A Novel Federated Learning Scheme for Generative Adversarial Networks

Jiaxin Zhang, Liang Zhao, *Member, IEEE,* Keping Yu, Geyong Min,
Ahmed Y. Al-Dubai, *Senior Member, IEEE,* and Albert Y. Zomaya, *Fellow, IEEE*

Abstract—Generative adversarial networks (GANs) have been advancing and gaining tremendous interests from both academia and industry. With the development of wireless technologies, a huge amount of data generated at the network edge provides an unprecedented opportunity to develop GANs applications. However, due to the constraints such as bandwidth, privacy, and legal issues, it is inappropriate to collect and send all data to the cloud or servers for analysis, training, and mining. Thus, deploying and training GANs at the edge becomes a promising alternative solution. The instability of GANs introduced by non-independent and identical data (Non-IID) poses significant challenges to training GANs. To address these challenges, this paper presents a novel federated learning framework for GANs, namely, Collaborated gAme Parallel Learning (CAP). CAP supports parallel training of data and models for GANs, breaking the isolated training among generators that exists in the previous distributed algorithms, and achieving collaborative learning among cloud, edge servers, and devices. Then, to further enhance the ability of CAP-GAN for addressing Non-IID issues, we propose a Mix-Generator module (Mix-G) which divides a generator into the sharing layer and personalizing layer. The Mix-G module extracts the generic and personalization features and improves the performance of CAP-GAN on extremely personalizing datasets. Experimental results and analysis substantiate the usefulness and superiority of our proposed CAP-GAN scheme which can achieve better results in the Non-IID scenarios compared with the state-of-the-art algorithms.

Index Terms—Generative adversarial networks, distributed learning and deployment, collaborative learning, non-independent and identical data.

1 Introduction

Generative Adversarial Networks (GANs), as a generative architecture model, can generate new data by learning the probability distribution of a given realistic dataset. GANs have attracted significant research attentions since being introduced by Goodfellow *et al.* [1]. GANs have been applied to various fields of generative tasks, such as image generation, video generation, object detection, and image super-resolution. Furthermore, GANs have the potential to be applied to support edge applications. For instance, GANs have been used in vehicle networking to predict the vehicle trajectories [2] and assist reinforcement learning networks to optimize transmission scheduling [3]. Studies on GAN-related applications have shown an explosive growth trend every year, while GANs have gained cross-domain attention for their great capacity in data generation, security enhancement, and deep representation learning.

As a result of the growing amount of data at the edge, traditional cloud-based machine learning training frameworks are facing significant challenges. However, deep learning (for example, GANs) relies on a large amount of data to achieve impressive performance. The data is distributed across the edge clients and causes privacy issues. Thus, gathering users' data at the edge for training is becoming impossible, which creates challenges for GANs applications. Federated Learning (FL) [4] is a decentralized learning algorithm for general machine learning. Basically, FL is a model-agnostic algorithm that enables training models on devices and transferring model parameters to the cloud or edge in order to reduce communication and privacy breaches, such as FedGAN [5] and FeGAN [6].

However, it requires a significant amount of computation and energy resources in training and inferencing to complete GANs (including generator and discriminator) on edge devices. Therefore, Federated Stochastic Gradient Descend-based GANs are proposed to solve this problem, and MD-GAN [7] is the most representative algorithm. MD-GAN divides the GAN into two pieces: the generator that runs on servers, and the discriminator that is deployed on devices. This reduces the computational burden required by a single device. MD-GAN relies on transporting gradient to train the split GAN and uses a single-server-multi-device design, which makes large-scale deployment unfeasible due to the computational and bandwidth limitations of a single server. To handle this problem, AC-GAN [8] scales MD-GAN to a multi-server-multi-device architecture and deploys GANs at the edge, relieving the strain on a single server. This architecture disperses the bandwidth and computation. Even AC-GAN uses a compression algorithm to further reduce the amount of communication workload, which makes it possible to deploy GANs on large-scale devices. Although the contributions of MD-GAN and AC-GAN on decentralized training of GANs are significant, they

Jiaxin Zhang and Liang Zhao are with the School of Computer Science, Shenyang Aerospace University, Shenyang 110136, China. (e-mail: jacazjx@foxmail.com, lzhao@sau.edu.cn).
Keping Yu is with Graduate School of Science and Engineering, Hosei University, Tokyo 184-8584, Japan. (e-mail: keping.yu@ieee.org).
Geyong Min is with the Department of Computer Science, University of Exeter, UK. (e-mail: g.min@exeter.ac.uk).
Ahmed Y.Al-Dubai is with the School of Computing, Edinburgh Napier University, UK. (e-mail: a.al-dubai@napier.ac.uk).
Albert Y. Zomaya is with the School of Computer Science, University of Sydney, Australia (e-mail: albert.zomaya@sydney.edu.au).
Liang Zhao is the corresponding author.

ignore the following multifaceted challenges when being deployed practically.

- **Energy bottleneck.** Energy is one of the precious resources for edge devices. Signal search and transmission are so energy-intensive that mobile users are reluctant to share data and models with their surroundings. Even if the devices around these users are often sparse and uncertain (due to mobility), it is not wise to keep the signal searching and wait to connect all the time. Thus, when deploying GANs in a cross-regional scenario, the methods like MD-GAN and AC-GAN that adopts device-to-device links to share discriminator will not work.
- **Limited coverage.** A single edge server (ES) covering a limited area means that there are not enough devices available for training a feasible GAN. Since geo-distributed edge users lack effective information exchange, each ES forms an **Information Island Phenomenon**, which leads to inconsistent generation capability of generators in each ES. In particular, generators with less data converge much slower than those with more data due to insufficient samples.
- **Non-independent and identical data (Non-IID).** Non-IID across devices and regions can lead to biased datasets, resulting in a biased GAN (e.g., mode collapse or overfitting). Consequently, it presents a challenge under such circumstances. In this work, we classify Non-IID in the generation task into two tiers: **Basic Non-IID** and **Fully Non-IID**. The distribution characteristic of Non-IID is the difference in data volume and category distribution between devices. Basic Non-IID permits categories to overlap among devices, while Fully Non-IID does not. Despite this observation, existing work only considers Non-IID in FeGAN, but this FL-based GAN is not suitable for devices with energy and computational restrictions. On the other hand, the state-of-the-art (SOTA) variations of FedSGD-GAN, such as MD-GAN and AC-GAN, do not address this issue.

This paper is dedicated to exploring the means of enhancing the computational support of GANs at the edge. Specifically, we investigate training and deploying GANs on multiple edge servers that have diverse clients. To address the aforementioned challenges, a novel distributed algorithm for training GANs in edge networks is proposed. The major contributions of this paper are as follows.

1) This paper investigates Federated Learning for Generative Adversarial Networks in a challenging scenario including *geographical limitations* (i.e., devices are sparsely distributed in the system), *energy constraints* (i.e., the devices have limited computation and communication ability), and *Non-IID issue* (i.e., the dataset in the system is highly personalized).
2) We propose a novel framework, called **C**ollaborative g**A**me **P**arallel **L**earning (CAP), that contains two parts, device-to-edge (D2E) and edge-to-cloud (E2C). These two parts form a closed loop internally and externally through collaboration and provide a channel for information exchange between different devices in different regions. CAP avoids the island information phenomenon and keeps the iterative state of generators consistent in different base stations. Thus, a three-tier architecture of CAP is more scalable at the edge than other architectures of the SOTA distributed algorithms.
3) To address the basic Non-IID issue, we introduce a *synthesis score* to obtain the feedback aggregation weight. This strategy allows for each edge server to obtain more comprehensive gradient feedback, which takes into account the size difference in the amount of data between devices and the game level difference in the discriminator.
4) To further solve the fully Non-IID problem, **Mix-Generator** (Mix-G) is proposed based on CAP, which only revises the architecture of the generator in edge servers. With the support of the Mix-G module, the generator consists of a sharing layer and a personalizing layer. By utilizing the weight sharing algorithm, weight sharing between ESs enhances the performance of CAP on Non-IID.

The rest of this paper is organized as follows. The related work is introduced in Section 2. After that, we present the preliminaries and problem setup in Section 3. The details of CAP are presented in Section 4, and the Mix-G module is presented in Section 5. Section 6 presents the experiments and compares the results with MNIST, Fashion-MNIST, and 2D mixture-Gaussian datasets. Finally, we provide the discussion in Section 7 and conclusions in Section 8.

2 Related Work

Training GANs with distributed edge servers is a recent emerging technique. In this work, we first introduce the background and related work of training GANs in a traditional centralized environment. Subsequently, we present the related work of distributed learning. In the end, we highlight recent researches on decentralized GAN training.

GAN was first proposed by Goodfellow *et al.* [1], who realized the application of game theory to neural networks and used two neural network games to learn the distribution of datasets to implement generative tasks. GAN has been widely applied in various fields, such as image analysis [9], object detection [10], medicine [11, 12], Internet of vehicle [2] and so on, due to its great performance of generative data. However, these algorithms are only suitable for centralized training using a centralized dataset [13–16].

Federated Learning-based GAN: To address the challenges, decentralized training algorithms have emerged recently. McMahan *et al.* [4] proposed a general algorithm, Federated Learning (FL). FedMes [17], proposed by Han *et al.*, extends FL to Mobile Edge Computing (MEC). With the popularity of FL, some algorithms were proposed to train GANs distributively using FL. FedGAN [5] and FeGAN [6] both extend FL to GAN. FedGAN uses two time-scale learning rates for two components of GAN, and periodically aggregates via an intermediary (parameter server). FeGAN, on the other hand, determines the aggregation weight to address Non-IID issues on distributed GAN by calculating the Kullback-Leibler distance between the local label categories and the global label categories. Despite the effectiveness of these FL-based approaches, they ignore the fundamental challenge of the computational burden of GANs on edge devices. GANs are, on average, twice the size of typical models

TABLE 1
The comparison of major distributed GAN.

Algorithms	Category	MEC Support	Aggregate algorithm	Label-Irrelevant	Non-IID Support	Metric
FedGAN[5]	Federated Learning-based GAN	-	**Weighted Average**	✓	✓	FID
FeGAN [6]	Federated Learning-based GAN	-	**Weighted Average**	-	✓	FID
MD-GAN [7]	Federated SGD-based GAN	-	Complete average	✓	-	MNIST SCORE (IS), FID
AC-GAN [8]	Federated SGD-based GAN	✓	Complete average	✓	-	IS, FID
CAP-GAN (ours)	Federated SGD-based GAN	✓	**Weighted Average**	✓	✓	MNIST SCORE, FID, MMD, MODE SCORE

(with both a generator and discriminator). Hence, training these models requires more energy and computation.

Federated SGD-based GAN: To address the above challenges, MD-GAN [7], proposed by Hardy *et al.*, adopts a method different from the previous FL approaches, federated stochastic gradient descent [4]. It has a single generator hosted by a parameter server and makes each client host a single discriminator. As it reduces about half the computational load of clients, this solution (FedSGD-based GAN) is eye-catching and has the potential to outperform FL-based GAN. AC-GAN, a more recent technique proposed by Zhang *et al.* [8], is the SOTA algorithm which follows the MD-GAN architecture and extends it to Mobile Edge Computing. AC-GAN addresses communication constraints by deploying multiple generators at the edge and reducing communication losses through compression algorithms. However, these two existing algorithms have three issues: 1) Both use **Gossip** algorithm [18] to make devices share their models with others. 2) Both pay no attention to the Non-IID issue. 3) Although AC-GAN supports MEC, its architecture is unsuitable for larger-scope deployment.

To further clarify the differences between CAP-GAN and other major distributed GANs, Table 1 shows the comparison in terms of five dimensions, including Category, MEC Support (i.e., whether the algorithm supports MEC or not), Aggregate algorithm (complete average or weighted average), Label-Irrelevant (i.e., whether the algorithm needs labels or not), Non-IID Support and Metric.

3 PRELIMINARIES

Before presenting our algorithm, in this Section, we introduce the detail of the SOTA algorithms including FL-GAN, MD-GAN, and AC-GAN, and define the problem setup.

3.1 Generative Adversarial Network

The two components of standard GAN, including a discriminator $\mathcal{D}$ and a generator $\mathcal{G}$, play a *min-max two-player* game, which is usually implemented by neural networks. During the game process, they have different goals. The generator $\mathcal{G}$ is a probability distribution generating function that learns to transform the random noise z to a distribution close to the real. The discriminator $\mathcal{D}$ is a classifier that aims at categorizing the input data into real or fake. The minimax game follows the objective function $V(G, D)$:

$$\begin{aligned} \min_G \max_D V(G,D) = &\mathbb{E}_{x\sim p_d}[\log(D_\omega(x))] \\ &+ \mathbb{E}_{z\sim p_z}[\log(1 - D_\omega(G_\theta(z)))], \end{aligned} \tag{1}$$

where z is the noise input of the generator and θ and ω are the parameters of the generator and the discriminator model.

To enable the algorithm to be used on any generic GAN model, we formulate the formula in the following general format:

$$\begin{aligned} \min_G \max_D V(G,D) = &\mathbb{E}_{x\sim p_d}[\mathcal{L}(D(x))] \\ &+ \mathbb{E}_{x\sim p_g}[\mathcal{L}(1 - D(x))], \end{aligned} \tag{2}$$

where x is a sample from the mixture dataset that consists of the generated distribution p_g and the raw distribution p_d (where $p_g \sim \mathcal{G}(z)$, and x is used for unified representation, thus $x \sim G(z)$ and $x \sim p_g$). $\mathcal{L}(\cdot)$ is some concave, increasing function. By replacing the $\mathcal{L}(\cdot)$, Eq. (2) can be various GAN variants. For instance, when $\mathcal{L}(t) = \log(t)$, it is equal to Eq. (1). When $\mathcal{L}(t) = \|\|^2$, it changes to LSGAN [19]. Specially, we use the previous one in our work. As Goodfellow *et al.* [1] proved the *Nash equilibrium*, the optimal $\mathcal{D}$ converges to $D(x)^* = \frac{p_d(x)}{p_d(x)+p_g(x)}$ with a fixed $\mathcal{G}$. For a fixed $\mathcal{D}$, the optimal $\mathcal{G}$ satisfies $p_g = p_d$. This theory supports many GAN-based applications. Even when considering the structure and methods of distributed training of GAN, it should still follow this base theory.

3.2 Distributed Learning Algorithms for GANs

3.2.1 Federated Learning-based GAN

There are two roles in FL, one is the parameter server (PS) and the other is the client. Each client is tasked with training a local model and subsequently sending it to the PS after L local training rounds. The server then aggregates these local models through weighted averaging and sends back the aggregated global model to the clients later. The server-side aggregation process at step t can be described as:

$$\begin{cases} \omega_s^*(t+1) = \sum_{k=1}^K \frac{n_k}{n}\omega_k(t), \\ \theta_s^*(t+1) = \sum_{k=1}^K \frac{n_k}{n}\theta_k(t), \end{cases} \tag{3}$$

where K is the number of devices in the system. The length of dataset in the device k is n_k, which satisfies $n = \sum_{k=1}^K n_k$. The client-side training process is the same as the standalone process (i.e., Eq. (1)).

3.2.2 Federated SGD-based GAN

MD-GAN and AC-GAN use a similar solution, and we take the MD-GAN as an example. In MD-GAN, the objective function in Eq. (2) is performed as two parts, including the generator and the discriminator, as follows.

$$l_g = \frac{1}{b}\sum_{x\in X_g} \mathcal{L}(1 - D(x)), \tag{4}$$

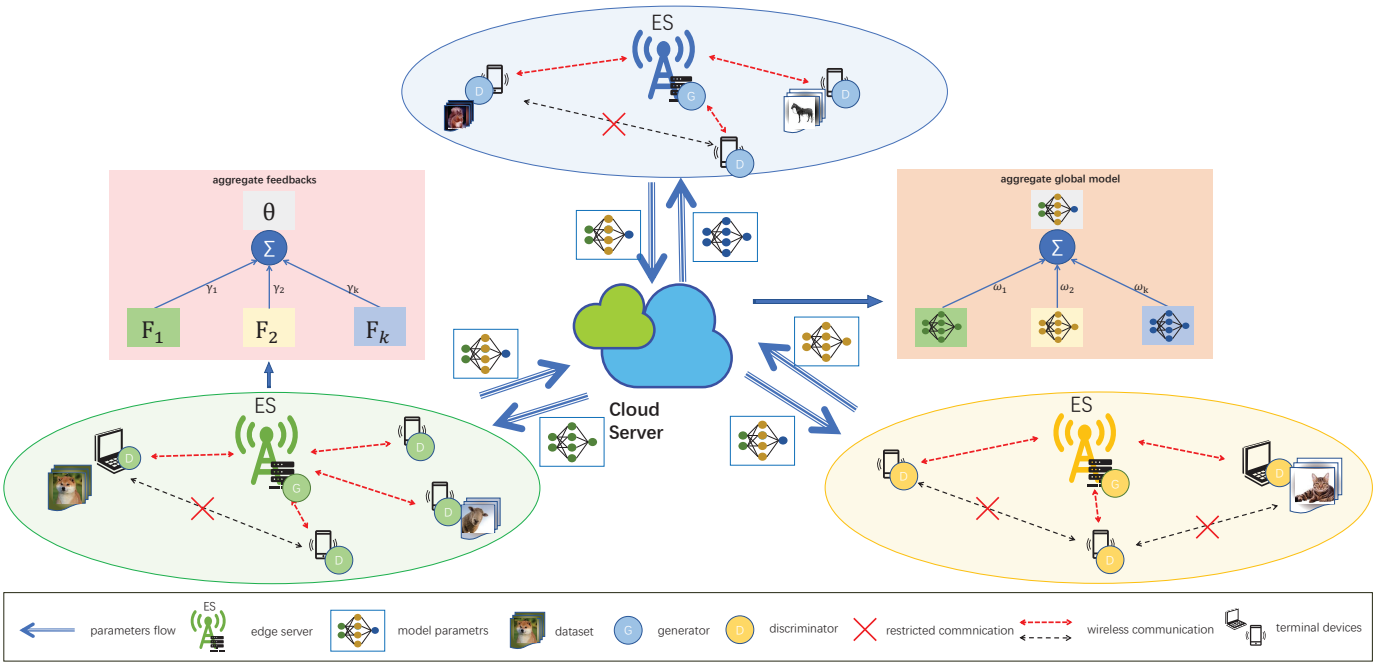


Fig. 1. The overview of CAP.

$$l_d = \frac{1}{b}\sum_{x\in X_r}\mathcal{L}(D(x)) + \frac{1}{b}\sum_{x\in X_g}\mathcal{L}(1-D(x)), \tag{5}$$

where X_r and X_g are the batch of data with size b. Based on the details presented in Section 3.1, the objective of Eq. (4) is to minimize the loss l_g, where $\mathcal{G}$ is implemented on the server. Conversely, the objective of Eq. (5) is to maximize the loss l_d, where $\mathcal{D}$ is deployed on the client.

In the beginning of each global iteration, $\mathcal{G}$ generates m batches of data, $M = \{x^{(1)}, ..., x^{(m)}\}$, $m \leq K$, each of size b. Then, two batches are assigned to each client from M, including x_g and x_d. x_d is utilized for training $\mathcal{D}$, while x_g is utilized for training $\mathcal{G}$. The client's local process for updating the gradient of $\mathcal{D}$ is:

$$\omega_i^*(t+1) = \omega_i(t) + O(\nabla_{\omega_i} l_d(x_r, x_d)), \tag{6}$$

where $O(\cdot)$ is an optimization function such as **SDG** [20] and **Adam** [21]. We use **Adam** as our optimization function for the entirety of the process. On the i-th client, $\mathcal{D}$ sends the feedback $F_i = \nabla_{x_g} l_g(x_g)$ to the server after L local iterations. Once the server collects the feedback from all K devices, the $\mathcal{G}$ is updated through,

$$\theta^*(t+1) = \theta(t) - O(\nabla_\theta \frac{1}{K}\sum_{i}^{K} F_i). \tag{7}$$

MD-GAN and AC-GAN have an extra step after every J-step, where the parameters of the discriminator are randomly exchanged between the two devices using the Gossip algorithm. This is done to enhance the generalization ability of the discriminator in each device and prevent overfitting. The step can be described as:

$$\omega_i = Swap(i, Gossip([K] \setminus \{i\})), \tag{8}$$

where $Swap(i, j)$ is a function to swap the parameters between discriminator i and j. $Gossip(\cdot)$ is a function that randomly selects an index from the device set $[K]$.

3.3 Problem Setup

We consider the scenario where there are E ESs, K clients, and one cloud server (CS) in the system. We regard the local area covered by each ES as the **cell** [17]. Let C_j ($j \in \{1, ..., E\}$) represent a set of K_j devices in a cell, satisfying $K = \sum_i^E K_j$. In this paper, we focus on more closely realistic data distribution where **Non-IID** exists both among cells and devices. Based on this consideration, we define the number of all data samples in the j-th cell devices as $N_j = \sum_{k\in C_j} n_k$, satisfying $N_i \neq N_j, \forall i, j \in \{1, ..., E\}$. In addition, let d denote the size of each sample in the dataset, which could be, for example, the total number of pixels in one image. Each edge server communicates T times with clients and T_c times with the cloud server.

4 The Proposed CAP Framework

This section describes the CAP framework. We first give the overview of CAP. Then, we detail the two key modules including the inner collaboration mechanisms of CAP and the complexity analysis of CAP-GAN.

4.1 Overview

Collaborative Game Parallel Learning (CAP) follows the learning architecture of MD-GAN and extends it to MEC. As shown in Fig. 1, CAP consists of one cloud server (illustrated as a green-blue cloud), a set of edge servers (shown as a tower), and numerous clients with the personalized dataset (shown as a phone or a laptop) distributed around the system. In this scenario, each client or cell is cut into a separate part as an information island. CAP utilizes two collaboration mechanisms to address this problem, including device-to-edge (D2E) and edge-to-cloud (E2C). For D2E, each edge server provides service for scoped clients (in a cell) through wireless communications, which is presented

as a red dashed line in the figure. Due to the distance and energy limitation, clients may not be able to communicate with other clients, i.e., the device-to-device connection is unavailable, which is indicated by a red cross. Furthermore, for E2C, the cloud provides service for global edge servers via the internet, which is shown as a blue dashed arrow. We detail the two collaboration mechanisms in the following subsection and Algorithm 1 shows the complete progress.

4.2 Collaboration Mechanisms

4.2.1 Collaboration within D2E

The collaboration within D2E is similar to MD-GAN and AC-GAN, with the exception of the aggregation algorithm implemented on each edge server. The aggregation weight vector in CAP differs from both of them. Specifically, CAP mitigates the Non-IID issue by a synthesis score consisting of the *size score* and *game score*. Intuitively, the Non-IID issue can be abstracted as two aspects: **size divergence** and **type divergence**.

Size Divergence performs as the dataset size of any client is random. CAP uses the *size score* to address it [4, 17, 22]. For each client in the j-th cell, the size score of client $k \in C_j$ can be described as:

$$\beta_k = \frac{n_k}{N_j}. \tag{9}$$

Type Divergence in GANs means that the dataset types of every client are treated as random. FeGAN [6] uses the Kullback-Leibler distance to calculate the type score to tackle this issue. However, this method has three disadvantages: 1) In a practical scenario, gathering the types from clients carries the risk of a privacy breach. 2) Knowing the label of the data in advance is not easily accessible as GAN is unsupervised. 3) This method cannot be applied to FedSGD-based GAN directly.

Based on the characteristics of GAN, CAP adopts a more private method, which exploits the game between discriminator and generator to get the *game score*. In the game phase of GAN (which is detailed in Section 3.1), the object of the generator and discriminator is to win the game. When the discriminator judges whether the input is real or not, the output of the discriminator reveals the equilibrium level of this game between $\mathcal{G}$ and $\mathcal{D}$ [23] (where the *Nash equilibrium* is $\mathcal{D}(x_d) = \mathcal{D}(x_g) = 0.5$). Therefore, the feedback in CAP contains the gradient of x_g (i.e., $\nabla_{x_g} l_g(x_g)$) and the loss of $\mathcal{G}$ (i.e., l_g), recording as $F = \{F^{(g)}, F^{(p)}\}$. Given this setup, we define the game score γ with *softmax* function as follows. In the cell j, the game score of client k is described as:

$$\gamma_k = \frac{\exp\left(\lambda F_k^{(p)}\right)}{\sum_n \exp\left(\lambda F_n^{(p)}\right)}, \tag{10}$$

where $\lambda \geq 0$ is a trainable-parameter to adjust the mixture strategies [23]. The *softmax* function (which is recommended in reference [23]) normalizes the *game score* to make $\sum_{k \in C_j} \gamma_k = 1$. Based on the aforementioned size score and game score, *synthesis score* of the k-th client is $s_k = \beta_k \times \gamma_k$.

Algorithm 1 Collaborative Game Parallel Learning-based GAN.

procedure Cloud(T_c)
for each round $t = 0, 1, ..., T_c - 1$ **do**
  Receive the parameters θ from all ES
  Aggregate the global parameter θ_{cloud} through (13)
  Send θ_{cloud} to all ESs
end for
end procedure

procedure ES(T, H)
for each round $t = 0, 1, ..., T - 1$ **do**
  $M = \{x^{(1)}, ..., x^{(m)}\} \leftarrow G(GaussianNoise(b))$
  Send $\forall(x_d, x_g) \in M$ to all devices in this cell
  Receive feedback from the clients in its cell
  Calculate the weight through Eq. (11)
  Update the generator through Eq. (12)
  if $t \mod \frac{NH}{b} = 0$ **then**
    Send θ to the cloud
    Receive θ_{cloud} from the cloud
    Update θ through Eq. (14)
  end if
end for
end procedure

procedure Device(T, L)
for each round $t = 0, 1, ..., T - 1$ **do**
  Receive all (x_d, x_g) from a edge server
  for $l \leftarrow 0$ **to** L **do**
    Update the discriminator through Eq. (6)
  end for
  Generate the feedback $(\nabla_{x_g} l_g(x_g), D(x_g))$
  Send feedback back to the edge server
end for
end procedure

According to the synthesis score, CAP gives weight to each client k contributing to the current global iteration by applying the *softmax* function, which is described as:

$$w_j = \sum_{k \in C_j} \frac{\exp s_k}{\sum_n \exp s_n}, \tag{11}$$

The *softmax* function normalizes the *synthesis score* to make the sum of weight as 1, i.e., $\sum_{k \in C_j} w_k = 1$. We also consider the linear averaging function and we will compare them in Section 6.2.3. Hence, edge server j updates its generator with new weights according to the following equation:

$$\theta_j^*(t+1) = \theta_j(t) - O(\nabla_{\theta_j} \sum_{k \in C_j} w_k F_k^{(g)}). \tag{12}$$

The existing study [23] shows λ is incremental, and it can let the generator adjust the performance of the discriminators itself to fight against more accurate adversaries. Due to the back-propagation separately executing on clients, updating λ by the recommendation in [23] (it is a centralized algorithm) needs each client to gather predictions from all other clients, which has unbearable time cost. Therefore, CAP simplifies the objective function of λ as *maximizing* the sum of predictions, $\max_{\lambda} V(\lambda) = \sum_k \gamma_k F_k^{(p)}$.

TABLE 2
Communication complexity for four major frameworks. C, E, and D stand for the cloud server, edge server, and clients, respectively.

Communication type	FeGAN	MD-GAN	AC-GAN	CAP-GAN
E $\rightarrow$ D/(E $\leftarrow$ D)	-	-	$2bdK_j/(bd)$	$2bdK_j/(bd+1)$
E $\rightarrow$ C/(E $\leftarrow$ C)	-	-	-	$\lvert\theta\rvert/(E\lvert\theta\rvert)$
C $\rightarrow$ D/(C $\leftarrow$ D)	$K(\lvert\theta\rvert+\lvert\omega\rvert)/(\lvert\theta\rvert+\lvert\omega\rvert)$	$2bdK/(bd)$	-	-

TABLE 3
Calculation complexity for four major frameworks.

	FeGAN	MD-GAN	AC-GAN	CAP-GAN
Edge server	-	-	$mbG_{op}+dbK_j$	$mbG_{op}+dbK_j$
Cloud server	$K(\lvert\omega\rvert+\lvert\theta\rvert)$	$mbG_{op}+dbK$	-	$E\lvert\theta\rvert$
Client	$2bL(G_{op}+D_{op})$	$bD_{op}(2L+1)$	$bD_{op}(2L+1)$	$bD_{op}(2L+1)$

To summarize, during each global iteration, each edge server j sends two batches of data (i.e., $\{x_g, x_d\}$) taken from the generated m batches data to clients in C_j. Following this, each client k executes its local training process with Eq. (6). Subsequently, each edge server j gathers the feedback F_k and calculates s_k from each contributing client k through Eq. (9) and Eq. (10) and the synthesis score acts on the aggregation gradient through Eq. (11) for the next step. The aggregation gradient is used to update the parameter of the generator in Eq. (12). Finally, each edge server j sends its generator to the cloud server for the global generator, which is detailed in the next section.

4.2.2 Collaboration within E2C

CAP applies the FL on the collaboration within E2C as each cell is isolated in the system (which is called **Information Island Phenomenon**). Therefore, in order to break the Information Island Phenomenon among cells, the FL algorithm is utilized to obtain the global generator by aggregating the local edge generator in order to generalize features. Therefore, the cloud receives the global parameter $\theta_{j\in E}$ after several epochs through,

$$\theta^*_{cloud}(t+1) = \sum_{j=1}^{E} \frac{N_j}{N}\theta_j(t), \tag{13}$$

where $N = \sum_{j=1}^{E} N_j$. Each edge server j receives the global parameters and updates its parameter through:

$$\theta^*_j(t+1) = \sigma\theta_j(t) + (1-\sigma)\theta^*_{cloud}(t+1), \tag{14}$$

where $\sigma \in [0,1]$ acts as control parameter of the sharing rate. When $\sigma = 1$, each edge server independently trains its generator. In this scenario, the collaboration mechanism does not work. While $\sigma = 0$, each edge server completely participates in the collaboration. Note that there is a tight connection between the distribution of the data generated by the generator and the geo-distributed real data. Reduce learning divergence [24] among edge servers by usually taking $\sigma = 0$. In addition, each edge server j traverses the dataset in its cell requiring $\frac{N_j H}{b}$ rounds, where H is a hyper-parameter for controlling the aggregation frequency. Thus the cloud server has the total round $T_c = T|\frac{N_j H}{b}$, where $|$ is an integral division operation.

4.3 Complexity Analysis

4.3.1 Communication complexity

In CAP, there are three types of communication:

- **Edge server to client communication**: In each global iteration, the edge server sends two-batch generated data to clients. Hence, the total communication of each edge server j is $2bdK_j$ (i.e., $2bd$ per client).
- **Each client to its edge server**: Different from MD-GAN or AC-GAN, each client k in CAP-GAN sends feedback including gradient $F_k^{(g)}$ and prediction $F_k^{(p)}$, where the size of gradient equals to raw data x and the prediction is a float. Thus, the total communication of each client k is $bd+1$.
- **Edge server to cloud server (vice-versa) communication**: Each edge server j regularly sends its generator with size $|\theta_j|$ to the cloud server. The cloud server receives parameters with the size of total $E|\theta_j|$ for aggregating the global generator.

Table 2 summarizes the communication complexity comparison of four major frameworks. Note that FeGAN is a variant of the FL-based GAN and others are the variants of the FedSDG-based GAN. In terms of the learning types, the communication size is determined by the GAN parameters (θ and ω) in the FL-based GAN, whereas it depends on the batch size in FedSGD-GAN variants. According to the study in MDGAN [7], using a small batch size can improve the performance of parallel learning. Therefore, the main factor determining the communication consumption is thus the size of the samples. As long as the size of the samples is much smaller than the number of parameters of the GAN, the communication load of FedSGD-GAN variants is smaller than FL-based GANs. In addition, the FedSGD-GAN variants require to communicate with the server during every L local iteration (where L is usually taken as the value 1), whereas FL-based GANs perform multiple epochs before the next communication. It causes the FedSGD-GANs variants to have more frequent communications between servers and clients with a given number of iterations.

In terms of the framework in FedSGD-GAN variants, the first observation is that CAP-GAN has similar communication consumption with AC-GAN and MD-GAN, except for the different sizes of K in MD-GAN and K_j in CAP-GAN.

It is obvious that most global iterations are dealt with at the edge with edge servers. This brings two advantages. First, clients have a shorter interval to receive the result of edge servers. Second, the core network has less traffic pressure. On the other hand, the total communication of edge servers in CAP-GAN is larger than AC-GAN, because CAP-GAN has the collaboration mechanism in E2C. Additionally, the communication consumption of clients in all frameworks is fixed.

4.3.2 Computation complexity

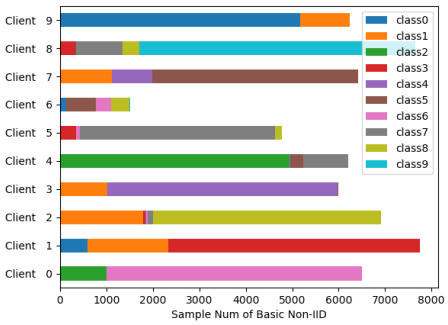
In CAP, there are three types of roles:

- **Workload at the client**: the computational load on clients in CAP-GAN equals that in MD-GAN. For per global iteration, each client receives two batches of data from its edge server and performs $2bLD_{op} + bD_{op}$ floating point of operations (FLOPs) (where D_{op} is the number of FLOPs of each feed-forward pass of the discriminator).
- **Workload at the edge server**: During each global iteration, every edge server j generates m batches data at the beginning and calculates the aggregation gradient of all feedbacks in the end. The FLOPs of generation operation are mbG_{op} (where G_{op} is the number of FLOPs of each feed-forward pass of the generator). The FLOPs of aggregation gradient are dbK_j.
- **Workload at the cloud server**: During each communication in E2C, the cloud server gathers the generator in all edge servers and aggregates to get the global generator. The FLOPs of each aggregation are thus $E|\theta|$.

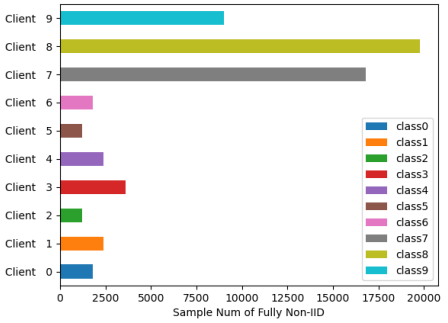
Table 3 presents a summary of the computational complexity of four major frameworks. The GAN is a kind of computation-intensive task. Especially, FeGAN requires clients to execute the full process of GAN per global iteration. This means that its workload is doubled when compared to FedSGD-GANs, totaling $2bL(G_{op} + D_{op})$. In comparison, MD-GAN, AC-GAN, and CAP-GAN have around half of the workload of FeGAN on clients. On the other hand, the edge servers in CAP-GAN have the same workload as those in AC-GAN, but less than MD-GAN and FeGAN. The reason is that both CAP-GAN and AC-GAN utilize edge servers to offload the learning tasks from clients, which balances the workload for each edge server. However, it should be emphasized that CAP-GAN pays more attention to addressing the Non-IID scenarios than AC-GAN. Furthermore, because CAP-GAN incorporates more devices (i.e., clients, edge servers, and a cloud server), the total computational consumption of the global system is inevitably higher than other algorithms.

4.4 Comprehensive analysis

The energy consumption of clients is majorly caused by calculation and wireless communication. According to the above analysis, the computation consumption on the client in CAP-GAN is the same as those in other FedSGD-GAN variants, whereas it is double in FL-GANs, given an equal number of communications. Furthermore, Under an equal number of local iterations, the FedSGD-GANs variants require more frequent communication between clients and servers than FL-GANs. The experimental results in Section 6.2.1 demonstrate that CAP-GAN has better performance than FeGAN with an equal number of interactions between the client and server (edge server).



(a) Basic Non-IID



(b) Fully Non-IID

Fig. 2. The difference between Basic Non-IID and Fully Non-IID.

5 The Mix-Generator Module

Non-Identically and Independently Distributed (Non-IID) is an open issue in the domain of Federated Learning. However, the issue of Non-IID is primarily discussed in the context of classification tasks, while it is not well studied in generation tasks, especially in GANs. After summarizing the Non-IID problems in previous FL works [4–6, 25], we divide the Non-IID problems into two main types - **Basic Non-IID** and **Fully Non-IID**.

In an IID scenario, each client is assigned an equal number and class of samples. However, the number and class of samples in clients are randomly assigned under both Basic and Fully Non-IID scenarios. The key difference between Basic and Fully Non-IID is that the former allows for the overlap of sample classes across different clients, while the latter ensures that every client receives a unique set of classes without duplication among clients. An example is shown in Fig 2(a), there are overlap classes among clients in a scenario of Basic Non-IID, for example, client 0 and client 4 both have class 2. Conversely, a Fully Non-IID scenario is shown in Fig. 2(b) where each client has a unique class, thus Fully Non-IID datasets represent the highest level of personalization in a system. Although Algorithm 1 can handle a certain degree of the Non-IID scenarios, using a

single model to gather all global features in such scenarios can be challenging [26].

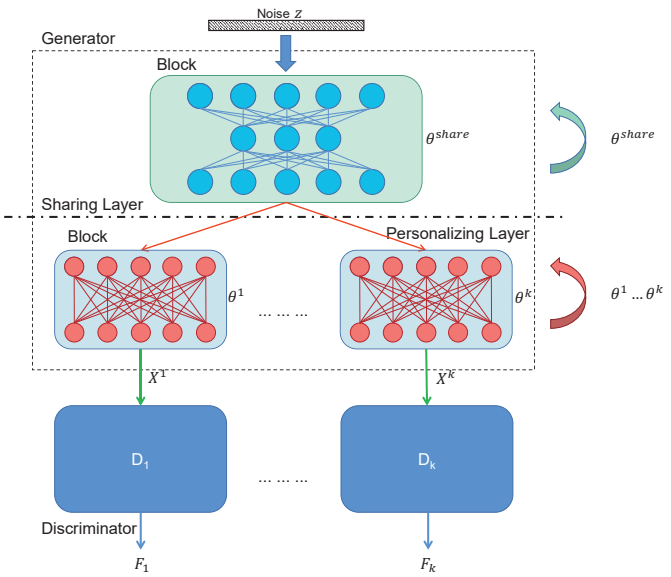


Fig. 3. Architecture of the Mix-Generator.

In order to address the challenge of Non-IID scenarios, we design the **Mix-Generator** (Mix-G) module, drawing inspiration from the CoupleGAN [27] model. CoupleGAN has shown that although data for the same task may be non-independent and non-identical, they still share similar representations. Because a single global model hardly covers all personalized participants, the generator is split into two layers by regarding the participants as identical tasks with different representations, including the sharing layer and the personalizing layer. Specifically, the personalizing layer has the same number of Neural Network blocks as the clients in a cell, where these blocks are called personalizing blocks. As shown in Fig. 3, these personalizing blocks use the common mid-feature output from the sharing layer, which can be described as:

$$X_j^{share} = \theta_j^{share}(z), \tag{15}$$

$$X_j^k = \theta_j^k(X_j^{share}), \tag{16}$$

where k presents the index of devices in the cell j. The mid-output θ_j^{share} is the global feature representation. Then, the personalizing block transforms the intermediate feature graph into a data representation with personalization characteristics. Therefore, the Mix-G module updates the generator in two steps.

The first step updates the personalizing blocks. For each feedback from client k, the personalizing block θ_j^k updates the parameter with personalization feedback gradient $F_k^{(g)}$, as follows:

$$\theta_j^k(t+1) = \theta_j^k(t) - O(\nabla_{\theta_j^k} F_k^{(g)}). \tag{17}$$

The second step updates the sharing layer. To aggregate the global feature representation, the sharing layer updates the parameter with weighted aggregation feedback gradients, as follows:

$$\theta_j^{share}(t+1) = \theta_j^{share}(t) - O(\nabla_{\theta_j^{share}} \sum_k w_k F_k^{(g)}). \tag{18}$$

Algorithm 2 The CAP-GAN enabling Mix-G.

procedure ES(T, H)
for each round $t = 0, 1, ..., T-1$ **do**
  $M = \{x^{(1)}, ..., x^{(m)}\} \leftarrow G(GaussianNoise(b))$
  Send $\forall(x_d, x_g) \in M$ to all devices in this cell
  Receive feedback from the clients in its cell
  Calculate weight through Eq. (11)
  Update personalizing layer through Eq. (17)
  Update sharing layer through Eq. (18)
  if $t \mod \frac{NH}{b} = 0$ **then**
    Send θ^{share} to the cloud
    Receive θ_{cloud} from the cloud
    $\theta^{share} \leftarrow \theta_{cloud}$
  end if
end for
end procedure

Algorithm 2 shows the complete process of CAP-GAN enabling the Mix-G module. The sharing layer in Mix-G ensures aggregating the global feature representation into the same hidden space [27], and the personalizing blocks can further transform the features in the hidden space into personalization features. This approach only expands the last layer of the original generator as opposed to configuring a full generator for each client. Thus, the communication of Mix-G has no change compared with the previous, while the computation of Mix-G has more consumption than pure CAP-GAN. In addition, the size of generator with enabling Mix-G is $|\theta_j| = \left|\theta_j^{share}\right| + \sum_{k \in C_j} \left|\theta_j^k\right|$, and the FLOPs of θ_j^k and θ_j^{share} are $\theta_{op_j}^k$ and $\theta_{op_j}^{share}$. The computation load of the edge server using the Mix-G module is $(mb(\theta_{op_j}^{share} + K_j\theta_{op_j}^k) + dbK_j)$.

Similarly, the FL algorithm is modified in the part between ES and the cloud. The parameters in Eq. (14) are replaced by the sharing block. Thus, the communication and computation of the cloud server are both $E\left|\theta^{share}\right|$. In addition, as each generator sends the sharing layer to the cloud server, this module can be regarded as **soft sharing** which is similar to hard sharing[1] in [27, 28].

Note that, the Mix-G module can address all Non-IID scenarios and well-perform on even the highest personalized scenarios. In addition, when using CAP-GAN in realistic scenarios and the additional computational burden brought by Mix-G is tolerable, Mix-G can be enabled to cope with unknown data distributions.

6 Experimental Evaluation

We compare our approach to three state-of-the-art benchmark algorithms (FL-GAN, MD-GAN, and AC-GAN) that used mixture-Gaussian, MNIST, and Fashion-MNIST in this section. We start by going over the setup needed for the experiments. To highlight the differences between these algorithms, we create three data distribution scenarios and then compare and analyze the data generated by them in each scenario. Then, using a set of preset noise production data, we observe how each algorithm performs over a range

1. Hard sharing is a restricted parameter sharing, which two models use the same parameter. Soft sharing is a loose parameter sharing, which allows the parameters of two models to be similar.

TABLE 4
The Generated Data Evaluation of FL-GAN, MD-GAN, AC-GAN, CAP-GAN.

Dataset	Model	Metric	IID	Basic Non-IID	Fully Non-IID
MNIST	AC-GAN	MNIST Score ↑	6.5244 (± 0.0994)	7.9712 (± 0.1296)	6.6093 (± 0.0972)
		FID ↓	279.4811 (± 2.1697)	76.2112 (± 0.9904)	74.2777 (± 1.2370)
	FL-GAN	MNIST Score ↑	6.7094 (± 0.1371)	4.8378 (± 0.1281)	3.7940 (± 0.0694)
		FID ↓	**59.8908 (± 1.9911)**	145.4118 (± 1.0244)	211.1123 (± 1.6114)
	MD-GAN	MNIST Score ↑	**7.3549 (± 0.1096)**	4.9577 (± 0.0577)	3.1687 (± 0.0388)
		FID ↓	77.5787 (± 1.3256)	89.7187 (± 2.0857)	81.6724 (± 1.8405)
	CAP-GAN	MNIST Score ↑	6.7285 (± 0.1171)	**8.3826 (± 0.0879)**	**9.1990 (± 0.1022)**
		FID ↓	96.3913 (± 1.2728)	**65.5164 (± 1.5935)**	**38.3307 (± 1.4022)**
Fashion-MNIST	AC-GAN	Mode Score ↑	**8.2757 (± 0.0644)**	8.3668 (± 0.0811)	7.6341 (± 0.0656)
		MMD (100×) ↓	4.52 (± 0.77)	3.30 (± 0.65)	12.1488 (± 1.0498)
	FL-GAN	Mode Score ↑	8.2234 (± 0.0790)	6.1999 (± 0.0722)	5.8465 (± 0.0873)
		MMD (100×) ↓	6.16 (± 0.47)	19.97 (± 0.88)	28.91 (± 0.42)
	MD-GAN	Mode Score ↑	8.2654 (± 0.0767)	8.2676 (± 0.0785)	6.1616 (± 0.0766)
		MMD (100×) ↓	3.86 (± 0.50)	4.19 (± 0.39)	21.19 (± 2.16)
	CAP-GAN	Mode Score ↑	8.2274 (± 0.0410)	**8.5013 (± 0.0899)**	**8.6185 (± 0.0513)**
		MMD (100×) ↓	**1.97 (± 0.37)**	**1.12 (± 0.20)**	**0.88 (± 0.09)**

of iterations. We test the impact of different weights of aggregating feedback. In the end, we discuss the performance of whether enable the Mix-G or not on Non-IID.

6.1 Experimental setup

Our experiments are all based on the PyTorch library. We create a simulation experiment platform on a GPU-based server outfitted with an Inter Xeon Gold 6226R processor, 128 GB of RAM, and an NVIDIA RTX 3090 GPU. Threads are used in this platform to simulate task execution in various nodes such as devices, ESs, and cloud servers in order to simulate overheads in a real-world environment [2]. Note that, the calculation of Federated SGD-based GAN is complex, we use multi-thread and Pytorch to build the auto-gradient chart. Inter-thread communication is used to simulate network communication, allowing GANs to be trained in the same order as a real distributed deployment. It is important to emphasize that we do not use gradient penalties [29] or other optimization techniques [30] to enhance the performance of our GAN. Our object is not to obtain a better performance than other GAN but to focus on minimizing the impact of the decentralized data distribution on the GAN.

6.1.1 Datasets

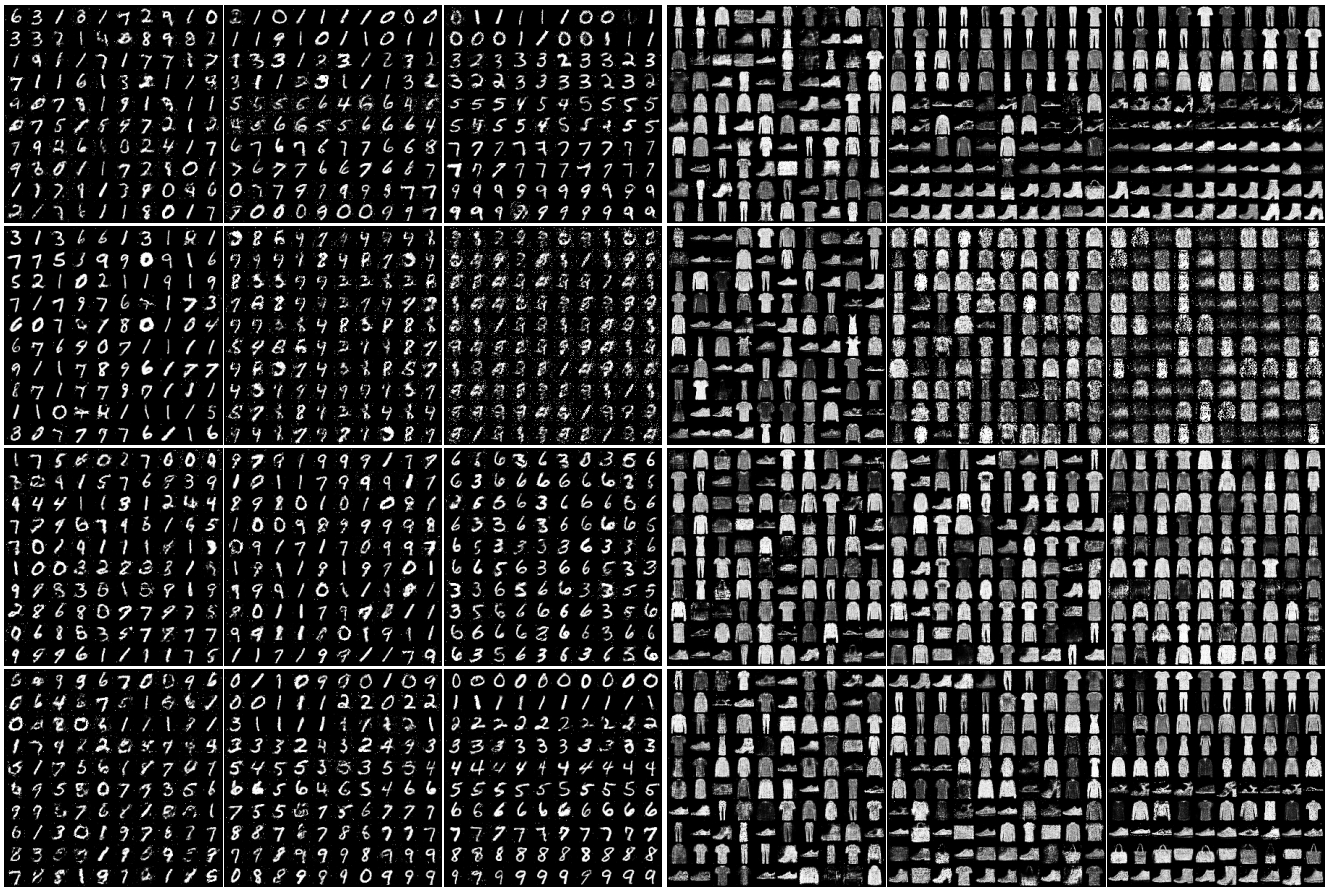
Two standard picture datasets (MNIST, and Fashion-MNIST), and a two-dimensional random Gaussian dataset, are used in this study. MNIST is made up of a training dataset of 60,000 gray-scale handwritten digit images (28 × 28 pixels) and a test dataset of 10,000 images, with 10 classes ranging from 0 to 9. The Fashion-MNIST dataset contains 60,000 training samples (28 × 28 pixels) and 10,000 test samples. T-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and boot are among the ten sample classes of Fashion-MNIST, which images are similar to MNIST. These datasets are frequently used to evaluate the generative power of GAN. We utilize the same normalized prepossessing operation on each image of these datasets before training any GAN. Furthermore, mixture-Gaussian is a common way for evaluating GAN performance, as it can visually analyze the change in the generative effect of GANs. Same as the other graph datasets, we create 10 two-dimensional Gaussian distributions with different means and variances, then combine them to create a two-dimensional (2D) mixture-Gaussian dataset. Since our work is concerned with the distribution of data among devices, we designed three scenarios, IID, Basic Non-IID, and Fully Non-IID. In different scenarios, the data contained in each device differs in the type of samples, the number of samples, and the specific settings as follows:

- **IID**: Each device has the same number of samples (i.e., $n_1 = n_2 =, ..., n_k$) and all classes of the global dataset (which the global dataset consists of all datasets in devices).
- **Basic Non-IID**: Each device has a random number of samples and part of classes of the global dataset, which allow existing overlapping classes between devices.
- **Fully Non-IID**: Each device has a subset of the global dataset and each subset only carries one class of data. Furthermore, the subsets have no overlap of class.

6.1.2 GAN model architecture

Different model architectures and parameters are employed on different datasets. For the 2DGM dataset, the generator uses three fully-connected networks with a size of 100 × 128, 128 × 256, and 256 × 2, and the discriminator consists of three fully-connected networks with a size of 2 × 128, 128 × 256, and 256 × 1. For MNIST and Fashion-MNIST datasets, the generator consists of five-layer fully-connected

2. The code of this platform and four algorithms is available at https://github.com/NetworkCommunication/CGL-GAN



(a) IID (b) Basic Non-IID (c) Fully Non-IID

Fig. 4. The sample images are generated under three data distributions on MNIST.

(a) IID (b) Basic Non-IID (c) Fully Non-IID

Fig. 5. The sample images are generated under three data distributions on Fashion-MNIST.

networks with sizes of 100 $\times$ 128, 128 $\times$ 256, 256 $\times$ 512, 512 $\times$ 1024, and 1024 $\times$ 784. The discriminator consists of a three-layer fully-connected network with sizes of 784 $\times$ 512, 512 $\times$ 256, and 256 $\times$ 1.

6.1.3 Metrics

Evaluating the performance of GAN may be a challenging task because the use of a single metric assessment sometimes does not prove the merit of the model. Therefore, we use different metrics for different datasets in order to better represent the performance differences of different algorithms.

- **2D Mixture-Gaussian dataset**: we design a method to calculate the difference in performance. We divide the 2D data into equal-length grids and count the number of real and generated data in these grids separately. To facilitate the subsequent calculations, we take out those grids for which data exist and the corresponding grids for which data are generated. The evaluation of the generated data using the following approach.

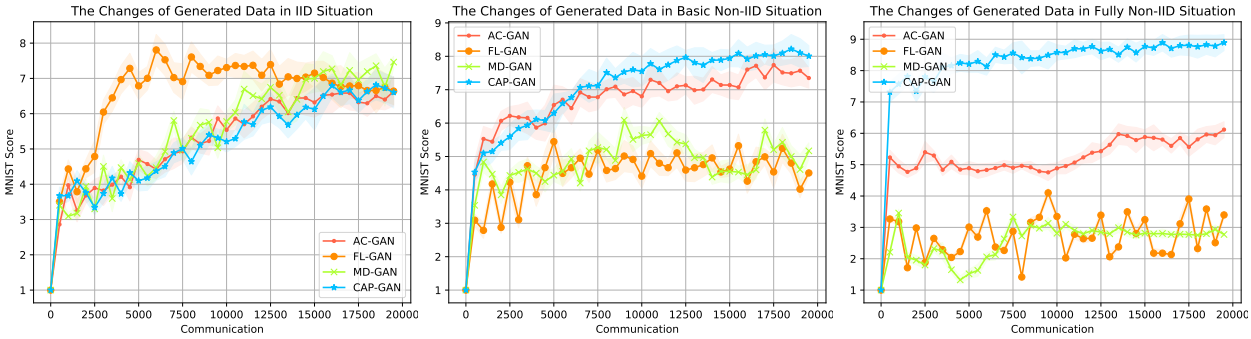
$$KL\ Score = KL(N_g \parallel N_r). \tag{19}$$

  N_g and N_r are the distribution of those grids from real data and generated data. *KL Score* (KLS) shows the gap between the distribution of the data generated by the model and the distribution of the real data, the smaller, the better.

- **MNIST dataset**: Compared with 2D datasets, high-dimensional datasets, such as images, present a challenge in measuring distribution differences. Fortunately, *Inception Score* [31] (IS) presents a mechanism for resolving it. IS is carried out on the generated data by a pre-trained classification model (Inception) in order to assess the quality and diversity of the data. Due to the original IS model being trained on CelebA, its ability to accurately classify the MNIST dataset is not entirely convincing. As a result, we employ *MNIST Score* [3], similar to IS, but using a classifier network adapted to MNIST. Another metric is *Fréchet Inception Distance* (FID) [32] [4] which is to measure the distance between p_g and p_d. It runs the Inception network on a sample of produced data and another sample of real data, assuming Gaussian distributions as outputs. The FID calculates the Fréchet Distance between a Gaussian distribution created from generated data and a Gaussian distribution generated from real data. Please note that the value of the MNIST Score increases as it gets better, while the FID decreases as it improves.
- **Fashion-MNIST**: Since Fashion-MNIST is an MNIST-like dataset, *Mode score* [33], which is an improved version of MNIST Score, is applied to this dataset for

3. Code available at https://github.com/ChunyuanLI/MNIST_Inception_Score

4. Code available at https://github.com/mseitzer/pytorch-fid



(a) IID (b) Basic Non-IID (c) Fully Non-IID

Fig. 6. The changes of MNIST Score of AC-GAN, FL-GAN, MD-GAN, and CAP-GAN under three scenarios.

evaluating the diversity of the generated models. Unlike the MNIST Score, the Mode Score adds a measure of the real dataset, and the Mode Score is maximized only when the class distribution of the generated data is equal to the class distribution of the real data. As for the clarity of the generated images, unlike the previous dataset, we use *Kernel Maximum Mean Discrepancy* (MMD) to calculate it. The MMD is calculated as follows:

$$MMD^2(\mathbb{P}_r, \mathbb{P}_g) = \mathbb{E}_{\substack{x_r, x_r^{'} \sim \mathbb{P}_r, \\ x_g, x_g^{'} \sim \mathbb{P}_g}} [k(x_r, x_r^{'}) - 2k(x_r, x_g^{'}) + k(x_g, x_g^{'})]. \quad (20)$$

where k is a kernel function that generally adopts Gaussian Kernel $k(x, y) = \exp - \frac{\|x-y\|^2}{2\sigma^2}$. Similar to FID, the MMD is based on converting high-dimensional data into Hilbert space and calculating the distance between two data. As a result, the lower the value of MMD, the better the model.

6.1.4 Configurations

Three benchmark algorithms chosen in this paper are FL-GAN, MD-GAN, and AC-GAN. In order to unify the metrics, we use the same values for the parameters that are common to all algorithms. These hyper-parameters can be determined randomly and there is no optimal selection in this paper. The following values are chosen based on the works on AC-GAN and MD-GAN. For the total number of devices, we let it be the same number of classes as the sample of all datasets, i.e., $k = 10$. For the number of ESs, we choose 5 in our algorithm and AC-GAN, and there is no ES in FL-GAN and MD-GAN which only have 1 parameter server in all experiments. Then, we define the batch size $b = 100$ and the local training iteration $L = 1$ (Especially for FL-GAN, $L = 5$). The loss function $O(\cdot)$ is $\log$ and the optimal function $\mathcal{L}(\cdot)$ is Adam, with a learning rate of 0.0002, a first-order gradient momentum decay of 0.5, and a second-order gradient momentum decay of 0.999. These models in ESs and devices communicate 20,000 iterations in the whole algorithm for graph datasets and 10,000 for the 2D mixture-Gaussian dataset. In addition, we let the sharing rate controlling parameter $\sigma = 0$ share the model with other ESs fully, and with each 1 epoch (i.e., $H = 1$), the ES shares its model to the cloud and receives the global model. In addition to experiment parameters, we also configure the set of devices being served by each ES. As our dataset is designed to be adequately representative of the geographical distribution of real-world data, we evenly distribute devices to all ESs to simplify calculations and experiments.

6.1.5 Comparison

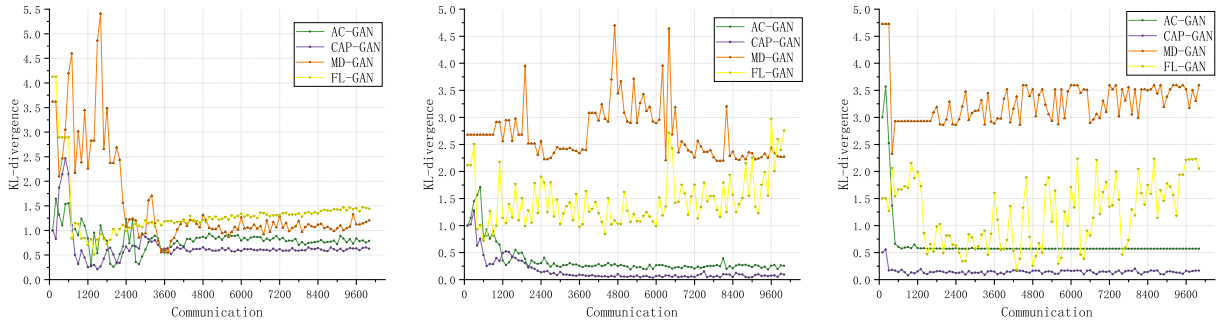
We choose four SOTA algorithms to compare the performance of CAP-GAN. we describe these algorithms as follows:

CAP-GAN: We employ pure CAP-GAN in the IID experiment without enabling the Mix-G module. Because the datasets in all clients have the same size and type in IID scenarios, enabling Mix-G is useless and redundant (as Mix-G has higher computation on edge servers). And we enable the Mix-G module on CAP-GAN in all Non-IID scenarios. A more detailed discussion will be presented in Section 6.2.4.

FL-GAN [7]: FL-GAN is an artificially created baseline of the FL-based GAN algorithm [7]. FL-GAN contains a cloud server (parameter server) and a set of devices with a complete GAN. Each client sends its local model (includes $\mathcal{G}$ and $\mathcal{D}$) to the cloud server every 10 local iterations. The cloud server aggregates the global modes through Eq. (3). Compared with CAP-GAN, FL-GAN has a double computation workload on clients and it does not support Non-IID scenarios.

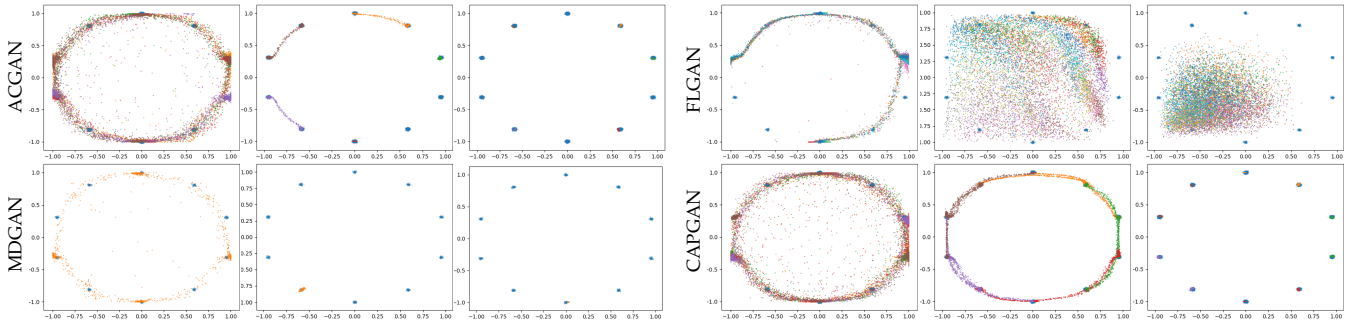
FeGAN [6]: FeGAN is a FL-based GAN for Non-IID issue. In order to customize the weight and list of entry clients of all rounds, FeGAN needs to collect the number of each class in each client. Clearly, this approach defeats the original intent and purpose of Federated Learning and Federated SGD. Due to it being the SOTA algorithm of FL-based GAN, we compare the algorithm of aggregation with FeGAN. We will discuss it in Section 6.2.3. Compared with CAP-GAN, FeGAN has a double computation load on clients.

MD-GAN [7]: MD-GAN has the same components as FL-GAN, a cloud server, and a set of devices. The cloud server holds a generator and each device holds a discriminator. The generator in the cloud sends generated fake data to devices and receives feedback to aggregate gradient through Eq. (7). Each device mixes the real local data and receives fake data to train its discriminator through Eq. (6) and sends the feedback which is calculated by Eq. (4). Compared with



(a) IID (b) Basic Non-IID (c) Fully Non-IID

Fig. 7. The changes of KL-divergence of AC-GAN, FL-GAN, MD-GAN, and CAP-GAN under three scenarios.



(a) IID (b) Basic Non-IID (c) Fully Non-IID (d) IID (e) Basic Non-IID (f) Fully Non-IID

Fig. 8. The images presented were produced using three distinct data distributions within 2DMG.

CAP-GAN, MD-GAN provides poor support on Non-IID scenarios and edge scenarios.

AC-GAN [8]: AC-GAN has similar components as CAP-GAN, which contains a set of ESs and each ES covers a set of devices. Each ES holds a generator, and each device holds a discriminator. The generator in each ES sends generated fake data to devices within its coverage and receives feedback to aggregate gradient through Eq. (7). Each device mixes the real local data and receives fake data to train its discriminator through Eq. (6) and sends the feedback to its ES which is calculated by Eq. (4). Compared with CAP-GAN, AC-GAN provides poor support in Non-IID scenarios.

Note that, in order to align with the scenario outlined in this paper, we have canceled the communication (as shown in Eq. (8)) between devices in the algorithms including MD-GAN, and AC-GAN.

6.2 Experimental results

The performance differences of AC-GAN, FL-GAN, MD-GAN, and CAP-GAN are compared in Table 4. All the results are compared with models built after 20,000 communications using the same collection of 10,000 Gaussian noises.

6.2.1 Convergence

Fig. 4 and Fig. 5 show the image result of four algorithms, and Table 4 shows the metric results of four algorithms. In the IID setting, AC-GAN, MD-GAN, FL-GAN, and CAP-GAN perform almost equally well. Since all clients share the same sample distribution, FL-GAN and MD-GAN can directly aggregate global information, resulting in high performance in terms of FID and MNIST Score metrics. However, as the distribution becomes more Non-IID, CAP-GAN outperforms the other methods.

The results show that FL-based GANs suffer from a significant amount of information loss, even generating blurry data when faced with unbalanced samples in the Fully Non-IID scenarios. However, CAP-GAN stands out by retaining features and patterns from all data, resulting in clear data with FID and MMD metrics that are significantly better than other state-of-the-art algorithms. This is because, in highly dispersed data, the feedback from each discriminator can become biased and lead to overfitting, making it difficult for generators to learn the accurate distribution characteristics of the data. The other three algorithms use the uniform aggregation of features to generate data, which makes it impossible for the generator to choose which class to favor, so it often ends up generating only one class.

In contrast, realistic scenarios often exhibit extremely discrete characteristics, which means that the first three algorithms cannot be adapted to practical application scenarios. CAP-GAN uses a separate personality layer for binding the data features of individual devices, which allows the personality layer to select from the output of the feature by the shared layer the parts that are useful for the devices it binds and generates more realistic data. Moreover, Fig. 6 shows this scheme can accelerate the convergence of GAN when the data distribution becomes more non-independent and identical. Since the features of a single type of dataset are also single, the personality layer converges quickly. Note

that, we set Algorithm 1 on the IID setting and Algorithm 2 on both Non-IID settings.

6.2.2 Diversity

The results shown in Fig. 4 and Fig. 5 are generated by using fixed noise after 20,000 rounds of communication.

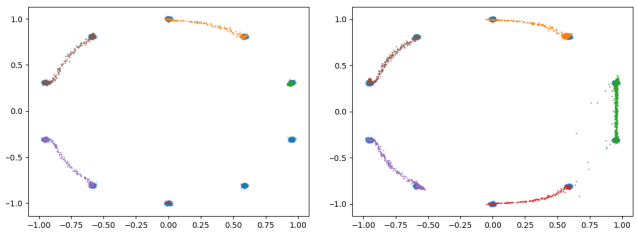
First, CAP-GAN has the ability to generate all types of samples across various datasets and scenarios, whereas the other three algorithms are limited to generating a more comprehensive range of data types in IID cases, and only a few classes of samples are generated in the other two Non-IID cases. The most obvious one is MD-GAN, which generates only 3, 6, and a small amount of 5 on the MNIST dataset in Fully Non-IID scenarios. AC-GAN is a little better, but it does not generate all the numbers, which miss the numbers 6 and 9. This is because when the sample class is single for each device in this case, the feedback from the discriminator becomes significantly biased and could be particularly sensitive to other classes of data generated by the generator.

Second, due to the uneven amount of data, the discriminators do not have equal classification power, making it challenging for AC-GAN and MD-GAN, which are based on average feedback, to handle all categories. In Fig. 8, we depict their performance on the 2DMG dataset, which provides a more intuitive result. In this figure, the blue dots represent the distribution of the dataset, while the other colored dots represent the distribution of data generated by the algorithm. The results indicate that only CAP-GAN can generate all types of data samples in all instances, demonstrating its superior performance in terms of diversity. Moreover, the outcomes of AC-GAN and MD-GAN reveal that their algorithms cannot ensure the generation of all classes of samples. And FL-GAN fails to adapt to the Non-IID scenario and generates bad samples.

In the Basic Non-IID example, there is a noteworthy result. In terms of the generated data, there is not much difference between AC-GAN and CAP-GAN outcomes, with AC-GAN falling short on just one type. However, this outcome supports our earlier findings that data imbalance leads to biased feedback from the AC-GAN generator. Such bias can cause the gradient to decrease in an uneven manner, resulting in a biased outcome. In contrast, the aggregation algorithm and the personality layer of CAP-GAN ensure that no characteristic is overlooked, thereby contributing to its excellent performance in generating diverse samples. In addition, although the difference between AC-GAN and CAP-GAN cannot be seen in Fig. 8, the results in Fig. 7 prove that the distribution of the data generated by CAP-GAN is more comprehensive.

TABLE 5
The FID result of different weights of aggregating feedback.

Weight	IID	Basic Non-IID
only size score	68.90	134.88
only game score	88.62	270.66
averaging score	**67.63**	225.56
linear average of synthesis score	94.46	238.97
softmax of synthesis score	74.82	**119.46**



(a) ACGAN (b) pure CAP-GAN

Fig. 9. The performance in the Basic Non-IID scenario.

6.2.3 The Effects of β and γ

Compared with MD-GAN and AC-GAN, we use a more complex scheme in aggregating the feedback. In general, the relationship between clients should be equal. This is one reason why they opted to use the mean (called the averaging score) gradient descent model. But in fact, this is an unfairness issue here as this completely ignores the impact of the number and class of data samples. Thus, we use *synthesis score* consisting of *size score* and *game score*.

To figure out the role of these weights, we design a set of experiments to test their performance under different distributions, and the results are summarized in Table 5. The experimental setup involves one server and the MNIST dataset, without the E2C and Mix-G modules. The FID is calculated based on the last 1000 results from the generator. Moreover, we show the results of two kinds of normalization on *synthesis score*. The first one is linear averaging as $w_k = \frac{s_k}{\sum_n s_n}$. The second one is *softmax* as $w_k = \frac{\exp s_k}{\sum_n \exp s_n}$. Note that, in the IID setting, the *size score* is equal to the average (e.g., MD-GAN). The results show that calculating the weights according to the number of samples can obtain better results than the average and *game score* under Non-IID.

We notice that using only the weight of *game score* does not work well by itself, while the weight of *synthesis score* obtains a significant result. We consider it is because GMAN [23] uses *game score* work in a centralized dataset rather than distributed dataset. For Non-IID, *size score* provides information about the data distribution, so that *synthesis score* can obtain better results. For diversity, Fig. 9 shows the weighted average can gain more diversity than the complete average. For devices with few samples, average weights can cause harmful updates to the global model due to the imbalance of data sizes between devices. Thus, combining the amount of data and the state of the discriminator can make the generator of each server take more account in an integrated manner.

To highlight the variances in the aggregation algorithm of FeGAN, we perform a separate comparison, employing the model configuration from FeGAN. Specifically, we replace the MLP model with LSGAN [19], which consists of multiple convolution layers. The remaining configuration of CAP-GAN is identical to the previous study. We compare the performance differences between the two aggregations in Basic Non-IID conditions. As shown in Fig. 10, the CAP-GAN algorithm exhibits superior convergence speed and convergence effect compared to FeGAN. It suggests that utilizing weight information obtained from the interaction

between the generator and discriminator can achieve better results than simply balancing the number of sample types.

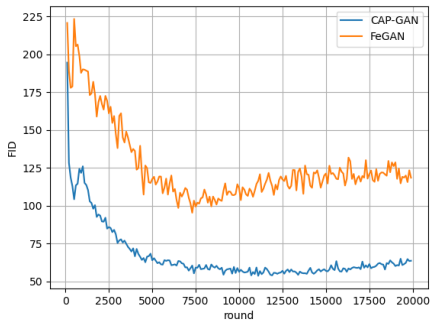


Fig. 10. Convergence of FeGAN and CAP-GAN on MNIST

6.2.4 The Effects of Mix-G Module

The Mix-Generator (Mix-G) module is the most significant module to address Non-IID. For any Non-IID distribution, the main feature of each client is non-uniform. The Mix-G module uses a personalizing layer to match the distinctive feature of a client. We design a similar experiment to figure out the impact of the personalizing layer. Except for adding the personalizing module, other settings are as same as that in the last experiments. Fig. 11 shows the result, where the lower FID represents the more clarity of the picture. The experiments reveal that Mix-G can enhance performance when the dataset exhibits greater Non-IID characteristics. The personalizing layer of Mix-G ensures that the unique features of the user are not discarded, and thus the Mix-G module generates all possible samples. For privacy, the biggest advantage of the Mix-G module is that the user's features are held by their dedicated module, and the sharing layer only contains the representation features of global datasets, which further enhances privacy.

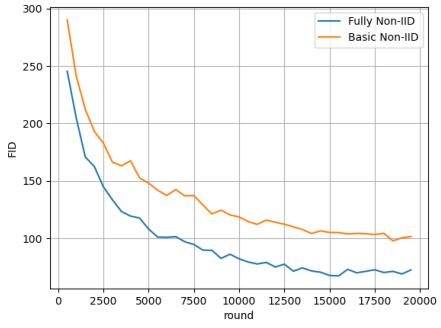
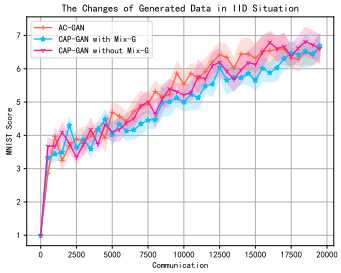
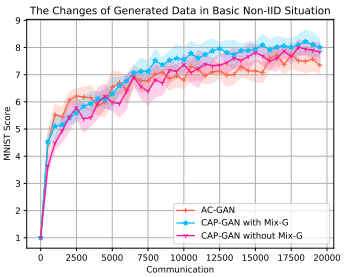


Fig. 11. The convergence of enabling the Mix-G module on MNIST in the Non-IID scenarios

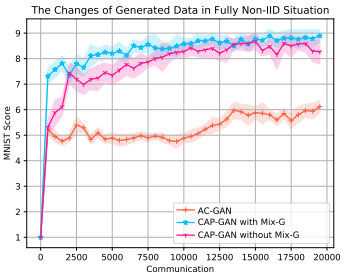
It is worth noting that the Mix-G module can be plugged into all scenarios, including both IID and Non-IID, even in the unknown scenarios. However, we do not advise enabling the Mix-G module in known IID scenarios because the IID datasets in all clients have the same feature. Enabling the Mix-G module to cope with these repetitive features violates the objective of Mix-G. In this way, the Mix-G becomes a redundant module and does not provide improvement while requiring more calculations, which is shown in Fig. 12(a). The Mix-G module is designed to handle the distinctive features of the client and enhance the quality and diversity of generated samples in Fully Non-IID scenarios. As illustrated in Fig. 12(b), enabling the Mix-G module slightly improves the performance even in the Basic Non-IID scenarios. In contrast, Fig. 12(c) shows that the convergence speed and the diversity of the generated data are substantially improved in the Fully Non-IID case.



(a) IID



(b) Basic Non-IID

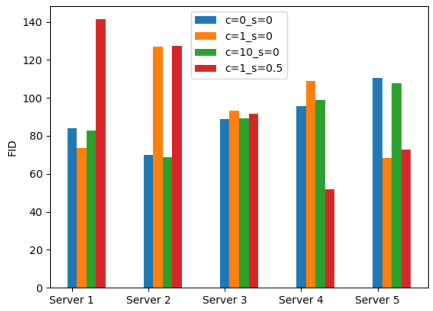


(c) Fully Non-IID

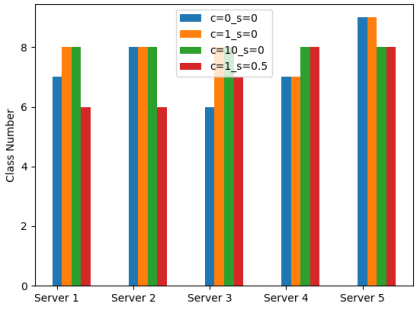
Fig. 12. Convergence comparison in the Non-IID scenarios with or without Mix-G.

6.2.5 The effects of other hyper-parameter

An obvious conclusion is that algorithms with edge scalability can reduce the load on servers and thus speed up the response time of applications. Hence, the systematic structure of CAP outperformed other models. For the cloud module, the effect of the cloud module is as same as the parameter server in FL. Without the cloud module, the IIP issue in edge also emerges between edge servers. We finish a set of experiments to show the impact of the cloud module and sharing rates. We set the cloud epoch as c and sharing rate as s. For instance, $c = 0$ and $s = 0$ mean the cloud server does not collect sharing layers from the edge server.



(a) FID scores on each edge server



(b) The number of generated sample class

Fig. 13. The result of 5 edge servers and 50 clients

Fig. 13(a) shows that E2C is limited in its ability to enhance clarity, whereas Fig. 13(b) demonstrates that it is capable of balancing the number of categories generated by each edge server. In the algorithm with E2C, the number of sample types generated by the server is almost the same. In addition, we find the sharing rate seems to be a benefit for some devices and a detriment for others. In our setting, the sharing rate is larger, the edge server adopts less of the global model. We argue that this is because some regions have significantly better sample quality and model state than others, and therefore using only the number of samples to define the aggregation weights of the model would subject these well-trained models to poor models. Consequently, using different sharing rates based on different generators on edge servers should be a good solution, but unlike defining the state of a discriminator, how to define the state of a generator in FedSGD-based GAN is an unknown problem.

7 Discussion

In this section, we discuss the limitations of CAP-GAN and the problems of distributed training of GANs that need to be addressed in the future.

(1) Asynchronous Mechanism: our approach consists of two parts, the D2E part is a synchronous mechanism, while the E2C part is asynchronous. A synchronous mechanism in D2E can ensure convergence of the model while sacrificing efficiency, which exists in all Federated SGD-based algorithms (MD-GAN and AC-GAN). Therefore, improving the structure of distributed algorithms for asynchronous training of GANs is a feasible study point in the future.

(2) Privacy Security: different from FL transmitting weights over the network, the servers in MD-GAN, AC-GAN, and CAP-GAN need to transmit the data generated by the generator to the user. During the later stages of training, even though the generated data is known as fake by discriminators, it already carries information about the user's data, making it a vulnerable target for attacks. Therefore, as a decentralized encryption and authentication algorithm, Blockchain is intuitively suitable for combining with CAP-GAN. Blockchain can provide security for generated data. These data are traceable in Blockchain due to they are generated in regular steps [34]. This is a meaningful research direction for the future.

(3) Load Balancing: the results of our experiment are based on only 10 devices and simulated on one computer. In practice, however, there are thousands of devices such as smartphones and IoTs in an area, which may cause pressure for one edge server. It is a challenge to consider how to share the pressure with the surrounding idle edge servers, which necessitates vitalizing the resources on the edge servers and taking into account the model consistency in these edge servers.

(4) Failure Tolerance: Li *et al.* [35] have pointed out that the failure rate of a task is higher when train models in more devices, which has a failure rate of 24% on 10,000 devices. It should also be considered fault-tolerant when trained GANs in a distributed manner, but since our focus in this paper is on the distribution of the dataset, we consider improving the fault tolerance of GANs trained in a distributed manner in the future.

(5) Mobility: these algorithms (including MD-GAN, AC-GAN, and CAP-GAN) only consider the problem in the case where the user does not move out the coverage. It is estimated that the number of mobile users at the edge is growing rapidly [36]. These studies [2, 3] have shown that GANs have the potential ability to assist vehicle trajectory predictions. Thus, considering a switch in models from one edge server to another presents a future challenge as well as an opportunity.

8 Conclusion

This work focuses on addressing the data distribution challenges when training GANs at the edge in order to make GANs usable in Non-IID scenarios. We propose a novel distributed algorithm called CAP-GAN that considers energy limits and geographical constraints. CAP-GAN employs a new aggregation scheme that allows the generators to obtain more accurate gradient update directions at each base station, avoiding falling into mode collapse. Moreover, federal learning is used in CAP-GAN to break down the information island phenomenon between base stations and aggregate global models and features. In addition, we introduce the Mix-G module to optimize its performance for addressing extremely dispersed data distributions. Mix-G adopts a novel structure, where the generator consists of shared and personality layers. Utilizing this structure can accelerate model convergence and enhance diversity, resulting in generated samples that better align with personalized

data. Through simulation experiments, the results have been demonstrated that CAP-GAN (combined with Mixed-G) is capable of generating all types of data and significantly improving data fitting in cases of non-independent identical distribution data, when compared to SOTA algorithms like AC-GAN, MD-GAN, and FL-GAN.

ACKNOWLEDGMENT

This work is supported in part by the Liaoning Province Applied Basic Research Program under Grant 2023JH2/101300194, and in part by the LiaoNing Revitalization Talents Program. Al-Dubai would like to acknowledge the support of the UK Engineering and Physical Sciences Research Council (EPSRC) programme grant: COG-MHEAR (Grant Reference: EP/T024917/1)

REFERENCES

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[2] L. Zhao, Y. Liu, A. Y. Al-Dubai, A. Y. Zomaya, G. Min, and A. Hawbani, "A novel generation-adversarial-network-based vehicle trajectory prediction method for intelligent vehicular networks," *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 2066–2077, 2020.

[3] F. Naeem, S. Seifollahi, Z. Zhou, and M. Tariq, "A generative adversarial network enabled deep distributional reinforcement learning for transmission scheduling in internet of vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4550–4559, 2021.

[4] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[5] M. Rasouli, T. Sun, and R. Rajagopal, "Fedgan: Federated generative adversarial networks for distributed data," *arXiv preprint arXiv:2006.07228*, 2020.

[6] R. Guerraoui, A. Guirguis, A.-M. Kermarrec, and E. L. Merrer, "Fegan: Scaling distributed gans," in *Proceedings of the 21st International Middleware Conference*, 2020, pp. 193–206.

[7] C. Hardy, E. Le Merrer, and B. Sericola, "Md-gan: Multi-discriminator generative adversarial networks for distributed datasets," in *2019 IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE, 2019, pp. 866–877.

[8] X. Zhang, X. Zhu, J. Wang, W. Bao, and L. T. Yang, "Dance: Distributed generative adversarial networks with communication compression," *ACM Transactions on Internet Technology (TOIT)*, vol. 22, no. 2, pp. 1–32, 2021.

[9] B.-C. Chen and A. Kae, "Toward realistic image compositing with adversarial learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8415–8424.

[10] K. Ehsani, R. Mottaghi, and A. Farhadi, "Segan: Segmenting and generating the invisible," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6144–6153.

[11] K. Armanious, C. Jiang, M. Fischer, T. Küstner, T. Hepp, K. Nikolaou, S. Gatidis, and B. Yang, "Medgan: Medical image translation using gans," *Computerized Medical Imaging and Graphics*, vol. 79, p. 101684, 2020.

[12] Y. Xue, T. Xu, H. Zhang, L. R. Long, and X. Huang, "Segan: adversarial network with multi-scale l1 loss for medical image segmentation," *Neuroinformatics*, vol. 16, no. 3, pp. 383–392, 2018.

[13] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[14] Y. Jiang, S. Chang, and Z. Wang, "Transgan: Two transformers can make one strong gan," *arXiv preprint arXiv:2102.07074*, 2021.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[16] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223.

[17] D.-J. Han, M. Choi, J. Park, and J. Moon, "Fedmes: Speeding up federated learning with multiple edge servers," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3870–3885, 2021.

[18] M. Blot, D. Picard, M. Cord, and N. Thome, "Gossip training for deep learning," *arXiv: Computer Vision and Pattern Recognition*, 2016.

[19] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2794–2802.

[20] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[22] Y. Liu, X. Yuan, Z. Xiong, J. Kang, X. Wang, and D. Niyato, "Federated learning for 6g communications: Challenges, methods, and future directions," *China Communications*, vol. 17, no. 9, pp. 105–118, 2020.

[23] I. Durugkar, I. Gemp, and S. Mahadevan, "Generative multi-adversarial networks," *arXiv preprint arXiv:1611.01673*, 2016.

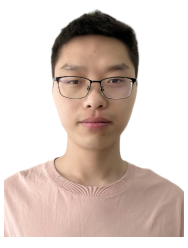
[24] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[25] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, 2020.

[26] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, "Exploiting shared representations for personalized federated learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 2089–2099.

[27] M.-Y. Liu and O. Tuzel, "Coupled generative adversar-

ial networks," *Advances in Neural Information Processing Systems*, vol. 29, 2016.
[28] A. Ghosh, V. Kulharia, V. P. Namboodiri, P. H. Torr, and P. K. Dokania, "Multi-agent diverse generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8513–8521.
[29] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in Neural Information Processing Systems*, vol. 30, 2017.
[30] S. Nowozin, B. Cseke, and R. Tomioka, "f-gan: Training generative neural samplers using variational divergence minimization," *Advances in Neural Information Processing Systems*, vol. 29, 2016.
[31] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in Neural Information Processing Systems*, vol. 29, 2016.
[32] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in Neural Information Processing Systems*, vol. 30, 2017.
[33] T. Che, Y. Li, A. P. Jacob, Y. Bengio, and W. Li, "Mode regularized generative adversarial networks," *arXiv preprint arXiv:1612.02136*, 2016.
[34] M. Ali, H. Karimipour, and M. Tariq, "Integration of blockchain and federated learning for internet of things: Recent advances and future challenges," *Computers Security*, vol. 108, p. 102355, 2021.
[35] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 583–598.
[36] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.



Jiaxin Zhang received the B.S. degree in Internet of Things Engineering from Shenyang Aerospace University, China. He is currently pursuing the master's degree with the School of Computer Science, Shenyang Aerospace University, China. His research interests mainly include GAN, federated learning, knowledge distillation, edge intelligence, and MEC.

Liang Zhao (Member, IEEE) is a Professor at Shenyang Aerospace University, China. He received his Ph.D. degree from the School of Computing at Edinburgh Napier University in 2011. Before joining Shenyang Aerospace University, he worked as associate senior researcher in Hitachi (China) Research and Development Corporation from 2012 to 2014. He is also a JSPS Fellow. His research interests include ITS, VANET, WMN and SDN. He has published more than 150 articles. He served as the Chair of several international conferences and workshops, including 2022 IEEE BigDataSE (Steering Co-Chair), 2021 IEEE TrustCom (Program Co-Chair), 2019 IEEE IUCC (Program Co-Chair), and 2018-2022 NGDN workshop (founder). He is Associate Editor of Frontiers in Communications and Networking and Journal of Circuits Systems and Computers. He is/has been a guest editor of IEEE Transactions on Network Science and Engineering, Springer Journal of Computing, etc. He was the recipient of the Best/Outstanding Paper Awards at 2015 IEEE IUCC, 2020 IEEE ISPA, 2022 IEEE EUC and 2013 ACM MoMM.

Keping Yu received the M.E. and Ph.D. degrees from the Graduate School of Global Information and Telecommunication Studies, Waseda University, Tokyo, Japan, in 2012 and 2016, respectively. He was a Research Associate, Junior Researcher, Researcher with the Global Information and Telecommunication Institute, Waseda University, from 2015 to 2019, 2019 to 2020, 2020 to 2022, respectively. He is currently an Associate Professor of the Graduate School of Science and Engineering, Hosei University, Tokyo, Japan. His research interests include smart grids, information-centric networking, the Internet of Things, artificial intelligence, blockchain, and information security.

Geyong Min is a Professor of High Performance Computing and Networking in the Department of Computer Science within the College of Engineering, Mathematics and Physical Sciences at the University of Exeter, United Kingdom. He received the PhD degree in Computing Science from the University of Glasgow, United Kingdom, in 2003, and the B.Sc. degree in Computer Science from Huazhong University of Science and Technology, China, in 1995. His research interests include Computer Networks, Wireless Communications, Parallel and Distributed Computing, Ubiquitous Computing, Multimedia Systems, Modelling and Performance Engineering.

Ahmed Y. Al-Dubai is Professor of Networking and Communication Algorithms in the School of Computing at Edinburgh Napier University, UK. He received the PhD degree in Computing from the University of Glasgow in 2004. His research interests include Communication Algorithms, Mobile Communication, Internet of Things, and Future Internet. He received several international awards.

Albert Y. ZOMAYA is the Peter Nicol Russell Chair Professor of Computer Science and Director of the Centre for Distributed and High-Performance Computing at the University of Sydney. To date, he has published $>$ 700 scientific papers and articles and is (co-)author/editor of$>$30 books. A sought-after speaker, he has delivered > 250 keynote addresses, invited seminars, and media briefings. His research interests span several areas in parallel and distributed computing and complex systems. He is currently the Editor in Chief of the ACM Computing Surveys and served in the past as Editor in Chief of the IEEE Transactions on Computers (2010-2014) and the IEEE Transactions on Sustainable Computing (2016-2020). He is a Fellow of the IEEE, the American Association for the Advancement of Science, the Australian Academy of Science, Royal Society of New South Wales, and a Foreign Member of Academia Europaea.