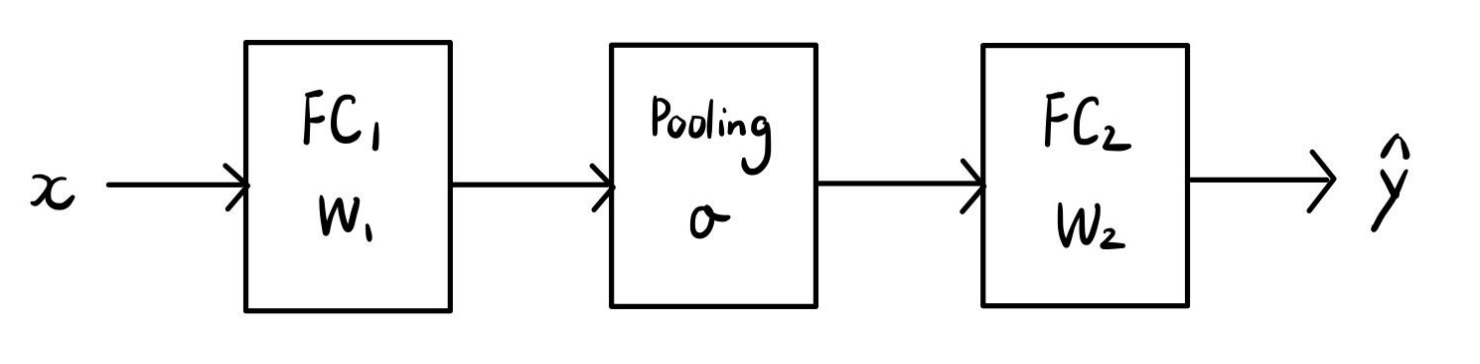
FedMA algorithm

- Permutation invariance



FedMA algorithm

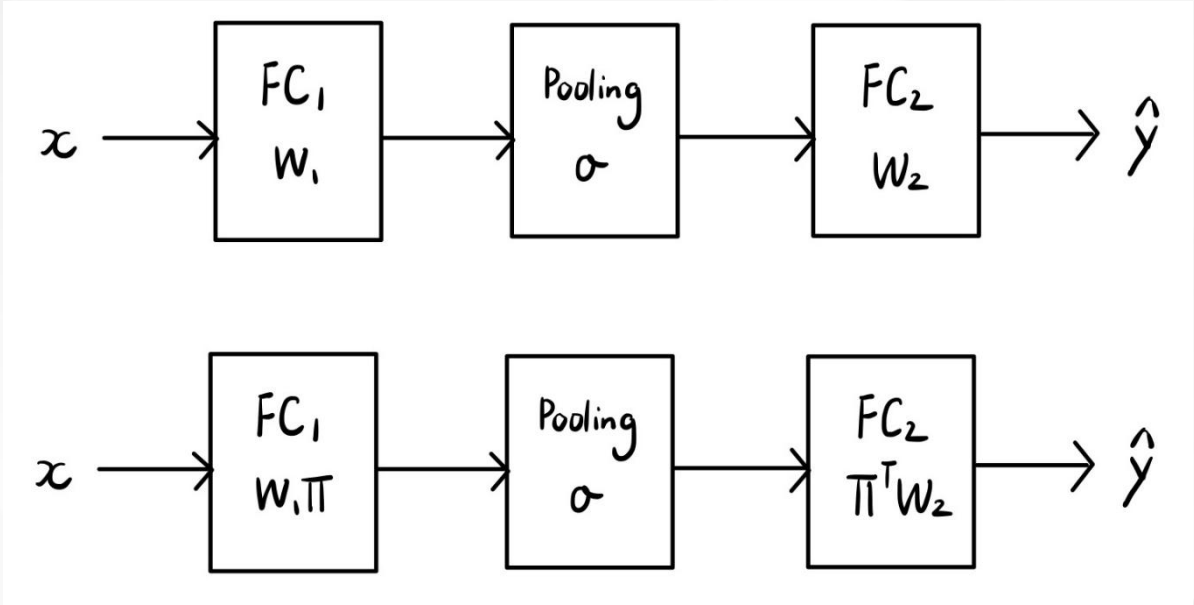
- Permutation invariance (fully-connected (FC) layer)



$$\hat{y} = \sigma(xW_1)W_2$$

FedMA algorithm

- Permutation invariance (fully-connected (FC) layer)



$$\hat{y} = \sigma(xW_1\Pi)\Pi^T W_2, \text{ where } \Pi \text{ is any } L \times L \text{ permutation matrix.}$$

FedMA algorithm

- Permutation invariance (fully-connected (FC) layer)

Client A:

$$\{W_1\Pi_j, \Pi_j^T W_2\}$$

Client B:

$$\{W_1\Pi_{j'}, \Pi_{j'}^T W_2\}$$

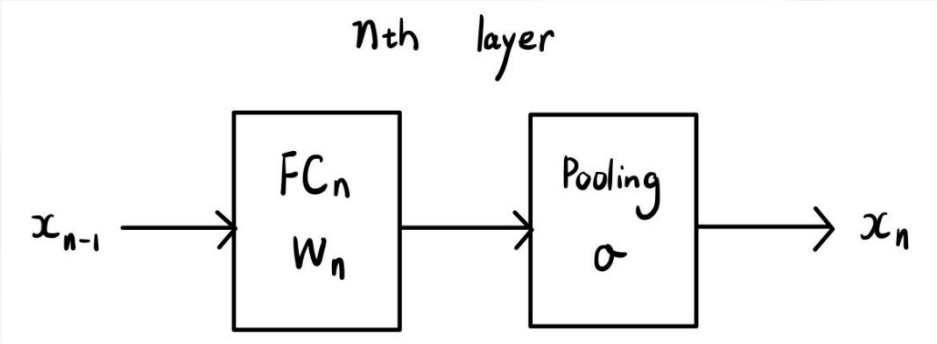
Aggregation

$$(W_1\Pi_j + W_1\Pi_{j'})/2 \neq W_1\Pi \text{ for any } \Pi$$

Solution: $(W_1\Pi_j\Pi_j^T + W_1\Pi_{j'}\Pi_{j'}^T)/2 = W_1$

FedMA algorithm

- Permutation invariance (FCs)

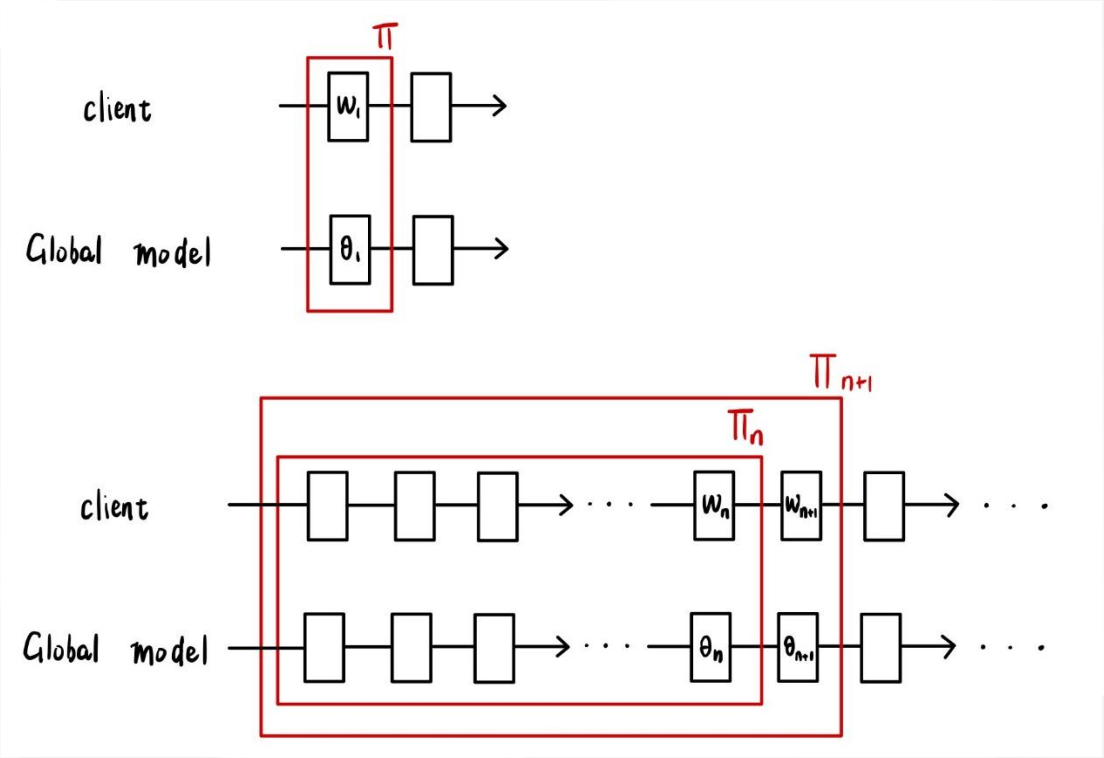


Simple FCs: $\hat{y} = \sigma(xW_1\Pi)\Pi^T W_2$

Deep FCs: $x_n = \sigma(x_{n-1}\Pi_{n-1}^T W_n \Pi_n)$

FedMA algorithm

- Permutation invariance (FCs)



$$x_n = \sigma(x_{n-1}\Pi_{n-1}^T W_n \Pi_n)$$

FedMA algorithm

- Permutation invariance （CNN)

FC : $x_n = \sigma(x_{n-1}\Pi_{n-1}^T W_n \Pi_n)$

CNN : $x_n = \sigma(\mathbf{Conv}(x_{n-1}, \Pi_{n-1}^T W_n \Pi_n))$

FedMA algorithm

- Permutation invariance (recall)

Client A:

$$\{W_1\Pi_j, \Pi_j^T W_2\}$$

Client B:

$$\{W_1\Pi_{j'}, \Pi_{j'}^T W_2\}$$

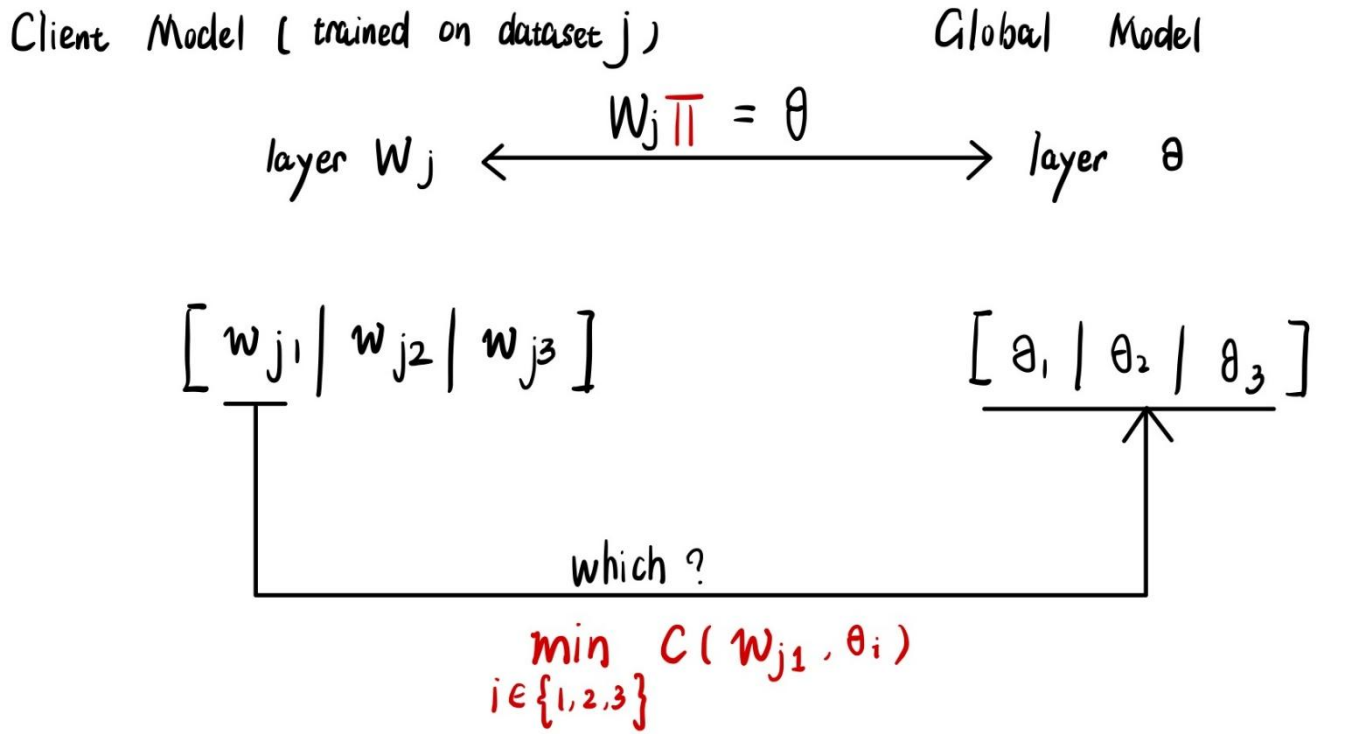
Aggregation

$$(W_1\Pi_j + W_1\Pi_{j'})/2 \neq W_1\Pi \text{ for any } \Pi$$

Solution: $(W_1\Pi_j\Pi_j^T + W_1\Pi_{j'}\Pi_{j'}^T)/2 = W_1$

FedMA algorithm

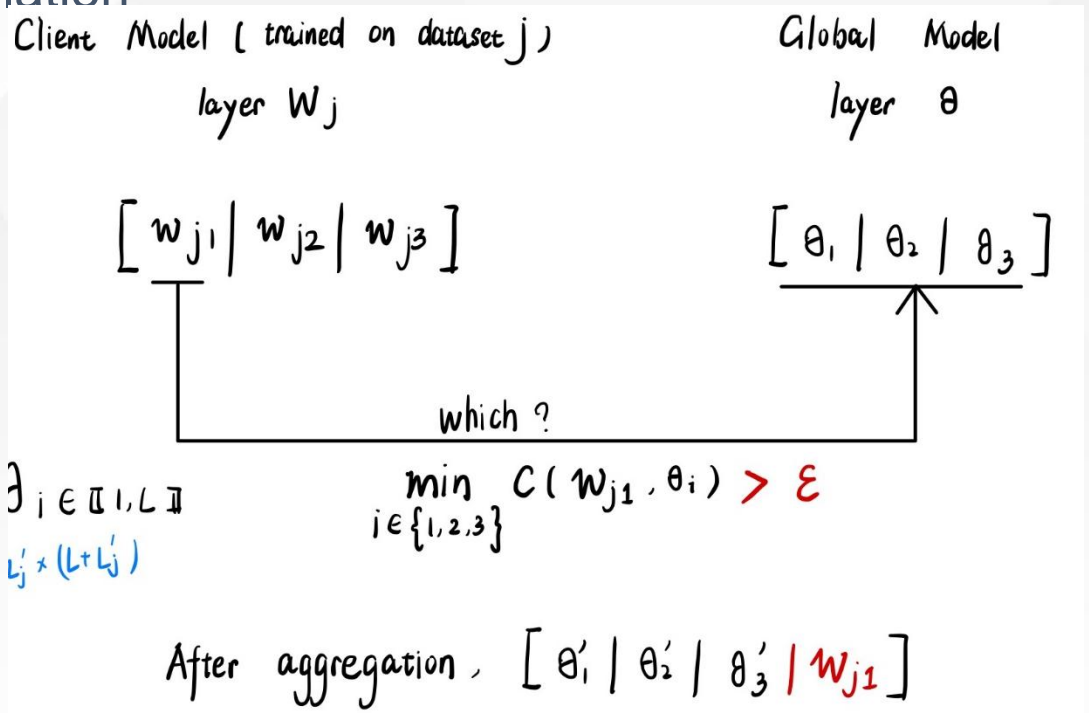
- Matched averaging formulation



$$\min_{\{\pi_{li}^j\}} \sum_{i=1}^{L} \sum_{j,l} \min_{\theta_i} \pi_{li}^j c(w_{jl}, \theta_i) \text{ s.t. } \sum_i \pi_{li}^j = 1 \; \forall \, j, l; \; \sum_l \pi_{li}^j = 1 \; \forall \, i, j.$$

FedMA algorithm

- Matched averaging formulation

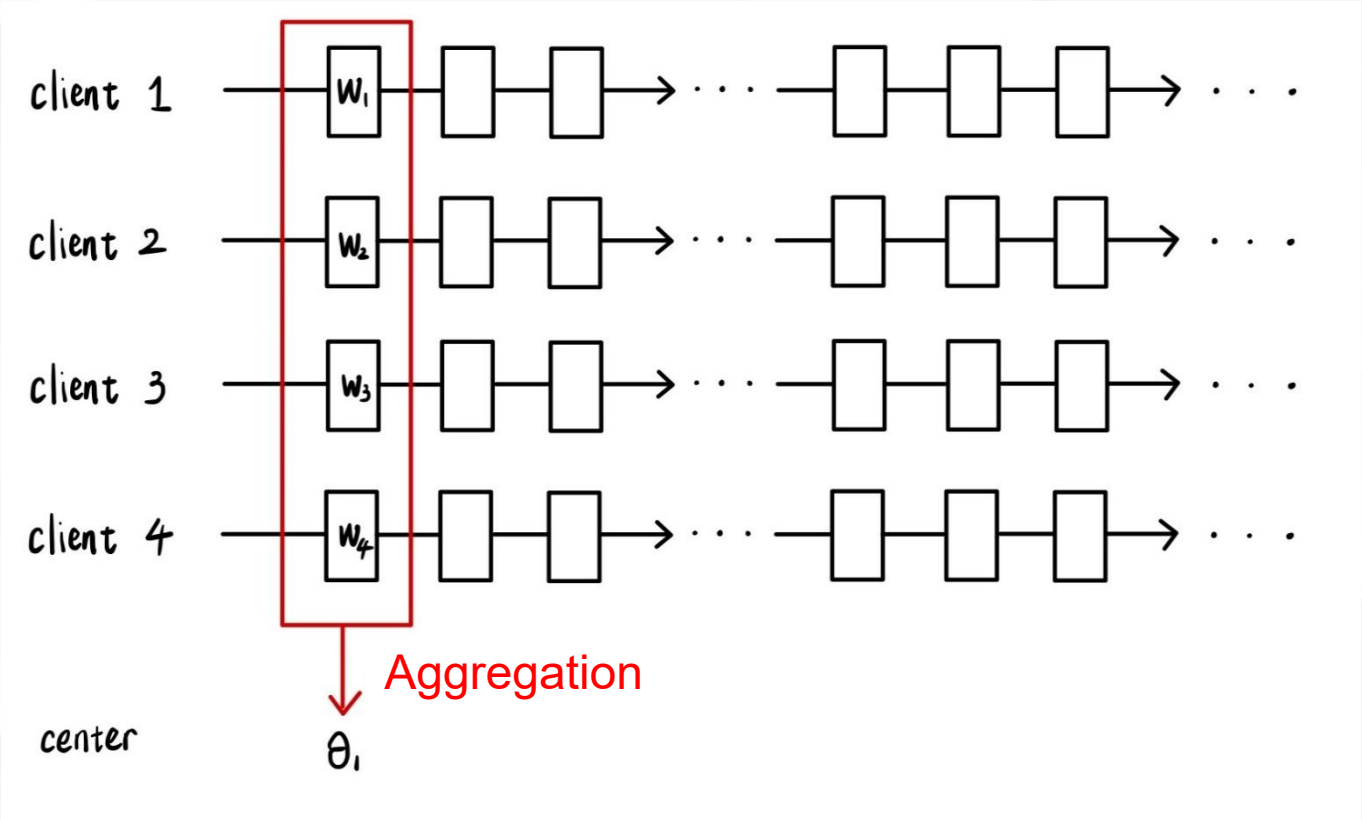


$$\min_{\{\pi_{li}^{j'}\}_{l,i}} \sum_{i=1}^{L+L_{j'}} \sum_{j=1}^{L_{j'}} \pi_{li}^{j'} C_{li}^{j'} \text{ s.t. } \sum_{i} \pi_{li}^{j'} = 1 \; \forall\, l; \; \sum_{l} \pi_{li}^{j} \in \{0,1\} \; \forall\, i, \text{ where}$$

$$C_{li}^{j'} = \begin{cases} c(w_{j'l}, \theta_i), & i \leq L \\ \epsilon + f(i), & L < i \leq L + L_{j'}. \end{cases}$$

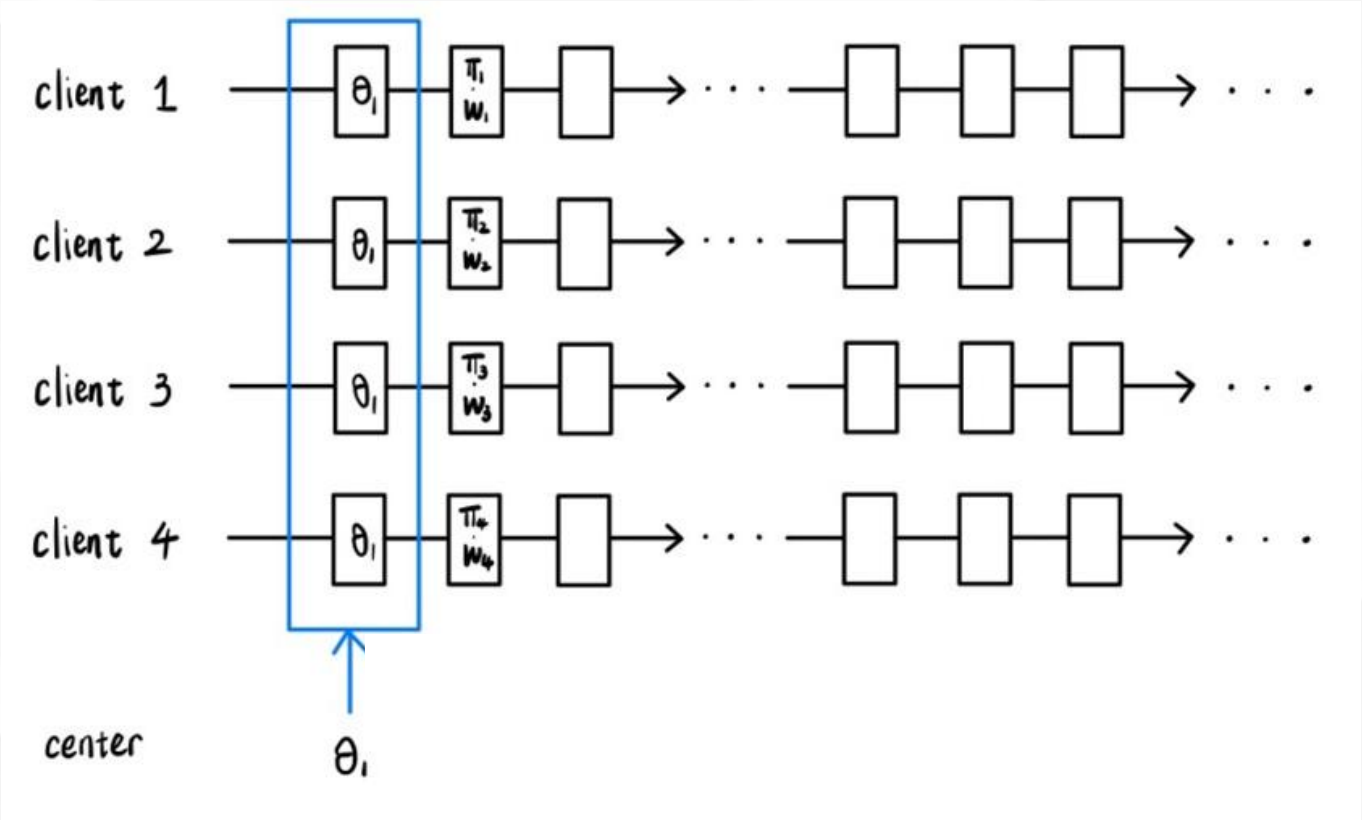
FedMA algorithm

- FedMA (https://github.com/IBM/FedMA)



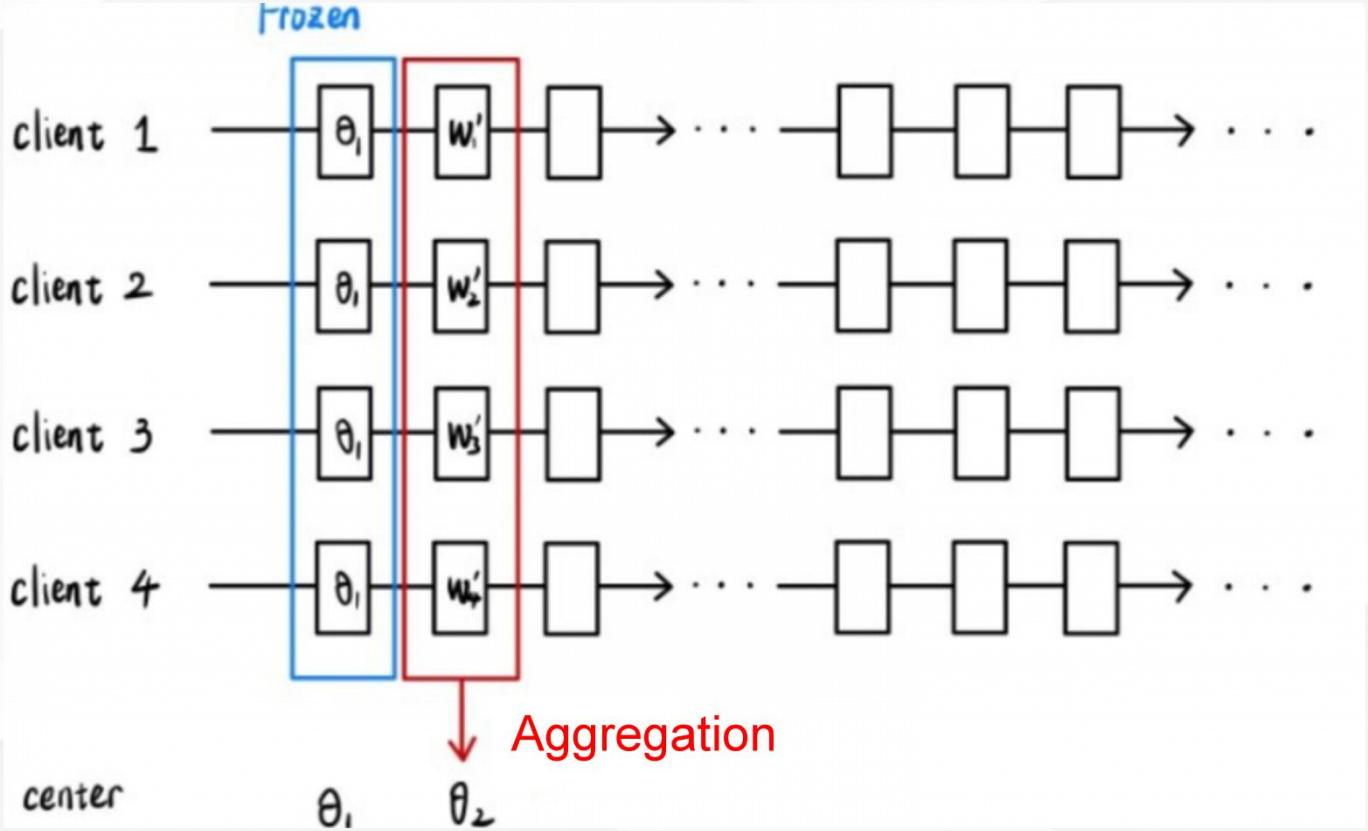
FedMA algorithm

- FedMA (https://github.com/IBM/FedMA)



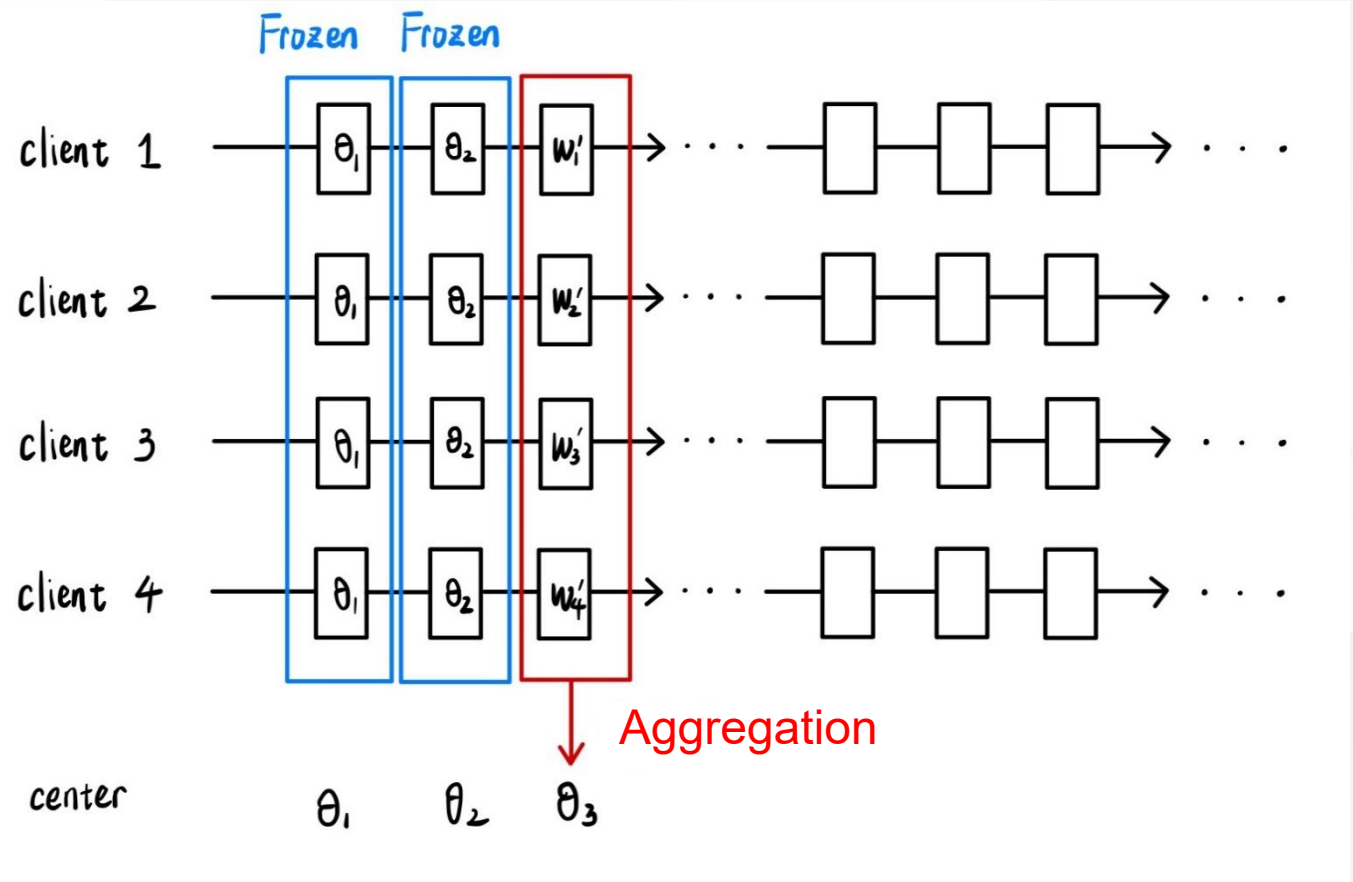
FedMA algorithm

- FedMA (https://github.com/IBM/FedMA)



FedMA algorithm

- FedMA (https://github.com/IBM/FedMA)



FedMA algorithm

- FedMA (https://github.com/IBM/FedMA)

Algorithm 1: Federated Matched Averaging (FedMA)

Input : local weights of N-layer architectures $\{W_{j,1}, \ldots, W_{j,N}\}_{j=1}^{J}$ from J clients
Output: global weights $\{W_1, \ldots, W_N\}$
$n = 1$;
while $n \leq N$ **do**
  if $n < N$ **then**
    $\{\Pi_j\}_{j=1}^{J} = \text{BBP-MAP}(\{W_{j,n}\}_{j=1}^{J})$; // call BBP-MAP to solve Eq. 2
    $W_n = \frac{1}{J} \sum_j W_{j,n} \Pi_j^T$;
  else
    $W_n = \sum_{k=1}^{K} \sum_j p_{jk} W_{jl,n}$ where p_k is fraction of data points with label k on worker j;
  end
  for $j \in \{1, \ldots, J\}$ **do**
    $W_{j,n+1} \leftarrow \Pi_j W_{j,n+1}$; // permutate the next-layer weights
    Train $\{W_{j,n+1}, \ldots, W_{j,L}\}$ with W_n frozen;
  end
  $n = n + 1$;
end

03

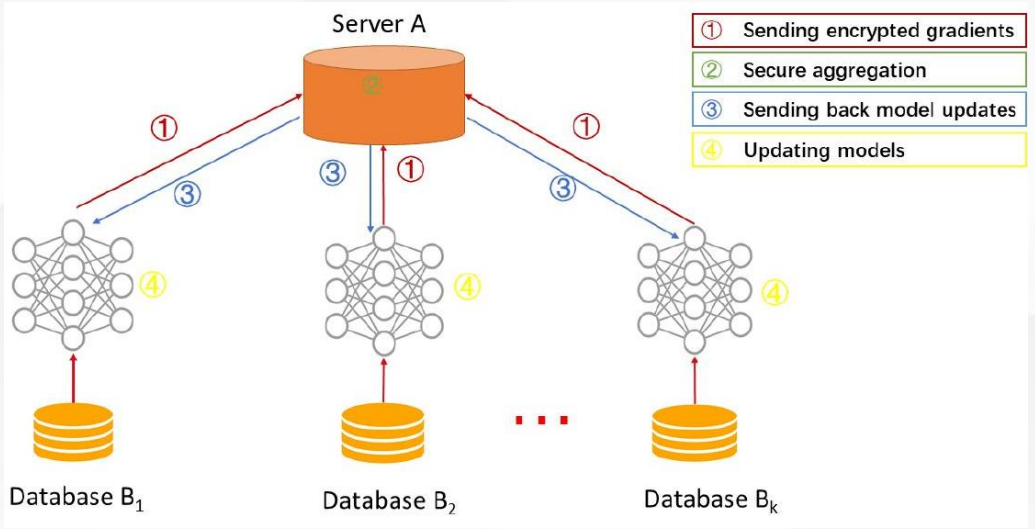
Advanced algorithms

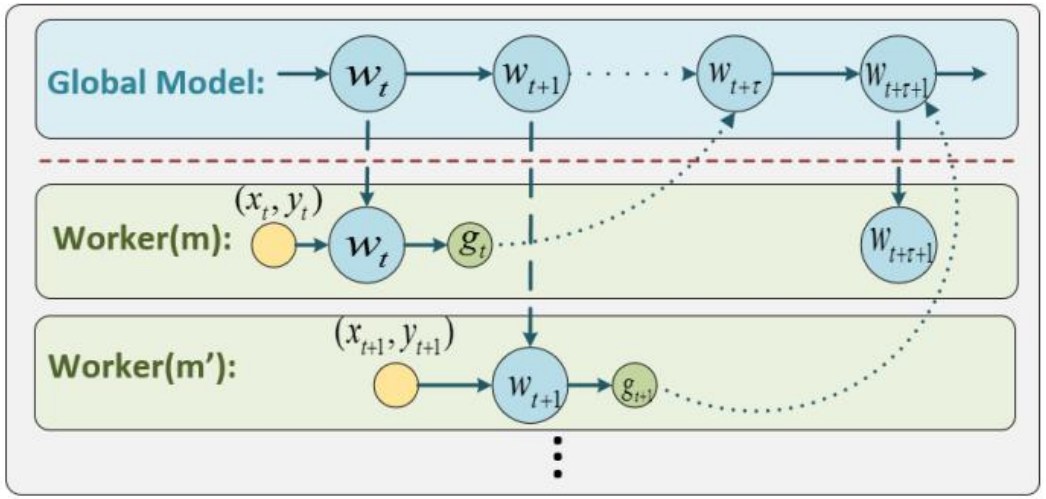
- SageFlow

SageFlow

Sageflow algorithm

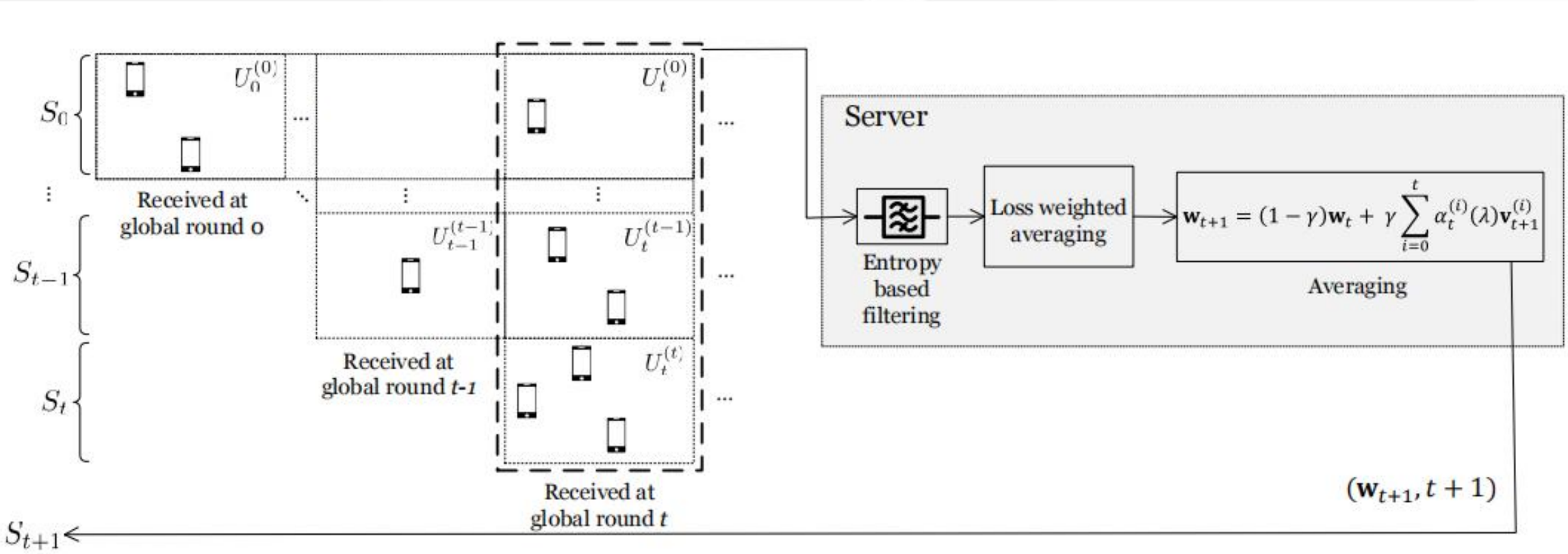
- Stragglers: slow devices
  - **Keep waiting**: slow down the overall process
  - **Drop out**: important data missing
  - **Asynchronous**(staleness): + adversaries?
- Attackers: malicious attacks launched by adversaries
  - **untargeted attacks**: model poisoning, data poisoning
  - **targeted/backdoor attacks**: misclassify the targeted subtasks
  - Robust Federated Averaging & Multi-Krum
- Large portion of adversaries
  - Straggler: increase attack ratio





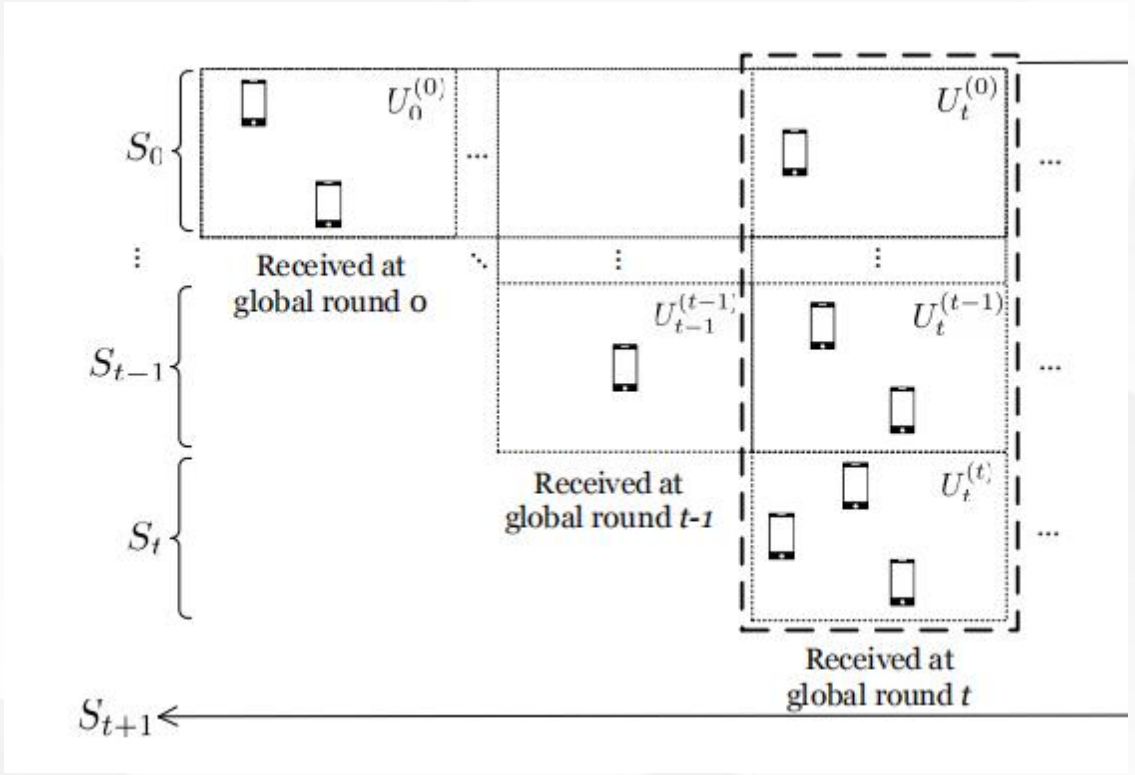
Sageflow algorithm

- Staleness-aware grouping
- Entropy-based filtering + Loss-weighted averaging



Sageflow algorithm

- Staleness-aware grouping
  - perform **periodic** global aggregation(fixed time deadline)
  - allow stragglers to be aggregated in **later rounds**
  - group with **same staleness** -> group representative model
  - aggregate according to **staleness**



Staleness group Number of data samples

$$\mathbf{v}_{t+1}^{(i)} = \sum_{k \in U_t^{(i)}} \frac{m_k}{\sum_{k \in U_t^{(i)}} m_k} \mathbf{w}_i(k)$$

$$\mathbf{w}_{t+1} = (1-\gamma)\mathbf{w}_t + \gamma \sum_{i=0}^{t} \alpha_t^{(i)}(\lambda)\mathbf{v}_{t+1}^{(i)}$$

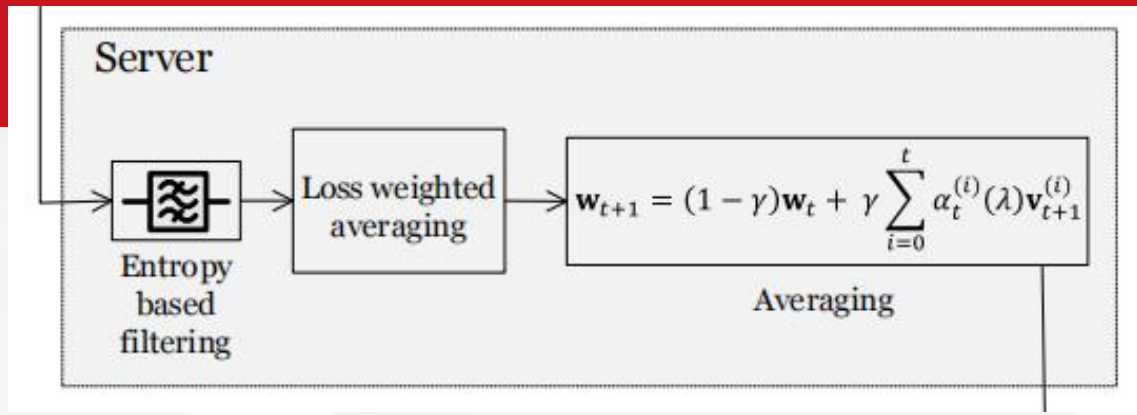
$$\sum_{i=0}^{t} \alpha_t^{(i)}(\lambda)\mathbf{v}_{t+1}^{(i)}$$

$$\alpha_t^{(i)}(\lambda) \propto \frac{\sum_{k \in U_t^{(i)}} m_k}{(t-i+1)^{\lambda}}$$

Staleness function

Sageflow algorithm

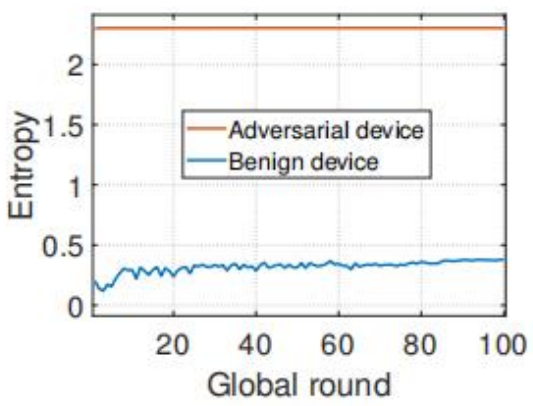
- Entropy-based filtering
  - **Public data** on server
  - **Filter out** high **entropy** models (loss)
  - For model poisoning



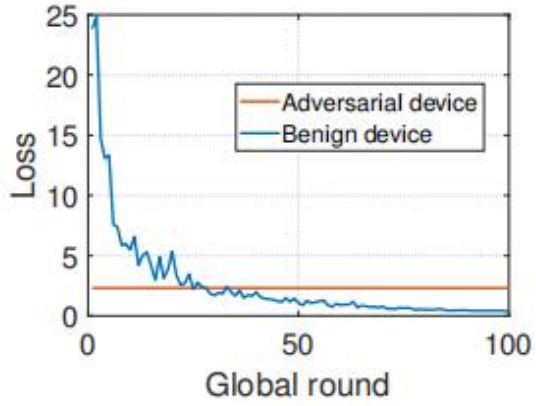
$$E_{x_{pub,j}}(k) = -\sum_{q=1}^{Q} P^{(q)}_{x_{pub,j}}(k) \log P^{(q)}_{x_{pub,j}}(k)$$

$$E(k) = \frac{1}{n_{pub}} \sum_{j=1}^{n_{pub}} E_{x_{pub,j}}(k)$$

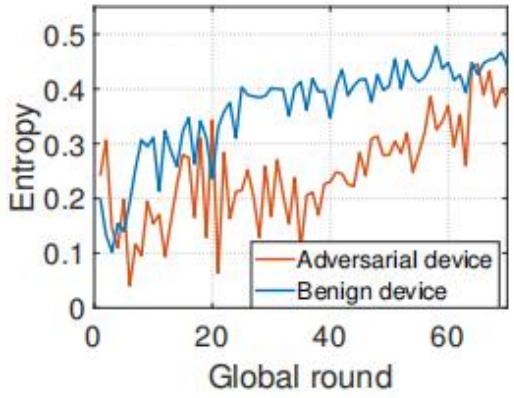
Shannon entropy



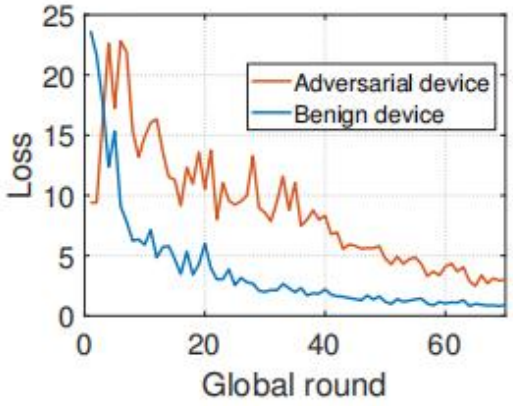
(a) Model poison, entropy



(b) Model poison, loss



(c) Data poison, entropy



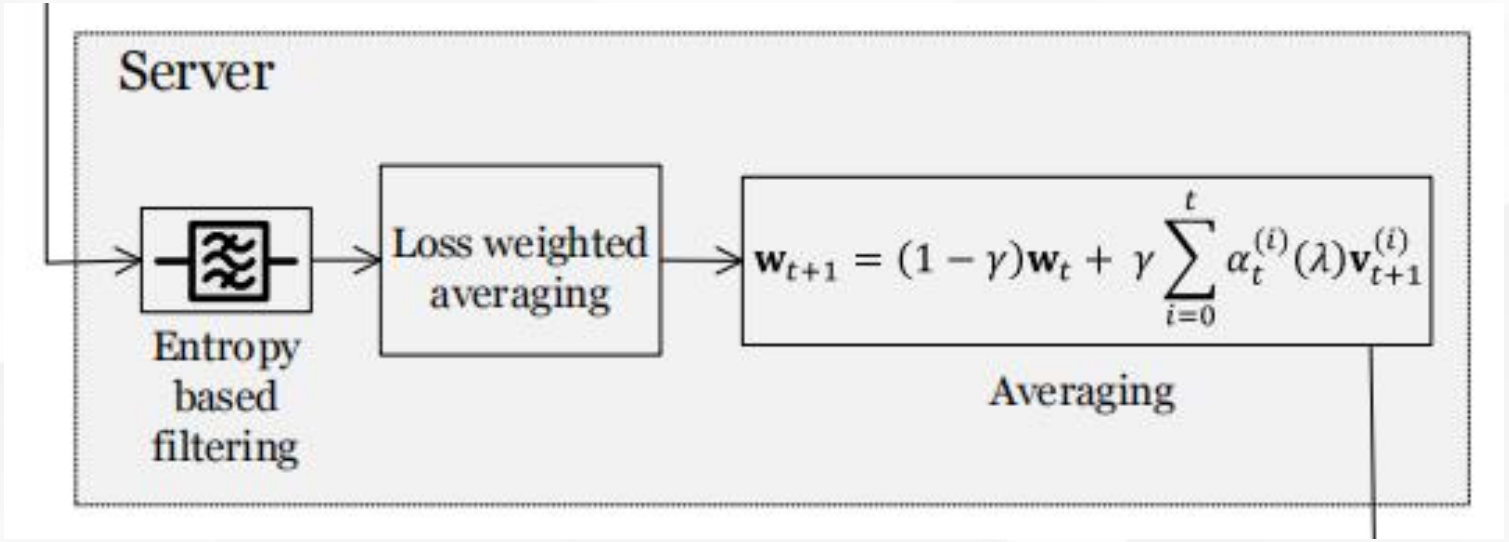
(d) Data poison, loss

Sageflow algorithm

- Loss-weighted averaging
  - Aggregation weight according to local models' **measured qualities**
  - Measure by **loss** on public data
  - data-poisoned model -> small weight + less impact
  - For data poisoning & scaled backdoor

$$\beta_t^{(k)}(\delta) \propto \frac{m_k}{\{F_{pub}(\mathbf{w}_t(k))\}^{\delta}} \quad \text{and} \quad \sum_{k \in S_t} \beta_t^{(k)}(\delta) = 1.$$

$$\mathbf{w}_{t+1} = \sum_{k \in S_t} \beta_t^{(k)}(\delta) \mathbf{w}_t(k)$$



Sageflow algorithm

- Time complexity

Public data: n_{pub} | Model parameters: $|w|$ | Clients number: K

$$\mathcal{O}(n_{pub}|w|K)$$

Algorithm 1 Proposed Sageflow Algorithm

Input: Initialized model $\mathbf{w}_0$, **Output:** Final global model $\mathbf{w}_T$

Process at the Server

1: **for** each global round $t = 0, 1, ..., T-1$ **do**
2: Choose S_t and send the current model and the global round $(\mathbf{w}_t, t)$ to the devices
3: Wait for T_d and then:
4: **for** $i = 0, 1, ..., t$ **do**
5: $U_t^{(i)}(E_{th}) = \{k \in U_t^{(i)} | E(k) < E_{th}\}$ // Entropy-based filtering in each group
6: $\mathbf{v}_{t+1}^{(i)} = \sum_{k \in U_t^{(i)}(E_{th})} \beta_t^{(k)}(\delta)\mathbf{w}_i(k)$ // Loss-weighted averaging in each group (with same staleness)
7: **end for**
8: $\mathbf{w}_{t+1} = (1-\gamma)\mathbf{w}_t + \gamma \sum_{i=0}^{t} \alpha_t^{(i)}(\lambda)\mathbf{v}_{t+1}^{(i)}$ // Averaging of representative models (with different staleness)
9: **end for**

Process at the Device: Device k receives $(\mathbf{w}_t, t)$ from the server and performs local updates to obtain $\mathbf{w}_t(k)$. Then each benign device k sends $(\mathbf{w}_t(k), t)$ to the server, while a malicious adversary sends a poisoned model depending on the type of attack.

Sageflow algorithm

- Theoretical Analysis
- Convergence analysis
  - Assumption 1: μ-strongly convex + L-smooth
  - Assumption 2: unbiased estimation
- Theoretical bound

$$F(x) \leq F(y) + \nabla F(x)^T (x-y) - \frac{\mu}{2}\|x-y\|^2$$

$$F(x) \geq F(y) + \nabla F(x)^T (x-y) - \frac{L}{2}\|x-y\|^2$$

$$\mathbb{E}\|\nabla F_k(\mathbf{w}_t^i(k), \xi_t^i(k)) - \nabla F(\mathbf{w}_t^i(k))\|^2 \leq \rho_1$$

$$\mathbb{E}[F(\mathbf{w}_T) - F(\mathbf{w}^*)] \leq \nu^T [F(\mathbf{w}_0) - F(\mathbf{w}^*)] + (1-\nu^T) Z(\lambda, E_{th}, \delta)$$

Convergence speed

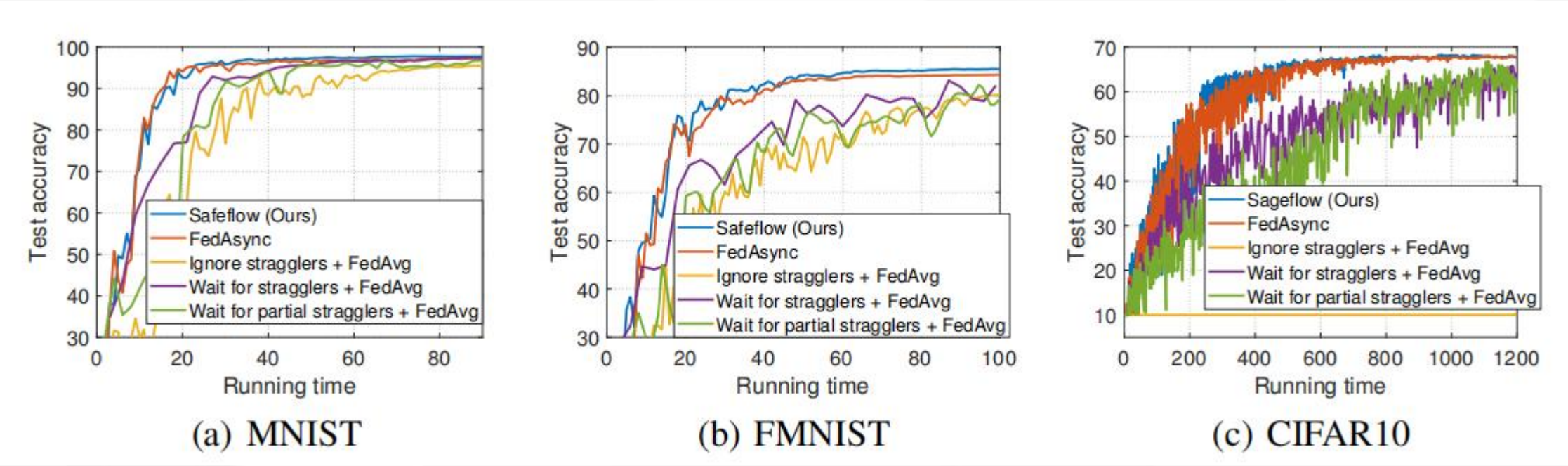
Error

Sageflow algorithm

- Datasets: MNIST, FMNIST, CIFAR10
  - 2% as public data
- Models: CNN(2conv+2fc), CNN(2conv+1fc), CNN(VGG-11)
  - ignore batchnorm
- FL setting: 100 clients, two classes for each client, 5 local epochs, batch size of 10

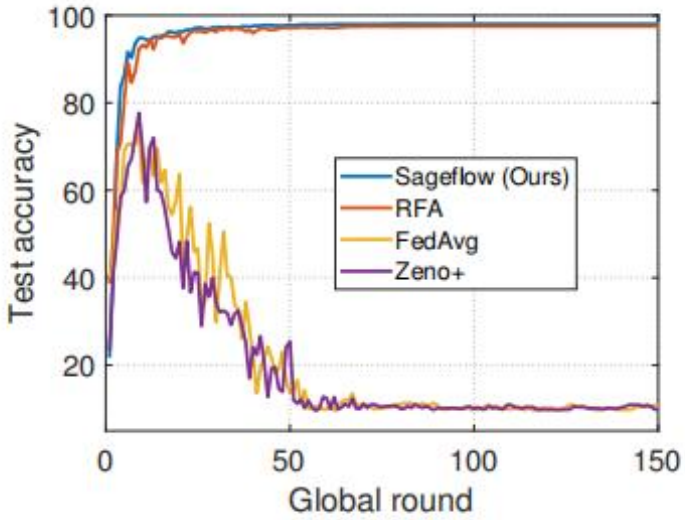
Sageflow algorithm

- Only stragglers: 10% participants
- Baselines: FedAvg(waiting, ignoring, waiting 50%), FedAsync
- Settings: uniform delay of [0,1,2] global rounds
- Ignoring lose significant data converges to a **suboptimal point**
- Waiting(all, 50%) requires the **largest running time** until convergence

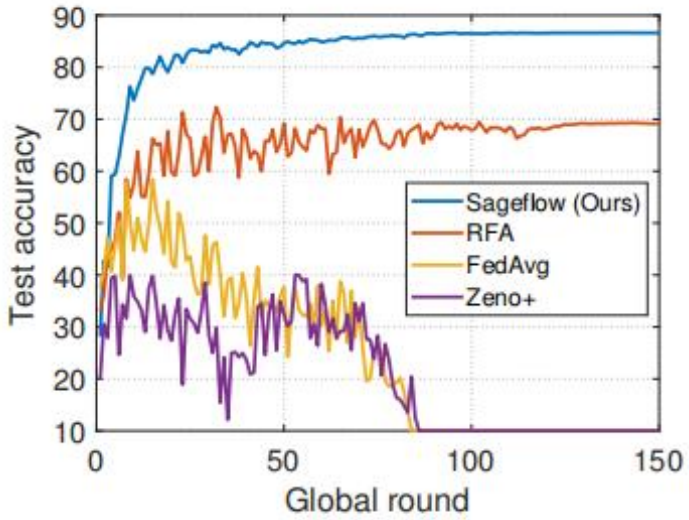


Sageflow algorithm

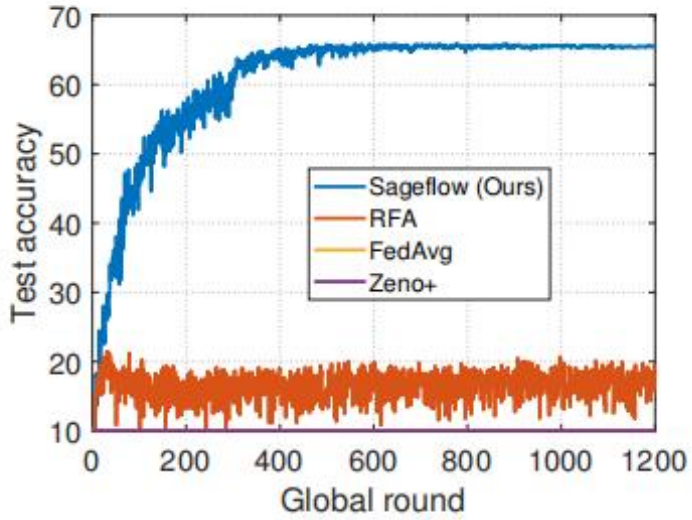
- Only adversaries: 20% participants
- Baselines: RFA, FedAvg, synchronized Zeno+, Multi-Krum
- Attacks: model(-0.1w), data(label-flipping), backdoor(model replacement, pixel-pattern attack)
- FedAvg **does not work well** on all datasets
- RFA: **complex model** led to worse performance
- Zeno+: bad on poisoning **but** good for backdoor
- Sageflow: **slow down** posioning



(a) MNIST



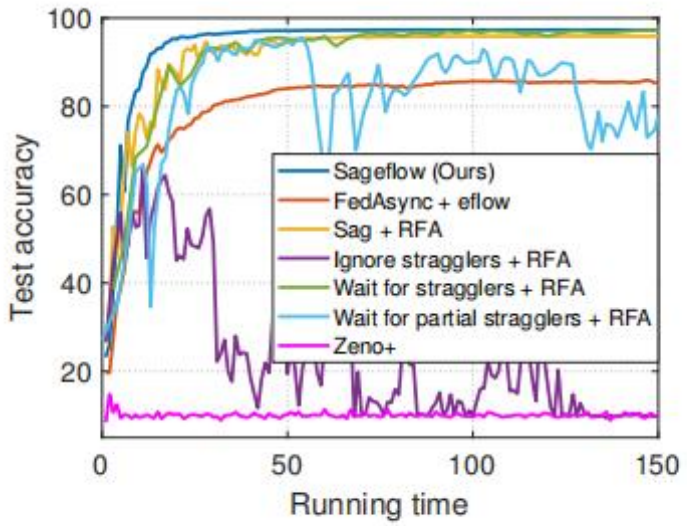
(b) FMNIST



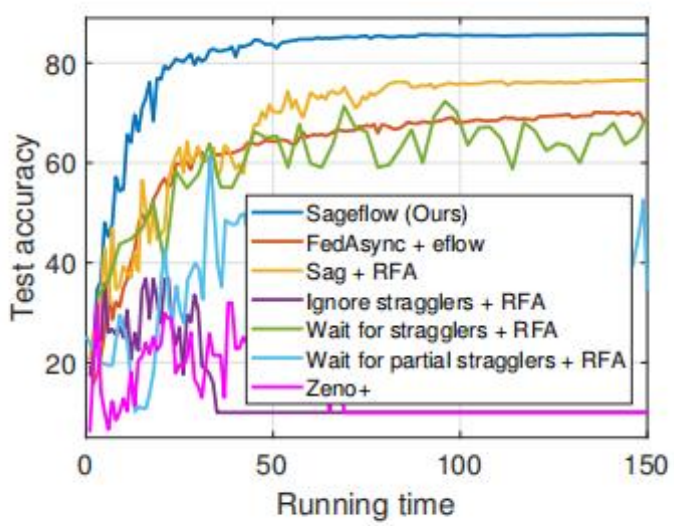
(c) CIFAR10

Sageflow algorithm

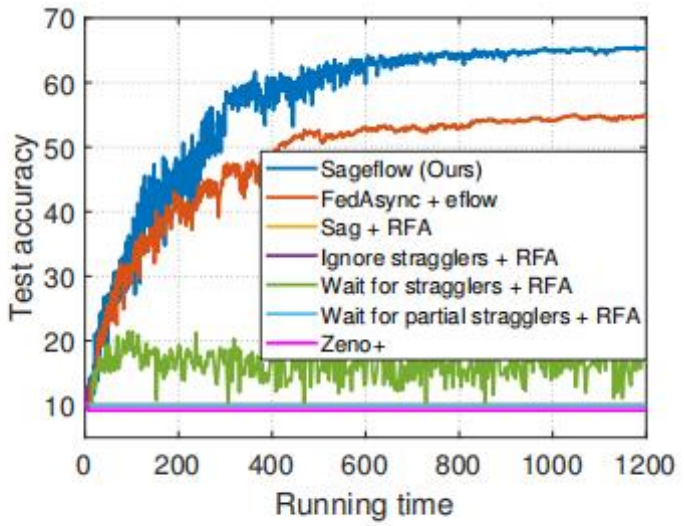
- Stragglers + adversaries: 20%(model/data), 10%(backdoor) participants
- Baselines: asynchronized Zeno+, Multi-Krum
- Zeno+: **does not perform well**(ignore staleness & entropy)
- Waiting + RFA: suffer from **straggler**
- Ignoring/Sag + RFA: poor(**high attack ratio**)
- eflow/RFA + FedAsync: poor(**one-by-one arrivals**)



(a) MNIST



(b) FMNIST



(c) CIFAR10

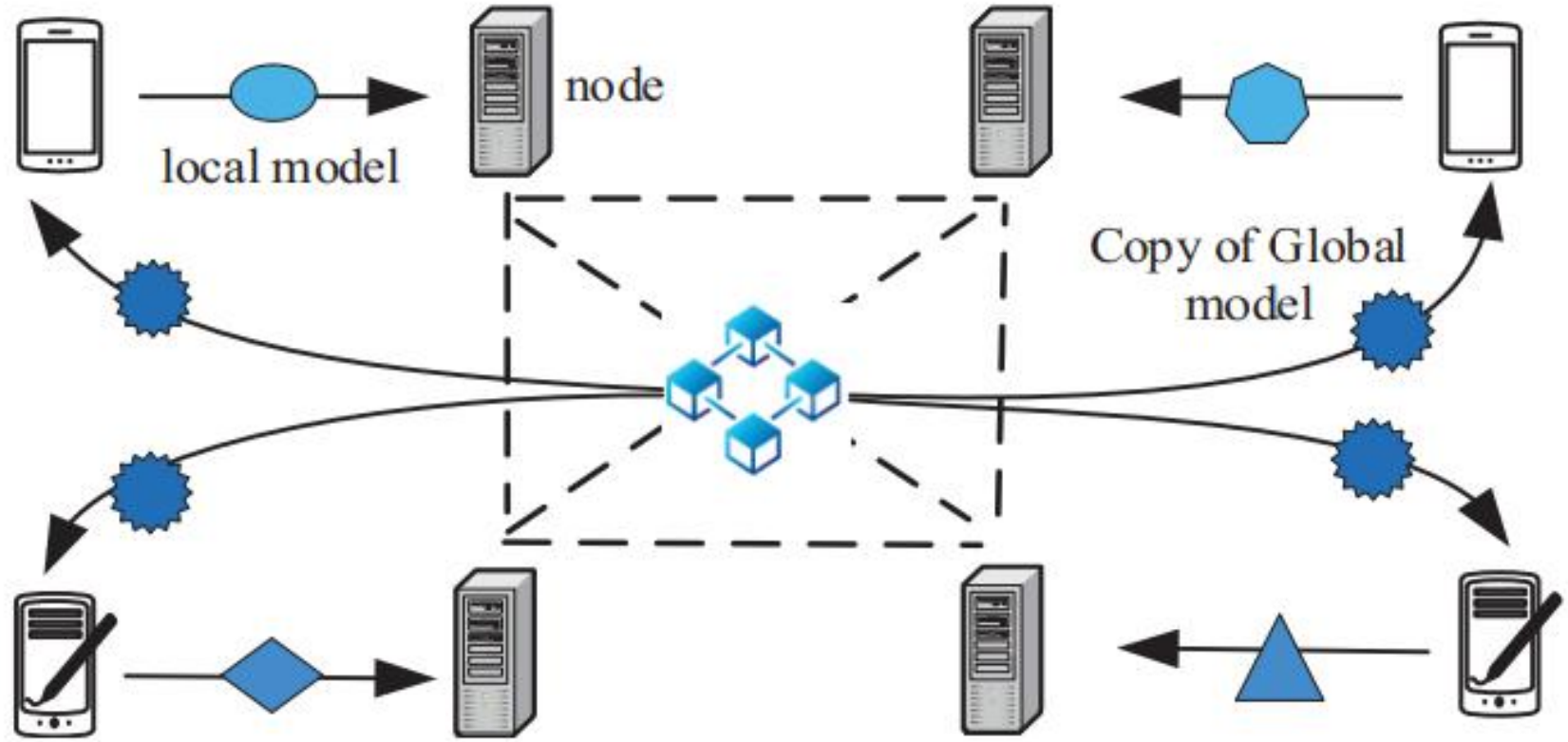
Sageflow algorithm

- Sageflow: robust FL scheme handle both **stragglers and adversaries**
  - staleness-aware grouping: stragglers
  - entropy-based filtering: model poisoning
  - loss-weighted averaging: data poisoning + backdoor
- Theoretical convergence analysis
- Extensive experimental results
- Future issues: Sageflow + secure aggregation

FLChain

FL + Blockchain

- complete dependency on the reliability of a central server for storage and computation of the global model update
- Any malicious activity leads to flawed global model update which is detrimental for accuracy of subsequent local model updates



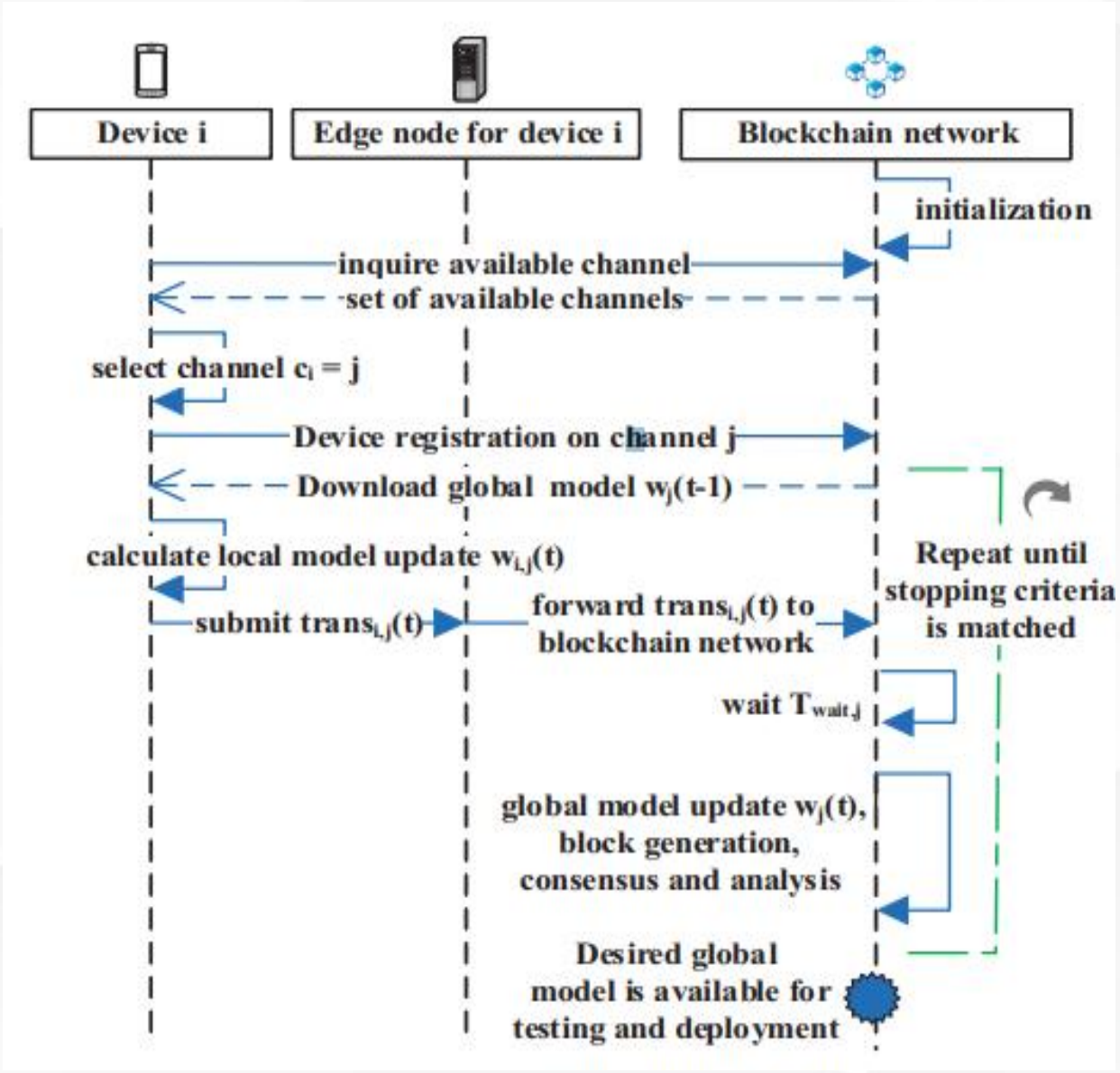
FL + Blockchain

- Initialization
- Channel Inquiry
- Channel Selection
- Device Registration
- Local Model Update
- Transaction Pool
- Global Model Update
- Consensus Protocol
- Analysis

Algorithm 1 : FLchain operation for channel j

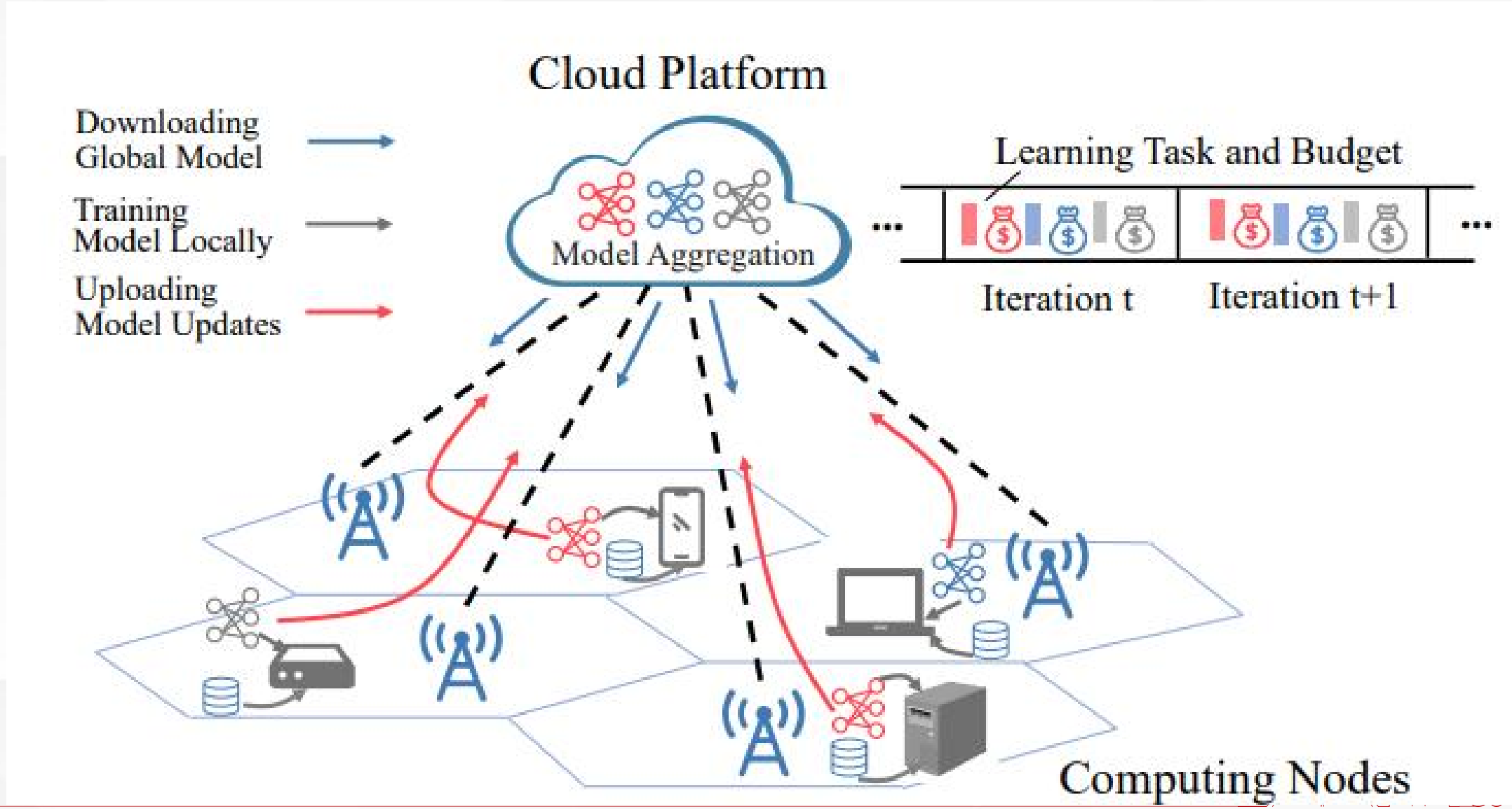
1: Setup channel j for global model M_j
2: initialization: $t = 0$; $w_j(0), w_{i,j}(0) \in [0, w_{j,max}]$;
3: **for all** $i \in D_j$ **do in parallel**
4: Inquire available channels
5: Select channel $c_i = j \in C \triangleq \{1, 2, 3, \ldots, C_n\}$
6: Register device i to channel j
7: **end for**
8: **while** $\|w_j(t) - w_j(t-1)\|_2 \leq \varepsilon_{threshold,j}$ **do**
9: **for all** $i \in D_j$ **do in parallel**
10: Download $w_j(t)$ from channel j to device i
11: $t \leftarrow t + 1$; $w^0_{i,j}(t) = w_j(t)$;
12: **for** $v = 1, ..., V$ **do**
13: $w^v_{i,j}(t) = w^{v-1}_{i,j}(t) - \eta \nabla \Phi$, and Eq. (6)
14: **end for**
15: $w_{i,j}(t) = w^V_{i,j}(t)$, Generate $trans_{i,j}(t)$ and forward to blockchain network
16: Wait for notification from channel j
17: **end for**
18: Calculate $w_j(t)$ using Eq. (7)
19: global model state trie updation, block generation and consensus
20: **end while**

FL + Blockchain



FAIR

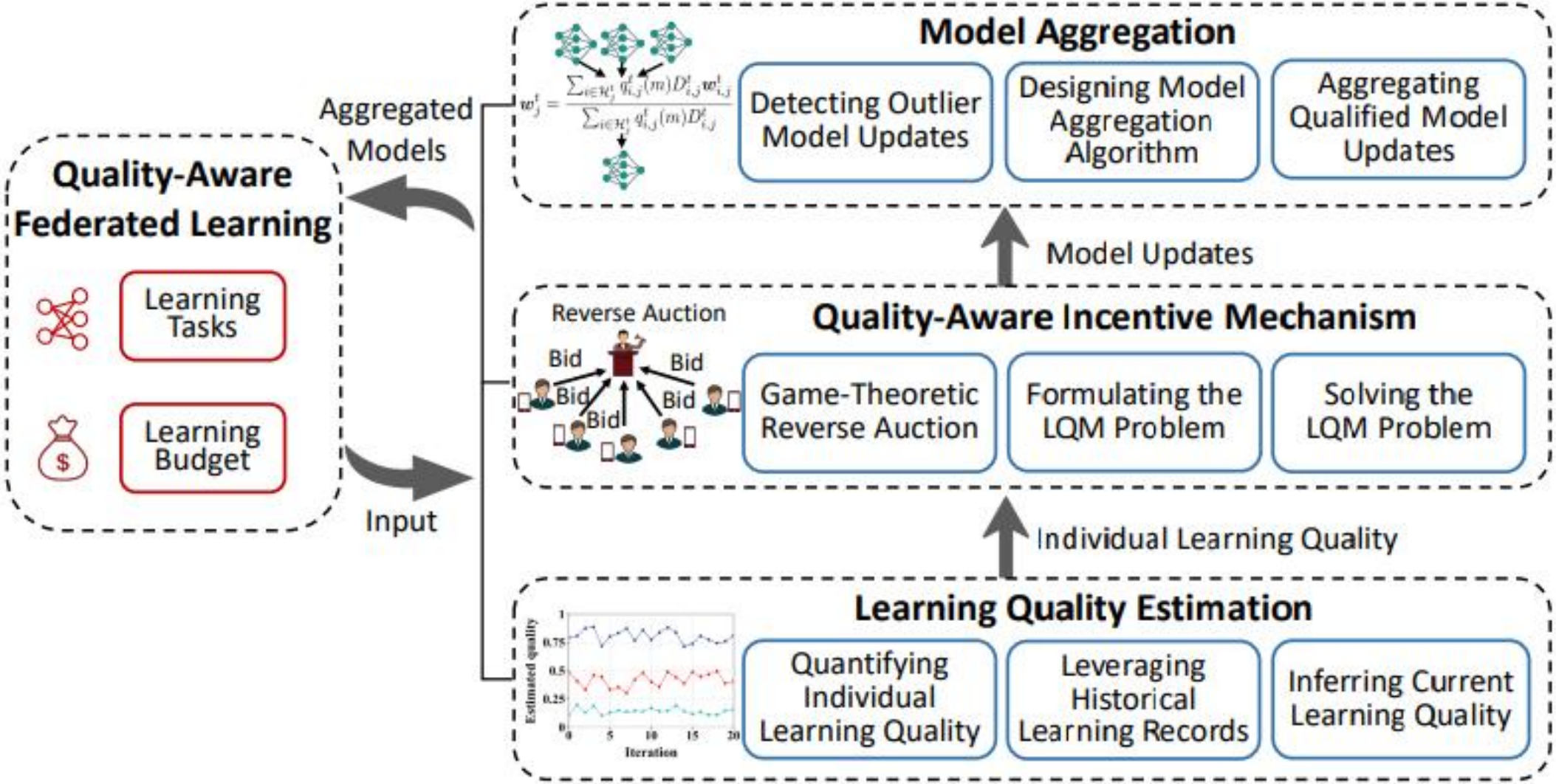
Federated Learning



Federated Learning

- Problems
  - mobile users do not participate in learning due to computation and energy consumption
  - model update quality of mobile devices can vary dramatically, inclusively aggregating low-quality model updates can deteriorate the global model quality
- Solution: Federated leArning with quallty awaReness
  - learning quality estimation: leverage historical learning records to estimate the user learning quality, the exponential forgetting function is utilized for weight assignment;
  - quality-aware incentive mechanism: within the recruiting budget, model a reverse auction problem to encourage the participation of high-quality learning users;
  - model aggregation: integrates the model quality into aggregation and filters out non-ideal model updates, to further optimize the global learning model

FAIR Architecture



FAIR Architecture

Estimating Learning Quality

$$m_{i,j}^t = loss_j(t_s) - loss_{i,j}(t_e).$$

$$q_{i,j}^t = \begin{cases} m_{i,j}^t D_{i,j}^t, & \text{if } t_s < t_{i,j}^u \leq t_e; \\ 0, & \text{otherwise.} \end{cases}$$

$$\widehat{q}_{i,j}^t = \frac{\sum_{k=0}^{r} \rho^{t_r - t_k} q_{i,j}^{t_k}}{\sum_{k=0}^{r} \rho^{t_r - t_k}}.$$

FAIR Architecture

Quality-Aware Incentive Mechanism

Algorithm 1: Solving the LQM problem.

Input : (1) bid price $b_{i,j}^t$; (2) budget B_j^t; (3) task set $\mathcal{L}_i^t$; (4) quality estimation values $\widehat{q}_{i,j}^t$.

Output : (1) the task allocation results $s_{i,j}^t$; (2) the payments $r_{i,j}^t$.

Initialize $\mathcal{N}_j^t \leftarrow \emptyset$, $p_j^t \leftarrow 0$ **for each** $l_j^t \in \mathcal{L}^t$;
Initialize $x_i^t \leftarrow 1$ **for each** $i \in \mathcal{N}$;
Initialize $r_{i,j}^t \leftarrow 0, s_{i,j}^t \leftarrow 0$ **for each** $i \in \mathcal{N}, l_j^t \in \mathcal{L}^t$;
foreach $i \in \mathcal{N}$ **do**
  foreach $l_j^t \in \mathcal{L}_i^t$ **do**
    $\mathcal{N}_j^t \leftarrow \mathcal{N}_j^t + \{i\}$;
  end
end
while $\exists\ x_i^t = 0$ ***and*** $\exists\ p_j^t = 0$ **do**
  Initialize $\mathcal{M}_j^t \leftarrow \emptyset$ **for each** $l_j^t \in \mathcal{L}^t$;
  foreach $l_j^t \in \mathcal{L}^t$ **do**
    if $p_j^t = 0$ **then**
      Sort all $i \in \mathcal{N}_j^t$ in descending order of $\frac{q_{i,j}^t}{b_{i,j}^t}$;
      Find the smallest k such that $\sum_{i=1}^{k} \frac{b_{k,j}^t}{q_{k,j}^t} q_{i,j}^t x_i^t > B_j^t$;
      for $i \leftarrow 1$ **to** $k-1$ **do**
        $\mathcal{M}_j^t \leftarrow \mathcal{M}_j^t + \{i\}$;
        $r_{i,j}^t \leftarrow \frac{b_{k,j}^t}{q_{k,j}^t} q_{i,j}^t$;
      end
      end
  end
  Find the task l_k^t with maximum $\sum_{i \in \mathcal{M}_k^t} q_{i,k}^t x_i^t$;
  Set $p_k^t \leftarrow 1$;
  foreach $i \in \mathcal{M}_k^t$ **do**
    if $x_i^t = 1$ **then**
      $s_{i,k}^t \leftarrow 1$;
      $x_i^t \leftarrow 0$;
    end
  end
end
return $(s_{i,j}^t, r_{i,j}^t)$;

The defined LQM problem can be formulated as

$$\max_{\mathcal{M}_j^t, \mathcal{R}_j^t} \quad \sum_{l_j^t \in \mathcal{L}^t} \sum_{i \in \mathcal{M}_j^t} \widehat{q}_{i,j}^t, \tag{10}$$

$$s.t. \quad \sum_{i \in \mathcal{N}} r_{i,j}^t s_{i,j}^t \le B_j^t, \quad \forall l_j^t \in \mathcal{L}^t, \tag{11}$$

$$r_{i,j}^t \ge b_{i,j}^t, \quad \forall l_j^t \in \mathcal{L}^t, \forall i \in \mathcal{M}_j^t, \tag{12}$$

$$s_{i,j}^t \in \{0,1\}, \quad \forall i \in \mathcal{N}, \forall l_j^t \in \mathcal{L}^t, \tag{13}$$

$$s_{i,j}^t = 0, \quad \forall l_j^t \notin \mathcal{L}_i^t, \forall i \in \mathcal{N}, \tag{14}$$

$$\sum_{l_j^t \in \mathcal{L}^t} s_{i,j}^t \le 1, \quad \forall i \in \mathcal{N}. \tag{15}$$

FAIR Architecture

Model Aggregation

$$\boldsymbol{w}_j^t = \frac{\sum_{i=1}^{N} D_{i,j}^t \boldsymbol{w}_{i,j}^t}{\sum_{i=1}^{N} D_{i,j}^t},$$

$$\boldsymbol{w}_j^t = \frac{\sum_{i \in \mathcal{M}_j^t} m_{i,j}^t D_{i,j}^t \boldsymbol{w}_{i,j}^t}{\sum_{i \in \mathcal{M}_j^t} m_{i,j}^t D_{i,j}^t},$$

Algorithm 2: Model aggregation algorithm.

Input : (1) winner node set $\mathcal{M}_j^t$; (2) local model parameters $\boldsymbol{w}_{i,j}^t$; (3) data size used for training $D_{i,j}^t$; (4) data quality $m_{i,j}^t$.

Output : aggregated model parameters $\boldsymbol{w}_j^t$.

Initialize high-quality node set $\mathcal{H}_j^t \leftarrow \{i : i \in \mathcal{M}_j^t\}$;
Set $D \leftarrow \sum_{i \in \mathcal{M}_j^t} m_{i,j}^t D_{i,j}^t$;
Set $\boldsymbol{w}_j^t \leftarrow \sum_{i \in \mathcal{M}_j^t} \frac{m_{i,j}^t D_{i,j}^t}{D} \boldsymbol{w}_{i,j}^t$;
foreach $i \in \mathcal{M}_j^t$ **do**
    Compute $d_{i,j}^t \leftarrow similarity(\boldsymbol{w}_j^t, \boldsymbol{w}_{i,j}^t)$;
end
Compute $\bar{\mu}_d$, $\hat{\mu}_d$, σ_d;
if $\bar{\mu}_d > \hat{\mu}_d$ **then**
    foreach $i \in \mathcal{H}_j^t$ **do**
        if $d_{i,j}^t > \hat{\mu}_d + \eta\sigma_d$ **then**
            $\mathcal{H}_j^t \leftarrow \mathcal{H}_j^t - \{i\}$;
        end
    end
else
    foreach $i \in \mathcal{H}_j^t$ **do**
        if $d_{i,j}^t < \hat{\mu}_d - \eta\sigma_d$ **then**
            $\mathcal{H}_j^t \leftarrow \mathcal{H}_j^t - \{i\}$;
        end
    end
end
Set $D \leftarrow \sum_{i \in \mathcal{H}_j^t} m_{i,j}^t D_{i,j}^t$;
Set $\boldsymbol{w}_j^t \leftarrow \sum_{i \in \mathcal{H}_j^t} \frac{m_{i,j}^t D_{i,j}^t}{D} \boldsymbol{w}_{i,j}^t$;
return $\boldsymbol{w}_j^t$;

Analysis

- Theoretical
  - Truthful
  - Individually rational
  - Computational Efficiency

Truthfulness 每一个报价都必须是真实的，等于自身的cost，如果不等于，则utility会变小

Individual Rationality 每一轮迭代的utility都大于等于0

Computational Efficiency 所有算法的执行时间为多项式时间

Evaluation

- Settings
  - Models: MLP, LeNet, MobileNet, VGG-11, EfficientNet-B0, ResNet-18
  - Dataset: MNIST, Fashion-MNIST, CIFAR-10, Street View House Numbers
  - Baselines: Theoretically optimal mechanism, Knapsack greedy mechanism, Bid price first mechanism
  - Hyperparameters

Settings	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{L}^t\rvert$	$b_{i,j}^t$	$D_{i,j}^t$	N_e	R_e	B_j^t
I	20	1	[1,3]	[1000,3000]	5	[0.05,0.5]	5
II	20	1	[1,3]	5000	[0,20]	0.5	10
III	100	4	[1,10]	[1000,10000]	30	[0,1]	[10,30]

- **Theoretically optimal mechanism:** It adopts the depth-first search approach to find the theoretically optimal solution for the LQM problem, which however cannot guarantee the truthfulness of nodes.
- **Knapsack greedy mechanism:** It greedily selects winner nodes based on the amount of data used for training divided by the bid price, i.e., $D_{i,j}^t/b_{i,j}^t$, where the data quality and truthfulness of nodes are not considered.
- **Bid price first mechanism:** It preferentially selects nodes with the lowest bid price without guaranteeing the truthfulness of nodes.

Evaluation

- Different incentive mechanisms

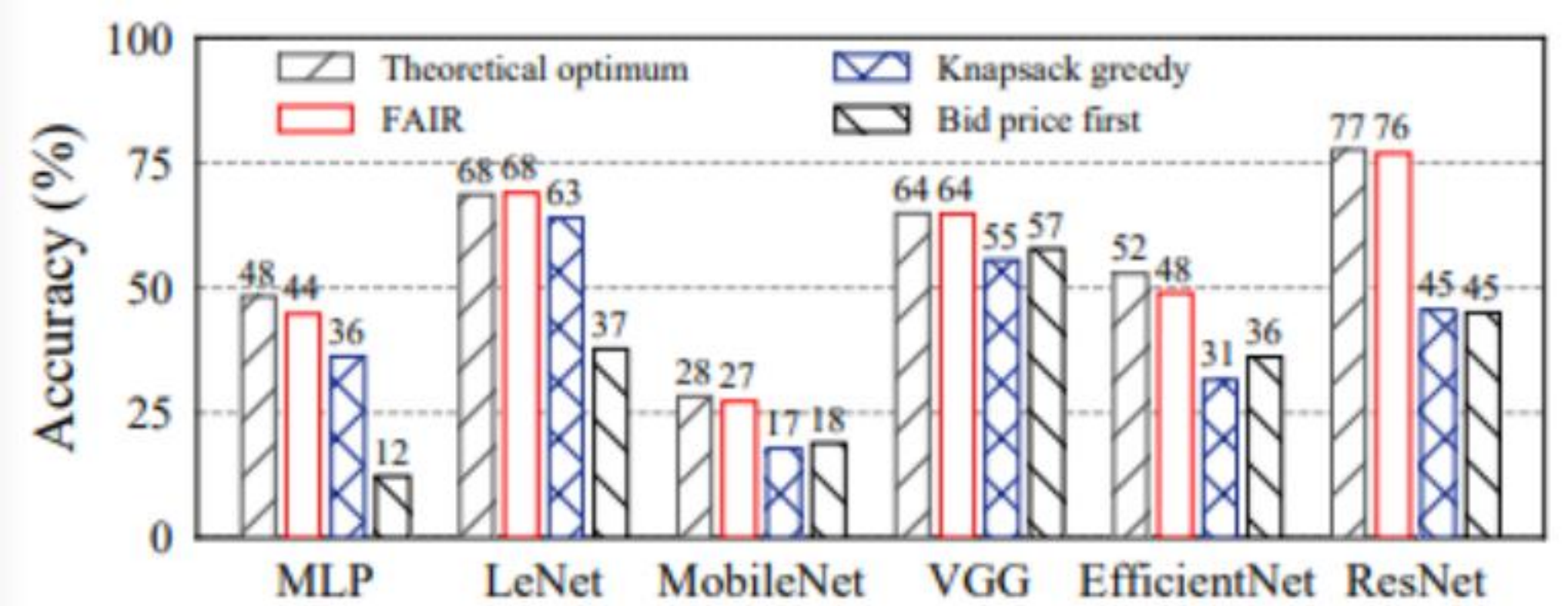


Fig. 4. The average model accuracy with different incentive mechanisms.

Evaluation

- Attack scenarios

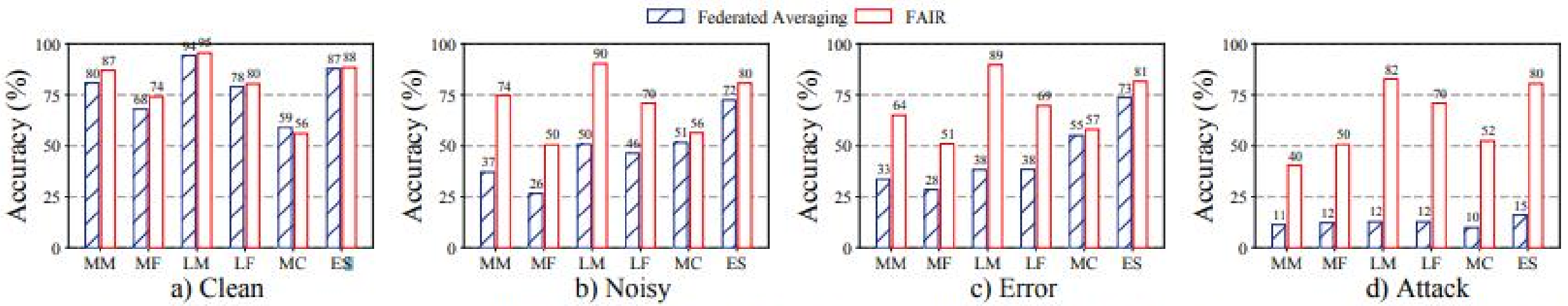


Fig. 3. The model aggregation performance of MLP-MNIST (MM), MLP-FMNIST (MF), LeNet-MNIST (LM), LeNet-FMNIST (LF), MobileNet-CIFAR10 (MC), EfficientNet-SVHN (ES) under different scenarios: a) Clean datasets; b) Noisy label datasets; c) Error label datasets; d) Attack datasets.

Evaluation

- Noise levels

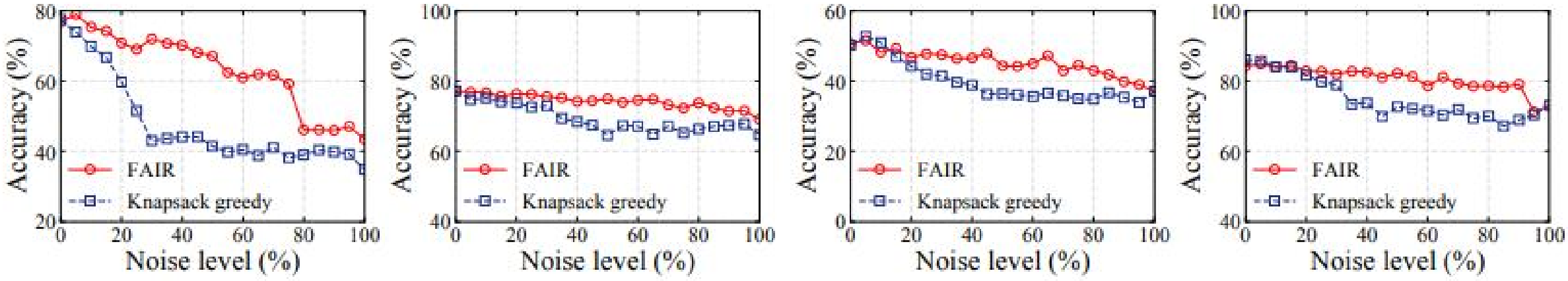


Fig. 5. The model accuracy vs. noise levels.

Evaluation

- Learning budgets

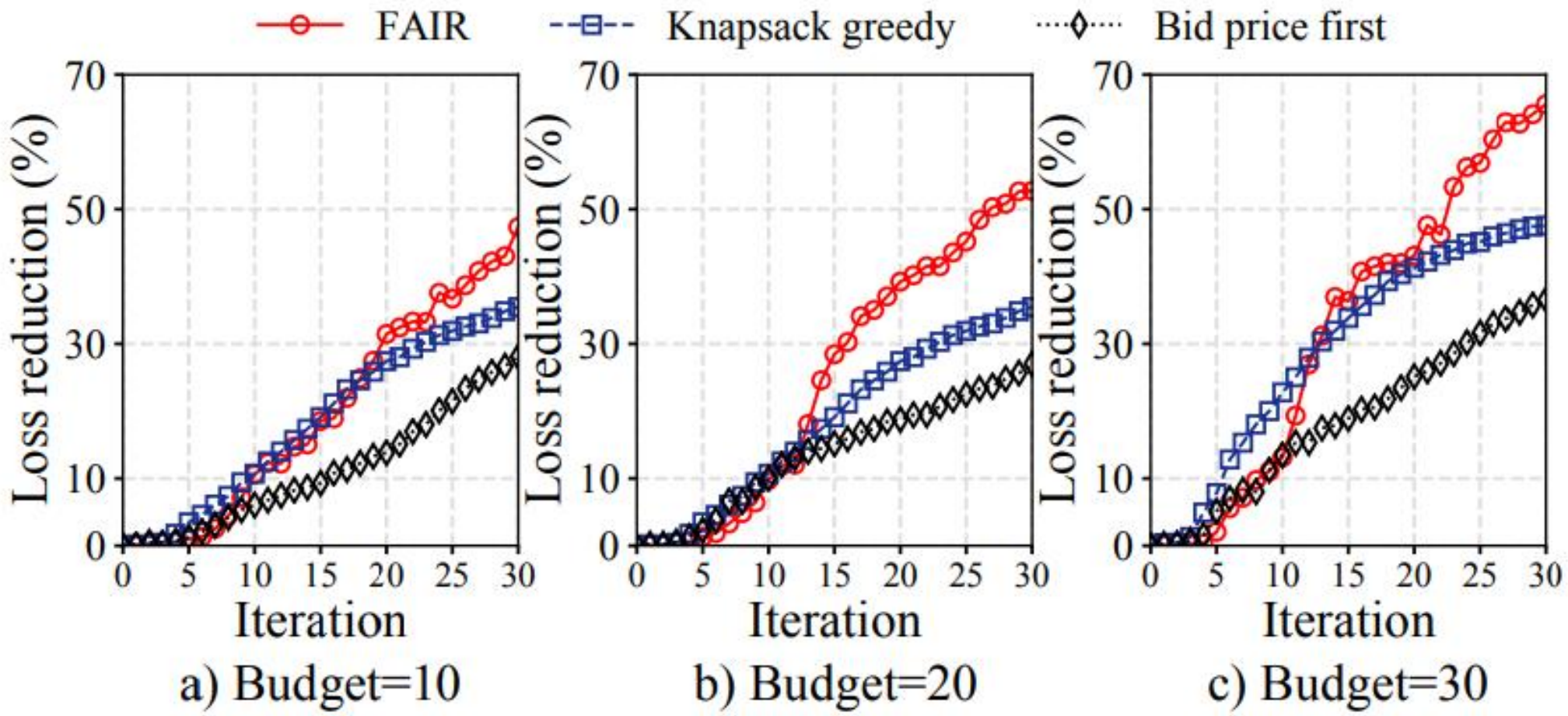


Fig. 6. The average test loss reduction of learning models vs. learning budgets.

Q&A

饮 水 思 源　爱 国 荣 校