Federated Generative Model on Multi-Source Heterogeneous Data in IoT

Zuobin Xiong, Wei Li, and Zhipeng Cai[*]
[1]Department of Computer Science, Georgia State University
Atlanta, Georgia, 30303 USA
zxiong2@student.gsu.edu, {wli28, zcai}@gsu.edu

Abstract

The study of generative models is a promising branch of deep learning techniques, which has been successfully applied to different scenarios, such as Artificial Intelligence and the Internet of Things. While in most of the existing works, the generative models are realized as a centralized structure, raising the threats of security and privacy and the overburden of communication costs. Rare efforts have been committed to investigating distributed generative models, especially when the training data comes from multiple heterogeneous sources under realistic IoT settings. In this paper, to handle this challenging problem, we design a federated generative model framework that can learn a powerful generator for the hierarchical IoT systems. Particularly, our generative model framework can solve the problem of distributed data generation on multi-source heterogeneous data in two scenarios, *i.e.,* feature related scenario and label related scenario. In addition, in our federated generative models, we develop a synchronous and an asynchronous updating methods to satisfy different application requirements. Extensive experiments on a simulated dataset and multiple real datasets are conducted to evaluate the data generation performance of our proposed generative models through comparison with the state-of-the-arts.

Introduction

Generative models have become a popular research direction in the field of deep learning thanks to its compelling ability to generate realistic data plausibly drawn from an existing data distribution. Different from discriminative models, a generative model can generate unlimited synthetic data to fulfill expected tasks by using the trained generator once it has been trained. So far, the breakthrough brought by generative models has rapidly produced a revolutionary impact on different fields, and this impact has already flourished in various real applications in the Internet of Things (IoT). In IoT environments, a variety of devices are interconnected to generate, collect, share, and process heterogeneous data for data-driven applications. With the help of the generative models, the generated data can facilitate the procedures of data collection and process in different aspects, such as reducing tedious data collection time, imputing missing data, augmenting data quality, and detecting abnormal samples. In (Lv et al. 2018; Tschuchnig, Ferner, and Wegenkittl 2020; Chen et al. 2021; Mohammadi, Al-Fuqaha, and Oh 2018), generative models have been used for traffic data generation, traffic modeling, traffic prediction, and traffic control in smart cities. Generative Adversarial Networks (GANs)-based models have been designed to generate and analyze medical data in smart medication systems (Vaccari et al. 2021; Chang et al. 2020; Qu et al. 2020). In video surveillance systems, many generative models are designed to support object recognition (Fabbri, Calderara, and Cucchiara 2017), movement capture (Shi, Liu, and Li 2017), anomaly detection (Ganokratanaa, Aramvith, and Sebe 2019), and super resolution tasks (Lee et al. 2018).

Yet, most of the existing works implement the centralized generative models, which first collect data from IoT devices to a central server and then train generative models to achieve generation goals. These centralized generative models may be vulnerable to the issues of single point failure and privacy leakage. Moreover, users' willingness to share data with a central server may be declined because of their privacy concerns, increasing the difficulty in data collection and hindering the further development of IoT applications. On the other hand, transmitting such a massive amount of data to a central server brings expensive communication cost to IoT. To break down the obstacles of privacy concerns and communication cost, designing distributed generative models should be a better solution. The integration of distributed generative models into IoT can benefit individuals and society in many aspects. For examples, in smart health, distributed generative models can be used to synthesize the pattern of a special tumor's MRI/CT scanning result without privacy violation, where the synthetic data can be used for data visualization and training dataset for other tasks; due to the existence of unaccessible data, pre-training a distributed generative model to collect features from underlying datasets can help model developers understand the dataset, such as checking the data sanity, detecting bias in the dataset, and debugging misclassified samples (Augenstein et al. 2019); and when multiple datasets on IoT devices are non-i.i.d. distributed, learning a distributed generative model can learn a mixed distribution and generate data with more diversity (Li et al. 2019).

Currently, only limited works have developed the distributed generative models but overlook the following cru-

[*]Corresponding Author.

cial issues in practical IoT scenarios: (i) most of the existing works adopt the federated learning style that needs to upload large-size model parameters, which burdens limited network resources; (ii) all of the existing works mainly focus on i.i.d. data for model training without studying non-i.i.d. data scenarios; and (iii) no work considers the heterogeneity of data domain in different IoT devices. Therefore, how to design a distributed generative model to realize data generation effectively and efficiently in IoT is still a challenging problem.

To solve this problem, we design a novel distributed generative model framework, taking into account the essential properties of IoT devices, including wide geographic distribution, low computation power, non-i.i.d. data, and heterogeneous data domains. Considering the data distribution and correlation in IoT devices under different applications, the problem of distributed data generation is studied in two scenarios: (i) **feature related scenario**, where the data of different communities has the same features but different labels; and (ii) **label related scenario**, where the data of different communities has the same labels but different features. The major contributions of this paper are summarized below.

- Based on the features of IoT applications, we design a three-layer hierarchical framework to deploy federated generative models, which is the first work to consider multi-source heterogeneous data for distributed data generation, to our best knowledge.
- According to the data scenarios in real IoT applications, we propose two generative models for multi-source data generation under our proposed hierarchical framework.
- We devise synchronous and asynchronous updating strategies for training the generators on the edges, which can be adopted by different application requirements.
- Intensive experiments are conducted on different datasets from multiple data domains, which can illustrate the performance of our data generation models compared with the state-of-the-arts.

Related Works

Generative adversarial networks (GANs) (Goodfellow et al. 2014) originally focused on data generation in a single dataset. In multi-source data scenarios, two or more data sources are provided to train variants of GANs, which is studied in conditional generation and joint generation. The conditional generation methods aim at learning a conditional distribution of one data source given other data source(s) as the additional information (Zhu et al. 2017; Isola et al. 2017). Differently, the joint generation methods try to learn the joint distribution of multiple data sources (Yoon, Jordon, and Schaar 2018; Pu et al. 2018). However, existing generative models on multiple data sources are centralized mode, which limits their applicability on distributed data silos.

For **generative models in distributed setting**, Generative Adversarial Parallelization (Im et al. 2016) is the first work applying GANs in a distributed setting, where each GAN model is trained on a distributed dataset. Similar idea is adopted by Hardy *et al.* (Hardy, Le Merrer, and Sericola 2018) in Gossip GAN, but the generators and discriminators are gossiped among neighborhood for adapting the global distribution. Durugkar *et al.* (Durugkar, Gemp, and Mahadevan 2017) and Hardy *et al.* (Hardy, Le Merrer, and Sericola 2019) simplify the previous structure via changing multiple generators to one while keeping multiple discriminators for training, which can provide the aggregated discriminator loss to the generator for better generation quality. Along with this line, in (Yonetani et al. 2019), authors consider the distributed data generation on multiple datasets, where the multiple discriminators are weighted averaged according to their loss values. Since the federated learning (Brendan McMahan et al. 2016) paradigm was proposed, it has been leveraged to develop various GANs, in which the minor difference among these methods is whether to aggregate generators (Triastcyn and Faltings 2019), discriminators (Augenstein et al. 2019), or both (Fan and Liu 2020; Rasouli, Sun, and Rajagopal 2020; Xin et al. 2020). However, a high communication cost for these generative models is transmitting model parameters iteratively. To circumvent, reformed methods in (Chang et al. 2020) and (Qu et al. 2020) set a central generator and update it with the loss value of local discriminators, where the data volume transmitted between server and local clients is much smaller than transferring models. But their solutions are not specified for non-i.i.d data, lacking a solid ground for landing practical applications. Above methods work well on a single data source but are not applicable on multiple heterogeneous data sources.

System Framework

As shown in Fig. 1, our federated generative model framework is built as a three-layer hierarchical structure consisting of IoT devices at bottom layer, edge servers in the middle layer, and a cloud server at the top layer. Particularly, each edge server is co-located at a base station covering a local region, in which the edge server and the covered IoT devices form a local community. For the purpose of data generation, a discriminator is deployed in every IoT device, a community generator is deployed in every edge server, and a global generator is deployed in the cloud server. Let $\mathbb{K}$ be the set of local communities, G_k be the community generator of community $k \in \mathbb{K}$, $\mathbb{J}_k$ be the set discriminators in community k, and D_{kj} be the discriminator of IoT device $j \in \mathbb{J}_k$ in community k.

The training process of our federated generative framework consists of two stages: (i) local community training for updating G_k at the edge servers, and (ii) global aggregation for updating G at the cloud server. In local community training, the IoT devices and the edge-side generators are trained to capture homogeneous data distribution; and then in global aggregation, the local trained results are transmitted to the cloud to obtain the global data distribution. The global aggregation is performed by I_{gl} iterations, each of which contains I_{lo} iterations of local community training. Due to the closely distributed locations, the data within a local community usually has the same features (*i.e.,* feature related scenario) or class labels (*i.e.,* label related scenario).

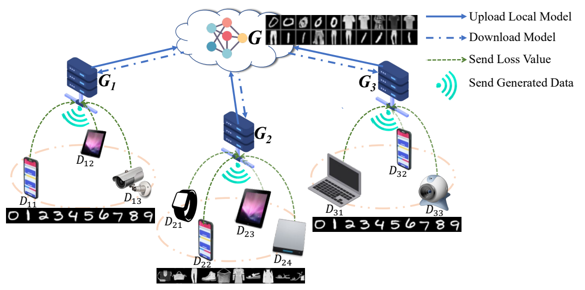


Figure 1: Example of the feature related data generation.

Feature Related Data Generation

In the feature related scenario, our objective is to learn a mixed data distribution of all communities via the global generator G in the cloud server, where the datasets from different communities have the same features but different labels. As shown in the illustrative example of Fig. 1, the three communities have the same feature (*i.e.*, gray scale pixel value in [0, 1]) and different data labels (*i.e.,* digit labels in $\{0, 1, \ldots, 9\}$ and cloth labels in {t-shirts, shoes,. . . , pants}). Finally, the global generator in the cloud server can generate the complete dataset with all labels including digits and clothes. This scenario commonly exists in real IoT applications. For example, medical data of a surgical hospital and a cancer hospital may cover different disease classes. IoT sensors in different locations can sense different air quality indices. Generating data from these different datasets through our framework can get a complete research dataset. The design of the generators and the discriminators, and their interactions is detailed in the following.

Generator

In our framework, there are two types of generators, *i.e.*, the community generator G_k and the global generator G. Similar to the original GANs, the community generator G_k produces two batches of data $G_k(z_d)$ and $G_k(z_g)$ with latent vector z_d and z_g drawn from $\mathcal{N}(0, 1)$. Then, these data is sent to local IoT devices, where $G_k(z_d)$ is served as fake data to update D_{kj} as shown in Eq. (2), and $G_k(z_g)$ is employed to calculate the loss function value $\mathcal{L}_{D_{kj}}(G_k)$ of G_k on D_{kj} for back propagation as shown in Eq. (3). In the cloud server, the global generator G has the same network structure as G_k, and its parameters are aggregated from all the community generators via Eq. (1).

$$G = \sum_{k \in \mathbb{K}} \frac{\exp(\mathcal{L}_{D_*}(G_k))}{\sum_{k' \in \mathbb{K}} \exp(\mathcal{L}_{D_*}(G_k))} G_k, \tag{1}$$

where the loss function $\mathcal{L}_{D_*}(G_k)$ can be computed by either $\mathcal{L}_{D_{sync}}$ in Eq. (4) or $\mathcal{L}_{D_{async}}$ in Eq. (5) as specified in Synchronous Updating and Asynchronous Updating.

Discriminator

Corresponding to community generator G_k, in each iteration of the local community training, the discriminator D_{kj} performs two-step operations: (i) discriminator updating and (ii) loss function computation. At the first step, D_{kj} is trained via the IoT device's real local data x that follows distribution $p_{kj}(x)$ and the fake data $G_k(z_d)$ that is generated by G_k, which is expressed in Eq. (2).

$$\begin{aligned} \max_{D_{kj}} \mathcal{L}(D_{kj}) = &\mathbb{E}_{x \sim p_{kj}(x)}[\log D_{kj}(x)] + \\ &\mathbb{E}_{z_d \sim p_z(z)}[\log(1 - D_{kj}(G_k(z_d)))]. \end{aligned} \tag{2}$$

After D_{kj} is updated, $G_k(z_g)$ is input to D_{kj} to calculate the loss function value of G_k as follows,

$$\mathcal{L}_{D_{kj}}(G_k) = \mathbb{E}_{z_d \sim p_z(z)}[\log(1 - D_{kj}(G_k(z_g)))]. \tag{3}$$

Next, the loss value $\mathcal{L}_{D_{kj}}(G_k)$ is sent back to G_k for the purpose of calculating gradients and updating G_k.

Updating Strategy

Considering the diversity of IoT devices (*e.g.,* laptops and smart phones) and the requirement of IoT applications, either synchronous or asynchronous methods can be adopted to update G_k during our local community training process.

Synchronous Updating In the synchronous manner, G_k is updated after collecting the loss value $\mathcal{L}_{D_{kj}}(G_k)$ from all the local discriminators within the community k during each updating period, where the length of a unit updating period can be pre-determined by the system. Unlike traditional aggregation methods (Chang et al. 2020; Qu et al. 2020) that adopt average aggregation of all collected values, our synchronous aggregation can work well when the local distribution $p_{kj}(x)$ is non-i.i.d. The aggregated loss $\mathcal{L}_{D_{sync}}(G_k)$ is computed as follows:

$$\mathcal{L}_{D_{sync}}(G_k) = \sum_{j \in \mathbb{J}_k} \frac{e^{\mathcal{L}_{D_{kj}}(G_k)}}{\sum_{j' \in \mathbb{J}_k} e^{\mathcal{L}_{D_{kj}}(G_k)}} \mathcal{L}_{D_{kj}}(G_k). \tag{4}$$

In Eq. (4), a larger value of $\mathcal{L}_{D_{kj}}(G_k)$ means the generated data $G_k(z_d)$ is more realistic as judged by D_{kj}, thus a bigger weight is assigned to $\mathcal{L}_{D_{kj}}(G_k)$ for aggregation. The effectiveness of such an exponential weighted aggregation method has been verified in the existing works (Yonetani et al. 2019) on non-i.i.d. data. After $\mathcal{L}_{D_{sync}}(G_k)$ is obtained, the community generator G_k can be updated accordingly through the gradient ascending method.

Asynchronous Updating Since the IoT devices may have different capacities, such as computation rate and transmission power, there might be some stragglers reducing the time efficiency of generator updating in the synchronous manner. To overcome this weakness, we propose an asynchronous aggregation method to update G_k.

Basically, in the asynchronous manner, G_k is updated immediately once a loss function value is received from a local discriminator so that there is no need to wait the slowest discriminators for updating. Notice that with respect to G_k, the received loss function value with a smaller degree of staleness has more contribution to accurate updating. By considering the impact of reception time on the updating performance, the "staleness" of $\mathcal{L}_{D_{kj}}(G_k)$ is defined to be $s_{kj} = T_{kj}^{rec} - T_k^{lat}$, where T_{kj}^{rec} is the time when $\mathcal{L}_{D_{kj}}(G_k)$ is received by G_k, and T_k^{lat} is the latest updating time of G_k. Accordingly, G_k can be updated with obtained loss function values asynchronously through Eq. (5).

$$\mathcal{L}_{D_{async}}(G_k) = \frac{1}{|\mathbb{M}_k|} \sum_{j \in \mathbb{M}_k} e^{-s_{kj}} \mathcal{L}_{D_{kj}}(G_k), \tag{5}$$

where $\mathbb{M}_k$ is the set of local IoT devices whose loss function values are received by G_k at the same time. As indicated in Eq. (5), a smaller degree of staleness is assigned a bigger weight on the aggregated loss value for updating G_k.

Training Process

At the beginning of the global training process, the parameters of G, G_k, and all the discriminators are initialized. In each global training iteration, G_k is updated via I_{lo} iterations of the local community training and then is sent to the cloud server for aggregation. Specifically, in each community k, G_k generates two batches of data and sends them to the discriminators within the same local community. The discriminators update weight parameters based on $G_k(z_d)$ and calculate the loss values based on $G_k(z_g)$ as feedback to G_k. After this step, the loss value $\mathcal{L}_{D_{kj}}(G_k)$ is sent to the edge servers for updating G_k, where the aggregated loss can be $\mathcal{L}_{D_{sync}}$ in the synchronous updating method or $\mathcal{L}_{D_{async}}$ in the asynchronous updating method, which can be predetermined by application requirements. After local community training stops, G_k is sent to the cloud server for the global aggregation that follows Eq. (1). At the end of each global training iteration, G is distributed back to local communities, preparing for the next iteration of global training process. After I_{gl} iterations of the global training process, G is output for feature related data generation. Obviously, when I_{gl} (*resp.* I_{lo}) is increased, the training result of G (*resp.* G_k) will become better with a longer training duration. Therefore, I_{gl} and I_{lo} can be set by considering the trade-off between training result and training time.

Label Related Data Generation

In the label related scenario, the distributed datasets in different communities have different feature distribution but the same labels; that is, the data in different communities belongs to different domains. Generally, the label related scenario can help applications that need data from distributed storage for domain adaption problem (Long et al. 2015). In vehicular networks, camera data captured by vehicles in one community and LiDAR data in another community may have the same object labels (Xiong et al. 2021), which is similar to our label related scenario.

Take Fig. 2 as an example, where the data in the three communities shares the same digit labels $\{0, 1, 2, \ldots, 9\}$ but differs in data features, such as background color, and luminance. The global generator G in the cloud aims at generating data from different domains with the same label. To this end, a domain classifier C is added in the cloud server to distinguish the original source of the generated data under our framework as presented in Fig. 2. More details of generators, discriminators, domain information, and training process are introduced as follows.

Generator

In the label related scenario, the generator is expected to generate multi-domain data distribution under various domain conditions, where in each community k, the domain condition, denoted by c_k, indicates the origin of

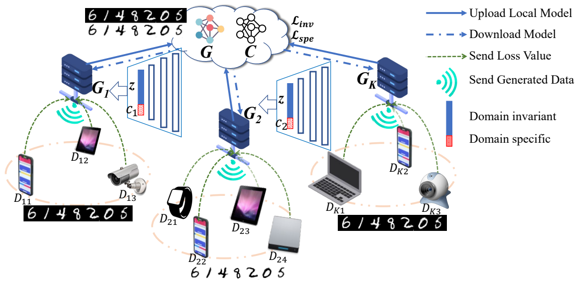


Figure 2: Example of the label related data generation.

the generated data. To achieve this purpose, the model of cGANs (Mirza and Osindero 2014) is adopted to embed the latent vector z as the domain invariant of all data domains (*e.g.,* the common class labels in $\{0, 1, 2, \ldots, 9\}$) and to embed the domain condition c_k as the domain specific of each data domain (*e.g.,* background color, and luminance).

Specifically, the community generator G_k takes the latent vector z and the domain specific condition c_k as its inputs for data generation. During the local community training process, two batches of fake data, $G_k(z_d, c_k)$ and $G_k(z_g, c_k)$, are generated by G_k and sent to D_{kj} to update the discriminator and to calculate the loss function value $\mathcal{L}_{D_{kj}}(G_k)$ for back propagation. In the cloud server, the global generator G has the same structure as G_k and is also aggregated by community generators as shown in Eq. (1).

Discriminator

The discriminators are set with two major operations, including discriminator update and loss function computation.

For discriminator update, D_{kj} is trained by the real data, $x \sim p_{kj}(x)$, and the generated data, $G_k(z_d, c_k)$, through Eq. (6), which is the same as the training process of cGANs

$$\begin{aligned}\max_{D_{kj}} \mathcal{L}(D_{kj}) =& \mathbb{E}_{x\sim p_{kj}(x)}[\log D_{kj}(x|c_k)]+ \\ & \mathbb{E}_{z\sim p_z(z)}[\log(1 - D_{kj}(G_k(z_d, c_k)|c_k))].\end{aligned} \tag{6}$$

After D_{kj} is updated, the loss function value of G_k, $\mathcal{L}_{D_{kj}}(G_k)$, is calculated based on $G_k(z_g, c_k)$ as follows,

$$\mathcal{L}_{D_{kj}}(G_k) = \mathbb{E}_{z\sim p_z(z)}[\log(1 - D_{kj}(G_k(z_g, c_k)|c_k))]. \tag{7}$$

Then, $\mathcal{L}_{D_{kj}}(G_k)$ is sent back to edge server to update G_k. The updating process of G_k in the label related scenario can be performed using either synchronous or asynchronous updating methods as feature related scenario does.

Domain Invariant & Specific

The same class labels of all communities' data is so-called "domain invariant" that normally exists in high-level representation space (Mao and Li 2018; Wang and Deng 2018). For a generator that maps latent vectors to high-dimension space, the output of its first layer denotes its high-level representation. The distance between the first-layer output of G under two different domain conditions can be defined as the "domain invariant loss" as shown in Eq. (8), which can guide the generator to produce the required high-level representation.

$$\mathcal{L}_{inv}(G) = \sum_{k\neq k'\in\mathbb{K}} \|G^{1st}(z, c_k) - G^{1st}(z, c_{k'})\|_2. \tag{8}$$

Since the global generator G is obtained on cloud server, with the domain condition c_k ($k \in \mathbb{K}$), G can generate $G(z, c_k)$ falling into different data domains. Accordingly, we can obtain the corresponding first-layer output, $G^{1st}(z, c_k)$, to calculate the domain invariant loss via Eq. (8), where L_2 norm is adopted to measure the difference in the high-level representation space. Through minimizing this distance, the generated result $G(z, c_k)$ with different domain conditions are forced to possess the same information with respect to z (*i.e.*, the same labels).

Domain specific is the special information that belongs to each domain. Here, the different domain condition c_k is used to represent domain specific. With the domain condition c_k, the generated data $G(z, c_k)$ is actually self-labeled, and thus can be used as the training dataset to fulfill a supervised domain classifier training. The domain classifier C in the cloud server is modeled as a multi-class classifier that predicts the generated data to corresponding data domains. Concretely, C is constructed as a neural network with considering cross entropy loss and is trained on the labeled dataset $\{G(z, c_k), c_k | k \in \mathbb{K}\}$, where c_k is used as the domain label. During the evaluation phase of C, we can use the loss value of C as the domain specific loss of G, which is computed in Eq. (9):

$$\mathcal{L}_{spe}(G) = \sum_{k \in \mathbb{K}} H(C(G(z, c_k)), c_k), \tag{9}$$

where $\mathcal{L}_{spe}(G)$ denotes the domain specific loss, and $H(\cdot, \cdot)$ represnts the cross entropy measurement. Since G is expected to produce more realistic data based on the domain condition c_k, minimizing $\mathcal{L}_{spe}(G)$ can force G to embed domain specific semantics into c_k.

Training Process

In each global training iteration, all local communities proceed their local community training to update G_k with I_{lo} iterations. During each location training iteration, two batches of generated data are sent to the local IoT devices to update discriminators and calculate loss function values. Then, the calculated loss values are transmitted to the edge servers for updating G_k based on Eq. (4) in a synchronous manner or Eq. (5) in an asynchronous manner, which can be decided by the system requirements. After G_k is trained and received by the cloud server, aggregation is performed to obtain G as shown in Eq. (1). Then, we can train the domain classifier C based on the dataset $\{G(z, c_k), c_k | k \in \mathbb{K}\}$ generated by G, and calculate the domain invariant loss $\mathcal{L}_{inv}(G)$ and domain specific loss $\mathcal{L}_{spe}(G)$. Thereafter, G, $\mathcal{L}_{inv}(G)$, and $\mathcal{L}_{spe}(G)$ are sent back to community generators for minimizing $\lambda_1 \mathcal{L}_{inv}(G) + \lambda_2 \mathcal{L}_{spe}(G)$ and next local training iteration. Particularly, $\lambda_1, \lambda_2 \in (0, 1]$ are two pre-determined hyperparameters used as the weights of loss functions. When the global training process ends, G is output to generate multi-domain data with the domain condition c_k.

Experiments

In this section, we investigate the quality of the generated data from both visualization and statistic aspects through comparison with the state-of-the-art models.

Setting	Metrics	Cent. GAN	Fed. GAN	M-D GAN	FR (sync)	FR (async)
i.i.d.	IS	3.386	4.215	4.228	**4.363**	3.208
	FID	19.84	17.14	14.69	**13.82**	52.24
	Accu.	82.21%	87.84%	90.60%	**90.65%**	53.17%
non-i.i.d	IS	3.386	2.281	2.839	**3.875**	2.922
	FID	19.84	74.80	96.12	26.15	77.18
	Accu.	82.21%	42.37%	45.22%	**86.05%**	51.20%

Table 1: Quantitative comparison of different models on MNIST and Fashion-MNIST in feature related scenario.

Experiment Settings

System Structure. The network environment is simulated on one server, where each community has 1 edge server and 5 IoT devices. Considering the available real datasets, the number of community varies with the scenario settings, which is demonstrated as follows.

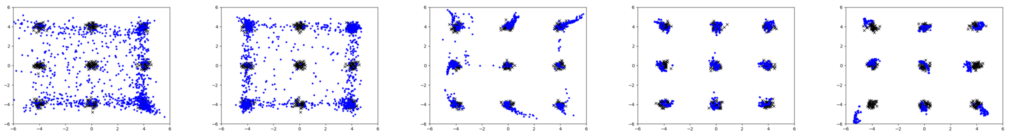
Datasets. For the feature related scenario, we choose two types of data in the experiments: (i) simulated gaussian mixed data, where the datasets of all local communities have the same feature (*i.e.,* variance) but different mean values; and (ii) MNIST dataset and Fashion-MNIST dataset, where both datasets have the same features but different class labels. For the label related scenario, we select (i) MNIST dataset and inverse MNIST dataset; (ii) sketch-photo dataset (Zhu et al. 2016), each of which has the same labels or semantic information but different domains.

Data Distribution. Both the i.i.d. data distribution and the non-i.i.d. data distribution are considered for the feature related and the label related scenarios in our experiments. Accordingly, by combining the two types of data distribution and the two scenarios, we have the following four types of settings for evaluation: (i) feature related scenario with i.i.d. data distribution, (ii) feature related scenario with non-i.i.d. data distribution, (iii) label related scenario with i.i.d. data distribution, and (iv) label related scenario with non-i.i.d. data distribution.

Baselines. Since our distributed generative models in the feature related and the label related scenarios are designed with different ideas and goals, we pick different baselines for the two scenarios. In the feature related scenario, centralized GAN (Cent. GAN), federated GAN (Fed. GAN) (Fan and Liu 2020), and multi-discriminator GAN (M-D GAN) (Chang et al. 2020) are used for performance comparison. In the label related scenario, RegGAN (Mao and Li 2018) (a centralized model to generate multi-domain data) and a federated GAN (FED GAN) model are selected as the baseline models.

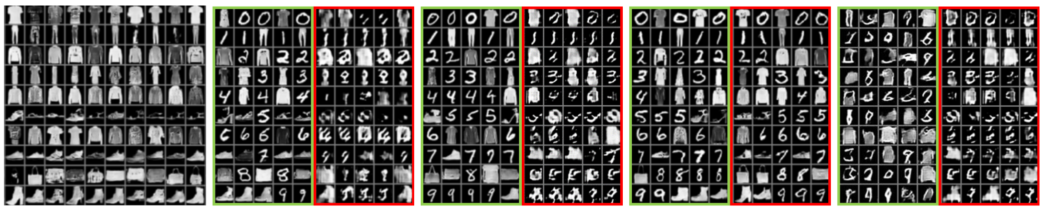
Feature Related Data Generation

Fig. 3 displays the generated data of baselines and our feature related (FR) data generation model with synchronous and asynchronous updating methods, in which the black "$\times$" markers denote the training data points coming from 9 communities, and the blue "$\cdot$" markers are the generated data points. Thus, if the blue "$\cdot$" can concentrate to the black "$\times$" points closer, the generated data is more similar to the real data, indicating the better performance of generative models.



(a) Cent. GAN. (b) Fed. GAN. (c) M-D GAN. (d) FR (sync). (e) FR (async).

Figure 3: Generated data of different generative models on the simulated gaussian mixed dataset.



(a) Cent. GAN. (b) Fed. GAN. (c) M-D GAN. (d) FR (sync). (e) FR (async).

Figure 4: Generated data of different generative models on MNIST and Fashion-MNIST dataset in i.i.d./non-i.i.d. setting.

In Fig. 3(a), the generated data mainly concentrates around one black data cluster at the bottom right corner, which means only partial data distribution of entire training dataset is learned by centralized GAN. Centralized GAN cannot perform well on such gaussian mixed data due to mode collapse. Fed. GAN averagely aggregates all the involved generators and discriminators on the server, producing the data that mainly concentrates around the black data clusters at the four corners. M-D GAN uses one generator to receive feedbacks from all discriminators for updating, which is a good way to learn the multi-source data distribution. But, as shown in Fig. 3(c), it also generates a number of outliers that are far away from the black data points, because the discriminators work alone without information exchange among all communities. Our FR data generation model is like a combination of Fed. GAN and M-D GAN, within each community, the GAN structure resembles to M-D GAN but uses an exponentially weighted aggregation; and in the cloud server, all community generators are exponentially weighted to obtain the global generator G so that the information among communities can be exchanged through distributing G back to communities. From Fig. 3(d), we find that almost all the generated data can concentrate on the 9 black data clusters accurately, reflecting the best generation performance when the synchronous updating method is used in our FR data generation model. While in Fig. 3(e), the majority of the generated data can concentrate on the 9 black data clusters closely, reaching a slightly worse generation performance than our FR (sync) data generation but is still better than other baselines.

For experiments on MNIST and Fashion-MNIST datasets, one community holds MNIST data and the other two hold Fashion-MNIST data. Inception Score (IS) (Salimans et al. 2016), Frechet Inception Distance (FID), and classification accuracy on generated data are the common

	Setting	**i.i.d.**			**non-i.i.d.**		
	RegGAN	FED GAN	LR (sync)	LR (async)	FED GAN	LR (sync)	LR (async)
IS	**4.037**	3.360	3.971	3.041	2.651	3.495	2.452
FID	**26.11**	33.20	28.56	41.35	53.26	45.22	59.82

Table 2: Quantitative comparison of different models on MNIST and Inverse MNIST in label related scenario.

metrics used to measure data generation performance. The generated images on i.i.d data (marked in green square) and non-i.i.d. data (marked in red square) are presented in Fig. 4. From the i.i.d. data of Fig. 4, we can get similar conclusion as in gaussian mixed data. Centralized GAN only generates blurred partial Fashion-MNIST data, failing to generate data from the whole dataset because of model collapse. Fed. GAN generates almost all labels, but some data is blurred and unclear, such as the 3rd and 5th rows in Fig. 4(b), because the average aggregation strategy used in Fed. GAN may not able to collect all important feedbacks from local entities. Fig. 4(c) and Fig. 4(d) show that M-D GAN and our FR (sync) data generation present similar results, and the result of Fig. 4(e) is slightly worse.

For non-i.i.d. data, by comparing the results of the three distributed generation models in Fig. 4 and Table 1, it is clear that our FR (sync) data generation model can achieve the best generation performance with non-i.i.d. data. Especially, the performance improvement of our FR (sync) generation model is more significant under non-i.i.d. setting.

Label Related Data Generation

For the data generation problem in the label related scenario, this paper is the first to propose distributed generative models, so there is no existing baseline. Instead, we build a distributed baseline model following the idea of federated

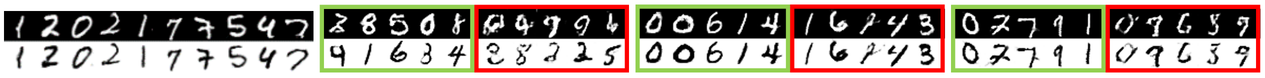

(a) RegGAN. (b) Federated GAN. (c) LR data generation (sync). (d) LR data generation (async).

Figure 5: Generated data of different generative models on MNIST and inverse MNIST dataset in i.i.d./non-i.i.d. setting.

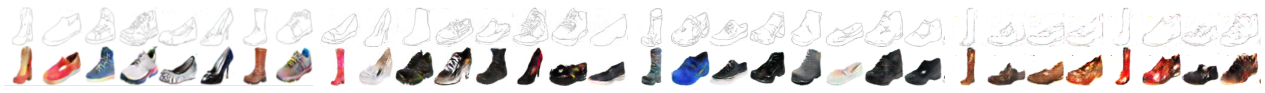

(a) RegGAN. (b) Federated GAN. (c) LR data generation (sync). (d) LR data generation (async).

Figure 6: Generated data of different generative models on (sketch-photo) Shoe dataset.

GAN (Fan and Liu 2020) for comparison. The structure of federated GAN (FED GAN) contains a cloud server and five local IoT devices. During each training iteration, each local device trains a GAN model and upload its generator and discriminator to the cloud server for aggregation. Then, the aggregated generator and discriminator are distributed back to all IoT devices for the next training iteration.

The generated data of the MNIST and inverse MNIST datasets are shown in Fig. 5, where the result of i.i.d. data is in green square, and non-i.i.d.data is in red square. RegGAN outputs the best generated results compared with the remaining three distributed models because it is a centralized model. Among these three distributed models, the results of FED GAN and our LR (sync) data generation have similar visual quality and statistical performance, which are clear and close to the generated results of RegGAN. On the other hand, our LR (async) data generation performs a little bit worse than FED GAN and LR (sync) data generation. Notably, the significant difference between our LR (sync) data generation and FED GAN is whether the generated data in the same column has the same domain invariant. In Fig. 5(a), Fig. 5(c), and Fig. 5(d), the generated data in each column always has the same class label. The reason that our LR data generation models can produce the same class label is that we use the domain classifier to minimize the domain invariant loss, which embeds the class label information into a domain invariant vector z. But, in Fig. 5(b), the domain invariant (*i.e.*, class label) of two images in the same column are not identical. FED GAN does not deploy any domain classifiers in the cloud server, and thus it can not guarantee the same class label with z during data generation. Also, for the generated data quality of our LR data generation model with different updating strategies, the synchronous method performs better than the asynchronous method, which is the same as the generation performance in the feature related scenario. Similar results on Shoes datasets can be found in Fig. 6, where the generated data of our LR data generation model with different update methods have the same semantic information in each column, but the generated data of FED GAN does not. More statistical results about the generated data are presented in Table 2.

The experiments of label related data generation with non-i.i.d. data are conducted on the MNIST dataset and the inverse MNIST dataset and shown in red box of Fig. 5. Compared with FED GAN, our LR (sync) data generation can still produce clear numbers from two different domains as shown in Fig. 5(c), which is closer to the results of RegGAN and is better than the results of FED GAN in Fig. 5(b). Similar to the results in i.i.d. setting, the generated data quality of our LR (async) data generation is still a little bit worse. But from the view point of domain invariant, our LR data generation model with different update methods can produce the same labels in each column. Yet, in Fig. 5(b), the generated data of FED GAN in the same column still has different labels, which demonstrate the effectiveness of the domain invariant loss function in our LR data generation models. The above observations validate that our LR data generation model can work on both i.i.d. and non-i.i.d. data with different update methods. In addition, more quantitative results of label related data generation in non-i.i.d. setting are provided in Table 2, where the data quality of our LR data generation model with synchronous update is better than FED GAN and very close to RegGAN even in such non-i.i.d. setting.

Conclusion

In this paper, we aim to solve the problem of distributed data generation with multiple heterogeneous data sources in feature related scenario and label related scenario. To this end, we design a hierarchical federated generative framework with the consideration of IoT features, including geographic distribution, low computation power, non-i.i.d. data, and heterogeneous data domains. Based on this framework, feature related data generation model and label related data generation model are proposed, which can solve the above data generation problems successfully in a synchronous manner or an asynchronous manner. Experiments are conducted on multiple datasets under different distribution settings to evaluate the performance of our models from both visualization and statistic aspects, which demonstrates the excellence of our models compared with the start-of-art baselines.

Acknowledgements

This work is supported by NSF grants No. 2011845, No. 1912753, and No. 2118083.

References

Augenstein, S.; McMahan, H. B.; Ramage, D.; Ramaswamy, S.; Kairouz, P.; Chen, M.; Mathews, R.; et al. 2019. Generative models for effective ML on private, decentralized datasets. *arXiv preprint arXiv:1911.06679*.

Brendan McMahan, H.; Moore, E.; Ramage, D.; Hampson, S.; and Agüera y Arcas, B. 2016. Communication-efficient learning of deep networks from decentralized data. *ArXiv e-prints*, arXiv–1602.

Chang, Q.; Qu, H.; Zhang, Y.; Sabuncu, M.; Chen, C.; Zhang, T.; and Metaxas, D. N. 2020. Synthetic learning: Learn from distributed asynchronized discriminator gan without sharing medical image data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 13856–13866.

Chen, Q.; Wang, W.; Huang, K.; De, S.; and Coenen, F. 2021. Multi-modal generative adversarial networks for traffic event detection in smart cities. *Expert Systems with Applications*, 177: 114939.

Durugkar, I. P.; Gemp, I.; and Mahadevan, S. 2017. Generative Multi-Adversarial Networks. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net.

Fabbri, M.; Calderara, S.; and Cucchiara, R. 2017. Generative adversarial models for people attribute recognition in surveillance. In *2017 14th IEEE international conference on advanced video and signal based surveillance (AVSS)*, 1–6. IEEE.

Fan, C.; and Liu, P. 2020. Federated generative adversarial learning. In *Chinese Conference on Pattern Recognition and Computer Vision (PRCV)*, 3–15. Springer.

Ganokratanaa, T.; Aramvith, S.; and Sebe, N. 2019. Anomaly event detection using generative adversarial network for surveillance videos. In *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 1395–1399. IEEE.

Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.

Hardy, C.; Le Merrer, E.; and Sericola, B. 2018. Gossiping GANs: Position paper. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, 25–28.

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

Im, D. J.; Ma, H.; Kim, C. D.; and Taylor, G. 2016. Generative adversarial parallelization. *arXiv preprint arXiv:1612.04021*.

Isola, P.; Zhu, J.-Y.; Zhou, T.; and Efros, A. A. 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1125–1134.

Lee, Y.; Yun, J.; Hong, Y.; Lee, J.; and Jeon, M. 2018. Accurate license plate recognition and super-resolution using a generative adversarial networks on traffic surveillance video. In *2018 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, 1–4. IEEE.

Li, D.; Chen, D.; Jin, B.; Shi, L.; Goh, J.; and Ng, S.-K. 2019. MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks. In *International Conference on Artificial Neural Networks*, 703–716. Springer.

Long, M.; Cao, Y.; Wang, J.; and Jordan, M. 2015. Learning transferable features with deep adaptation networks. In *International conference on machine learning*, 97–105. PMLR.

Lv, Y.; Chen, Y.; Li, L.; and Wang, F.-Y. 2018. Generative adversarial networks for parallel transportation systems. *IEEE Intelligent Transportation Systems Magazine*, 10(3): 4–10.

Mao, X.; and Li, Q. 2018. Unpaired multi-domain image generation via regularized conditional GANs. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2553–2559.

Mirza, M.; and Osindero, S. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Mohammadi, M.; Al-Fuqaha, A.; and Oh, J.-S. 2018. Path planning in support of smart mobility applications using generative adversarial networks. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 878–885. IEEE.

Pu, Y.; Dai, S.; Gan, Z.; Wang, W.; Wang, G.; Zhang, Y.; Henao, R.; and Duke, L. C. 2018. Jointgan: Multi-domain joint distribution learning with generative adversarial nets. In *International Conference on Machine Learning*, 4151–4160. PMLR.

Qu, H.; Zhang, Y.; Chang, Q.; Yan, Z.; Chen, C.; and Metaxas, D. 2020. Learn distributed GAN with Temporary Discriminators. In *European Conference on Computer Vision*, 175–192. Springer.

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

Salimans, T.; Goodfellow, I.; Zaremba, W.; Cheung, V.; Radford, A.; and Chen, X. 2016. Improved techniques for training gans. *Advances in neural information processing systems*, 29: 2234–2242.

Shi, Q.; Liu, X.; and Li, X. 2017. Road detection from remote sensing images by generative adversarial networks. *IEEE access*, 6: 25486–25494.

Triastcyn, A.; and Faltings, B. 2019. Federated generative privacy. *arXiv preprint arXiv:1910.08385*.

Tschuchnig, M. E.; Ferner, C.; and Wegenkittl, S. 2020. Sequential iot data augmentation using generative adversarial networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4212–4216. IEEE.

Vaccari, I.; Orani, V.; Paglialonga, A.; Cambiaso, E.; and Mongelli, M. 2021. A Generative Adversarial Network (GAN) Technique for Internet of Medical Things Data. *Sensors*, 21(11): 3726.

Wang, M.; and Deng, W. 2018. Deep visual domain adaptation: A survey. *Neurocomputing*, 312: 135–153.

Xin, B.; Yang, W.; Geng, Y.; Chen, S.; Wang, S.; and Huang, L. 2020. Private fl-gan: Differential privacy synthetic data generation based on federated learning. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2927–2931. IEEE.

Xiong, Z.; Xu, H.; Li, W.; and Cai, Z. 2021. Multi-Source Adversarial Sample Attack on Autonomous Vehicles. *IEEE Transactions on Vehicular Technology*, 70(3): 2822–2835.

Yonetani, R.; Takahashi, T.; Hashimoto, A.; and Ushiku, Y. 2019. Decentralized Learning of Generative Adversarial Networks from Non-iid Data. *arXiv preprint arXiv:1905.09684*.

Yoon, J.; Jordon, J.; and Schaar, M. 2018. RadialGAN: Leveraging multiple datasets to improve target-specific predictive models using Generative Adversarial Networks. In *International Conference on Machine Learning*, 5699–5707. PMLR.

Zhu, J.-Y.; Krähenbühl, P.; Shechtman, E.; and Efros, A. A. 2016. Generative visual manipulation on the natural image manifold. In *European conference on computer vision*, 597–613. Springer.

Zhu, J.-Y.; Park, T.; Isola, P.; and Efros, A. A. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, 2223–2232.