Distributed Generative Model: A Data Synthesizing Framework for Multi-Source Heterogeneous Data

Zuobin Xiong, *Member, IEEE,* Wei Li, *Member, IEEE,* Yingshu Li, *Member, IEEE,* and Zhipeng Cai, *Fellow, IEEE*

***Abstract*—Recent advancement in generative AI influenced a broad area with successful applications across multiple domains, including computer vision, natural language processing, and the Internet of Things (IoT). However, many existing implementations rely on centralized architectures, which introduce security and privacy concerns while also increasing communication overhead. Limited research has explored the development of distributed generative models, particularly in scenarios where training data originates from various heterogeneous sources. To fill the gap, this paper introduces a distributed generative model framework aimed at enhancing data generation in hierarchical IoT systems. The proposed framework supports distributed data generation across three distinct scenarios: feature-related data, label-related data, and feature-label non-related data. Furthermore, both synchronous and asynchronous update mechanisms are incorporated to accommodate diverse application requirements within IoT environments. Comprehensive experiments using simulated, image, and tabular datasets are conducted to assess the performance of the proposed framework in comparison with state-of-the-art methods. The results indicate that the framework effectively produces high-quality synthetic data while preserving the integrity of downstream tasks. Beyond large language models (LLMs), these findings suggest that generative AI have the potential to transform data generation in distributed IoT systems and be extended to a broader range of applications.**

***Impact Statement*—Generative AI is the most eye-catching star in the AI field now, which relieves humans from the tedious process of repeating tasks. However, the recent focus on large general generative models with billions of parameters is not prepared for some specific scenarios, such as the distributed IoT settings with heterogeneous data. The distributed generative model framework we introduce in this paper tries to mitigate this limitation by using small models in distributed devices. The proposed distributed generative model can produce realistic generated data in simulated data, image data, and even tabular data, which are the most common data formats in IoT, with competitive visual and statistical quality. This proposed framework can be promising when facing a variety of distributed applications, including healthcare and smart manufacturing, which will attract more interest in specialized generative AI in industrial IoT apart from the general generative AI models.**

***Index Terms*—Distributed Learning, Generative Adversarial Networks, Multi-source Heterogeneous Data, Federated Learning**

This work was supported by the National Science Foundation under grants No. 2429960, 2434899, 2416872, 2315596, 2244219, and 2146497.

Zuobin Xiong is with the Department of Computer Science, University of Nevada Las Vegas, Las Vegas, NV, 89154 USA (email: zuobin.xiong@unlv.edu)

Wei Li, Yingshu Li, and Zhipeng Cai are with the Department of Computer Science, Georgia State University, Atlanta, GA, 30303 USA (e-mail: wli28@gsu.edu; yili@gsu.edu; zcai@gsu.edu).

I. Introduction

The impact of generative models has been revolutionary, with many research areas and applications benefiting from the recent products such as GPT and Midjourney. However, in IoT scenarios, where various interconnected devices produce heterogeneous data for data-driven applications, generative models do not flourish impressively as in general AI applications. Many research and applications in IoT areas face special data formats and thus need customized generative models. In this vein, many customized generative models have been applied in various domains, including traffic data analysis [1], [2], video surveillance for anomaly detection [3], [4], and Intrusion Detection Systems (IDS) to enhance system reliability [5], [6]

In the above examples, many existing generative models are implemented in a centralized manner, leading to challenges such as single points of failure, privacy risks, and high communication costs, as highlighted in prior studies [7], [8], [9]. These limitations make it increasingly difficult to aggregate raw data from IoT users to a central server, restricting data availability and impeding application development. Additionally, transmitting large volumes of data to a central server results in substantial communication overhead for resource limited devices. Integrating generative models with distributed frameworks offers potential benefits to solve these problems for both individuals and society. For instance, distributed generative models can synthesize medical imaging patterns while preserving privacy, enabling data visualization and the creation of training datasets for various applications. Moreover, leveraging distributed generative models for feature extraction from IoT devices can assist developers in assessing data quality, identifying biases, and debugging misclassified instances [10]. In scenarios where IoT data is non-i.i.d. across multiple devices, learning a distributed generative model facilitates the capture of a mixed distribution, improving data diversity and representation.

Some existing works have developed distributed generative models but overlooked crucial issues in practical application scenarios: (i) limited network and computation resources when using the traditional federated learning framework. (ii) prior literature focuses on non-i.i.d. data in feature space and does not study more intricate data distribution. (iii) multi-source heterogeneity of data in distributed settings is less explored. Considering the above issues, designing a distributed generative model in non-i.i.d. and multi-source heterogeneous data scenarios remains an open challenge.

To tackle this challenge, we introduce a novel distributed

generative framework designed to accommodate the unique characteristics of IoT applications, including wide geographic distribution, limited computational resources, non-i.i.d. data, and diverse data domains. Our approach features a three-layer hierarchical architecture for deploying distributed generative models, marking one of the first efforts to address multi-source heterogeneous data in distributed data generation. While our framework shares similarities with the work by Xiong et al. [11], we place greater emphasis on handling non-i.i.d. data distributions and exploring asynchronous training strategies. Additionally, we identify a feature-label non-related scenario commonly found in real-world IoT applications and propose an innovative method within our hierarchical framework to generate data in this setting effectively. To summarize, the contribution of this work is as follows:

- We develop a three-layer hierarchical framework for deploying distributed generative models, specifically designed to accommodate the characteristics of IoT applications and support multi-source heterogeneous data generation.
- Three distinct distributed generative schemes are formulated within the proposed framework, aligning with the realistic data distribution patterns observed in IoT environments.
- We introduce both synchronous and asynchronous training strategies for edge-based data generators, allowing for flexible adaptation based on application-specific requirements.
- Comprehensive experiments are conducted on a variety of simulated and real-world datasets to evaluate the performance of our generative models, demonstrating their effectiveness in comparison with state-of-the-art approaches across different scenarios.

This paper is organized as follows. We review the related works on generative models in Section II. After presenting the model framework in Section III, we design three distributed generative models for three different scenarios in Section IV, Section V, and Section VI. We then evaluate our proposed methods through experiments in Section VII. Finally, this paper is concluded in Section VIII.

II. Related Works

A. Multi-Source Data Generative Models

Multiple data source generation problems can be categorized into conditional generation and joint generation. Conditional generation aims to model the conditional distribution of one data source given the presence of others as auxiliary information. This approach is widely used in various applications, including image-to-image translation [12], [13], [14], domain adaptation [15], [16], [17], and domain generalization [18], [19], [20], among others. Zhu *et al.*[12] proposed CycleGAN, a framework for unpaired image-to-image translation between multiple domains. Later, Isola *et al.*[13] developed pix2pix, a universal image-to-image translation framework that maps one image to its corresponding domain. Conditional generative adversarial networks (cGANs) have also been used for privacy-preserving image translation [14] and face attribute transfer [16]. Other works have focused on joint or multi-domain generation. CoGAN [21] was the first to learn a joint distribution without paired images. Mao and Li [22] used the domain label as a condition to train a single conditional generator for joint distribution without paired data. RadialGAN [15] trained multiple autoencoder-based generators that can capture the distribution of multiple joint data sources, improving data augmentation quality and addressing missing data issues. JointGAN [23] used multiple generators and a single discriminator to learn the joint distribution from multiple marginal samples. Besides, shared representation in the latent space has also been explored as a meaning of capturing joint information. In [24], multiple autoencoder-based GANs were used to extract the shared low-dimensional representation between sources, which was then used as a domain-invariant feature to train a classifier for domain adaptation. Gonzalez *et al.* [25] trained a cross-domain disentanglement network to decompose the latent space into domain-shared and domain-exclusive representations, and then the domain-shared representation was recovered to obtain the joint distribution through training decoders.

B. Distributed Generative Models

GAP [26] is the first approach to integrate GANs into a distributed framework, where each GAN model is trained using a standalone dataset. Gossip GAN [27] adopted a similar structure, where generators and discriminators exchange information within local neighborhoods to collectively approximate the global data distribution. Later, Durugkar *et al.*[28] and Hardy *et al.*[29] refined this structure by replacing multiple generators with a single one while retaining multiple discriminators, enabling the generator to leverage an aggregated discriminator loss for improved data generation. Expanding on this direction, Yonetani *et al.* [30] explored distributed data generation across multiple data sources, where the discriminators are averaged based on the weight of their loss values. Yet, these distributed generative methods are mostly evaluated on i.i.d. data within traditional distributed storage and are time-consuming during training. Another line of research combines federated learning [31] and various distributed GANs for the sake of the performance of FL in non-i.i.d data [32], [33], [34], [10]. The differences among these methods are whether to send generators [32], discriminators [10], or both [33], [34] for aggregation. However, a high communication cost caused by transmitting model parameters iteratively exists in these FL-based generative models. To address this limitation, the approaches in [35] and [36] introduce a central generator that is updated using the loss values from local discriminators. This method significantly reduces the amount of data exchanged between the server and local clients compared to transmitting entire models. Qu *et al.* [36] improves the above model by introducing temporary discriminators, endowing the proposed method with the capability to handle more flexible scenarios where local clients can join and quit at any time. In [37], [6], the authors utilize a similar framework to perform network intrusion detection on distributed devices in IoT by training a central generator/auto-encoder on a server. Although these

distributed GANs methods can solve the non-i.i.d. data of some trivial cases, none of them is specified for our realistic multi-source heterogeneous data generation, which lacks a solid ground for landing a practical application.

Moreover, as a more special yet commonly encountered data generation problem, distributed data with different feature and label spaces is more challenging due to its complex feature type and distributions (*e.g.,* continuous, categorical, ordinal, multi-mode). To solve this problem in a distributed setting, Zhao *et al.* [38] apply FL and TGAN [39] together to design the first Fed-TGAN framework. Later, an improved work GTV [40] is developed by the same authors to generate such mixed data features and labels in vertical federated learning scenarios. Their works are initial attempts at federated mixed-type data generation, and some important details are not considered, such as imbalanced category values. Duan *et al.* [41], [42] also devise GAN-based federated generation methods for this kind of data, but they use a strong assumption that all clients' feature distribution is the same, which violates the essence of non-i.i.d. of FL.

In summary, all the existing generative models are not prepared for distributed and multi-source heterogeneous data, which limits their applicability to distributed IoT scenario. In this work, we try to design a general framework for distributed data generation, in which three realistic data distributions are considered, including feature-related data, label-related data, and feature-label non-related data are realized.

III. System Framework

The proposed distributed generative model framework is described in Fig. 1, consisting of a set of IoT devices, a set of edge servers $\mathbb{K}$, and a cloud server G. In this setup, each edge server in the community k maintains a community generator G_k. Within the community k, a set of IoT devices $\mathbb{J}_k$ is covered and each device holds one discriminator D_{kj}, where D_{kj} is the discriminator on IoT device $j \in \mathbb{J}_k$ of community $k \in \mathbb{K}$.

Given the proximity of data sources within a local community, the data often exhibits correlations either in the feature space or the label space. When analyzing data distribution and relationships across different communities, this paper categorizes the data generation problem into three scenarios: (i) Feature-related scenario, where communities share the same feature space but have distinct labels; (ii) Label-related scenario, where communities have identical labels but differ in feature space; and (iii) Feature-label independent scenario, where neither features nor labels are necessarily the same across communities. The details of data generation under these three scenarios are illustrated in Section IV, Section V, and Section VI, after introducing the training process and updating strategy.

A. Training Process

In the distributed generative framework, the training process contains two phases: local community training, where G_k is updated at the edge servers for I_{loc} steps of local community training, and global training, where G is aggregated at the cloud server for I_{glo} iterations. In the local community training process, the community generator G_k is updated by the discriminator information D_{kj} ($j \in \mathbb{J}_k$) from distributed IoT devices. Different from most of the existing methods that use federated learning to obtain GANs, our local community training structure has a single generator and multiple distributed discriminators. Unlike the existing FL-based generative model that aggregates discriminators or generators to obtain the GANs, our local community training uses a single generator to collaborate with multiple discriminators for information exchange. In addition, different from the FL-GANs where the cloud server and local clients are transmitting model parameters, our local community training only sends two batches of generated data and the loss function value of community generators in each iteration, resulting in much better communication efficiency.

In the global training, the model parameters of the community generator G_k are aggregated on the global server side to calculate the global generator G, and one global iteration is done. Next, the aggregated global generator G is distributed back to the community generators for the next local community training iteration. It's worth noting that during the entire training process, the private data of each IoT device is always stored on IoT devices locally without leakage. Therefore, the proposed distributed generative framework can guarantee the data generation by collaboration among server, edge and IoT devices with privacy preservation.

B. Updating Strategy

Synchronous Updating. Synchronous updating means that G_k is updated after collecting the loss value $\mathcal{L}_{D_{kj}}(G_k)$ from covered IoT devices inside the community k during this training cycle, in which the updating time unit is pre-determined by the system. Compared with the traditional aggregation methods that adopt average aggregation for all loss functions, the proposed synchronous updating is able to capture the non-i.i.d data scenario. The complete loss value aggregation $\mathcal{L}_{D_{sync}}(G_k)$ is computed as follows:

$$\mathcal{L}_{D_{sync}}(G_k) = \sum_{j \in \mathbb{J}_k} \frac{e^{\mathcal{L}_{D_{kj}}(G_k)}}{\sum_{j' \in \mathbb{J}_k} e^{\mathcal{L}_{D_{kj}}(G_k)}} \mathcal{L}_{D_{kj}}(G_k). \tag{1}$$

The exponential weighted aggregation method has shown its performance on non-i.i.d. data in recent research [28], [30].

Asynchronous Updating. Asynchronous updating is proposed to handle the slow IoT devices in the system, due the limited computation and transmission power. In the asynchronous manner, the edge generator G_k is updated immediately upon the received loss function value from a local discriminator. In the general asynchronous scenario, the receiving time of a loss function value is vital to compute its weight during aggregation. For a loss function, the time gap between its receiving time and the last updating time is defined as its "staleness": $s_{kj} = T_{kj}^{rec} - T_k^{lat}$, where T_{kj}^{rec} is the timestamp when $\mathcal{L}_{D_{kj}}(G_k)$ arrives at the server G_k, and T_k^{lat} is the time of G_k's last updating. Accordingly, G_k can be updated asynchronously through Eq. (2).

$$\mathcal{L}_{D_{async}}(G_k) = \frac{1}{|\mathbb{M}_k|} \sum_{j \in \mathbb{M}_k} e^{-s_{kj}} \mathcal{L}_{D_{kj}}(G_k). \tag{2}$$

Here, $\mathbb{M}_k$ represents the set of local IoT devices whose loss function arrive at G_k simultaneously. As shown in Eq. (2), a

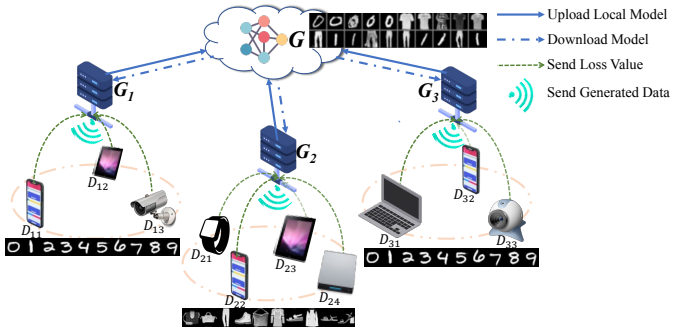


Fig. 1. The illustration of feature-related data generation.

smaller staleness is given a higher weight in the aggregated loss value when updating G_k. In addition, the staleness s_{kj} of a device is used to improve the learning rate of D_{kj} as a multiplier $e^{s_{kj}} \times \eta$, so that the straggler IoT device can have a larger step size to reconcile the local training time.

IV. Data Generation in Feature-Related Scenario

In this scenario, the goal is to learn a unified data distribution across all communities using the global generator G in the cloud server. In this case, datasets from different communities share the same feature space but have distinct labels. An illustration of this setup is provided in Fig. 1. Ultimately, the global generator G in the cloud server can synthesize a comprehensive data distribution encompassing all digits and clothing labels. The design of generators, discriminators, and their interactions in this proposed framework are similar in three scenarios. Therefore we only describe the basic components in this section and leave the detailed design in the supplementary materials Appendix-A.

1) Generator: In the generator setting, the community generator G_k and the global generator G share the same network structure. Community generators G_k generate two batches of data, which are used as fake data to update D_{kj} and to calculate the loss function value $\mathcal{L}_{D_{kj}}(G_k)$ of G_k on local devices D_{kj} for back propagation. The global generator G is aggregated from all cover community generators G_k via Eq. (3).

$$G = \sum_{k \in \mathbb{K}} \frac{\exp(\mathcal{L}_{D_*}(G_k))}{\sum_{k' \in \mathbb{K}} \exp(\mathcal{L}_{D_*}(G_k))} G_k, \tag{3}$$

where the loss function $\mathcal{L}_{D_*}(G_k)$ is calculated by Eq. (1) or Eq. (2) as detailed in supplementary materials.

2) Discriminator: On each IoT device, the discriminator D_{kj} operates in two steps: (i) updating its parameters and (ii) computing the loss function. (i) Specifically, to update D_{kj}, it is trained using both real local data x, which follows the data distribution $p_{kj}(x)$, and synthetic data $G_k(z_d)$ generated by G_k, as formulated in Eq. (4).

$$\begin{aligned} \max_{D_{kj}} \mathcal{L}(D_{kj}) =& \mathbb{E}_{x \sim p_{kj}(x)}[\log D_{kj}(x)] + \\ & \mathbb{E}_{z \sim p_z(z)}[\log(1 - D_{kj}(G_k(z_d)))]. \end{aligned} \tag{4}$$

(ii) Then, the fake data $G_k(z_g)$ is fed into the discriminator for calculating the loss function of G_k as follows,

$$\mathcal{L}_{D_{kj}}(G_k) = \mathbb{E}_{z \sim p_z(z)}[\log(1 - D_{kj}(G_k(z_g)))]. \tag{5}$$

Algorithm 1: Feature Related Data Generation.

Input: $\mathbb{K}$, $\mathbb{J}_k$, I_{glo} and I_{loc}
Output: G

System initialization G, G_k, and D_{kj} with $j \in \mathbb{J}_k$;
while $I_{glo} > 0$ **do** // global training
  for $k \in \mathbb{K}$ **do**
    while $I_{loc} > 0$ **do** // local training
      Select z_d and z_g from $\mathcal{N}(0, 1)$;
      Synthesize $G_k(z_d)$ and $G_k(z_g)$;
      Send $G_k(z_d)$ and $G_k(z_g)$ to D_{kj} for $j \in \mathbb{J}_k$;
      for $j \in \mathbb{J}_k$ **do**
        Train D_{kj} via Eq. (4);
        Calculate $\mathcal{L}_{D_{kj}}(G_k)$ based on Eq. (5);
      end
      if *Synchronous* **then**
        Train G_k via Eq. (1);
      end
      if *Asynchronous* **then**
        Train G_k via Eq. (2);
      end
      $I_{loc} \leftarrow I_{loc} - 1$;
    end
    Send G_k to server;
  end
  Update G based on Eq. (3) ; // global aggregation
  for $k \in \mathbb{K}$ **do**
    $G_k \leftarrow G$; // distribute G to communities
  end
  $I_{glo} \leftarrow I_{glo} - 1$;
end
Return G

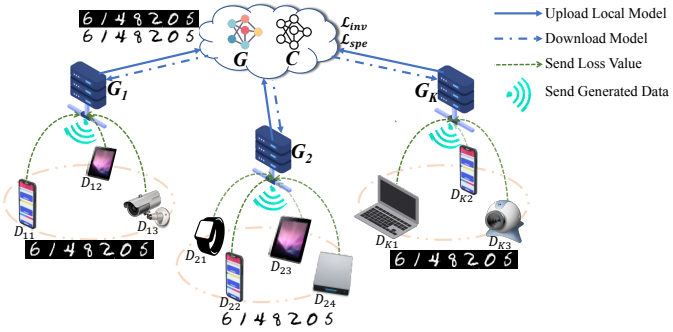


Fig. 2. The illustration of label-related data generation.

The value $\mathcal{L}_{D_{kj}}(G_k)$ is transmitted back to G_k to calculate gradients and update G_k.

The entire training process contains I_{glo} iterations of the global training, each of which has I_{loc} iterations of the local community training, which are described in Algorithm 1.

V. Data Generation in Label-Related Scenario

In this scenario, local datasets across different communities have distinct feature distributions but share the same labels, indicating that the data from each community belongs to a different domain. An example of label-related data generation is illustrated in Fig. 2, where three communities generate data with the same digit labels $0, 1, 2, \ldots, 9$ but different feature distributions. The global generator G in the cloud server aims to generate data from two separate feature spaces (i.e., two domains) simultaneously. A domain classifier C is introduced

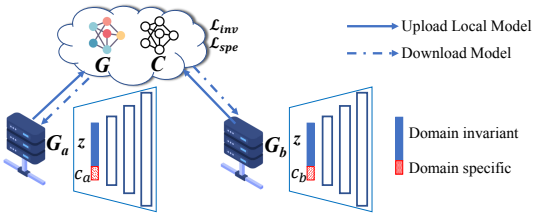


Fig. 3. Domain invariant vector and domain-specific vector.

in the cloud server to differentiate between the feature spaces of the generated data, as depicted in Fig. 3. Further details of the label-related data generation process, including conditional generators, discriminators, and the training procedure, can be found in the supplementary materials Appendix-B.

1) Domain Invariant: In the label-related scenario, the common class labels of all communities are defined as "domain invariant", which resides in high-level representation space [22], [43]. As a neural network generator, the high-level representation is the output of its first layer. The "domain invariant loss" is defined as the distance between the first-layer outputs of G under two different domain conditions, and it serves to guide the generator in producing the desired high-level representation. By optimizing this distance, the generator forces the output $G(z, c_k)$ to maintain the same class labels for the latent vector z, regardless of the domain condition.

To compute the loss, we first aggregate G in the cloud server by combining the G_k models sent by the communities, as described in Eq.(3). Then, using the domain condition c_k, G generates $G(z, c_k)$ for different c_k, each corresponding to a distinct data domain. Based on this generated data, we can extract the first-layer output, $G^{1st}(z, c_k)$, and calculate the domain invariant loss using Eq. (6), where the L_2 norm is employed to measure the difference in the high-level representation space.

$$\mathcal{L}_{inv}(G) = \sum_{k \neq k' \in \mathbb{K}} \|G^{1st}(z, c_k) - G^{1st}(z, c_{k'})\|_2. \tag{6}$$

Therefore, optimizing the loss function $\mathcal{L}_{inv}(G)$ can enforce the global generator G to generate similar high-level representations given distinct conditions, which can help strengthen the association between domain invariant and latent vector z.

2) Domain Specific: Domain specificity refers to the unique information associated with each domain (i.e., feature space). The domain condition c_k represents this domain-specific information, and it is used to self-label the generated data $G(z, c_k)$, which can then be used as a dataset to train a supervised domain classifier C. During the evaluation of C, we compute the domain-specific loss for G using the cross-entropy loss function $\mathcal{L}_{spe}(G) = \sum_{k \in \mathbb{K}} H(C(G(z, c_k)), c_k)$. Since the global generator G is expected to generate data with different features based on the domain condition c_k, minimizing $\mathcal{L}_{spe}(G)$ encourages G to encode domain-specific semantics within c_k. The training process for label-related data generation is outlined in Algorithm 2.

Algorithm 2: Label Related Data Generation.

Input: $\mathbb{K}$, $\mathbb{J}_k$, I_{glo} and I_{loc}
Output: G

System initialization G, C, G_k, D_{kj} with $j \in \mathbb{J}_k$, $first$=TRUE;
while $I_{glo} > 0$ **do** // global training
  for $k \in \mathbb{K}$ **do**
    while $I_{loc} > 0$ **do** // local training
      if $first$=*FALSE* **then**
        Train G_k via $\lambda_1 \mathcal{L}_{inv}(G) + \lambda_2 \mathcal{L}_{spe}(G)$;
      end
      Select z_d and z_g from $\mathcal{N}(0, 1)$;
      Synthesize $G_k(z_d, c_k)$ and $G_k(z_d, c_k)$;
      Send $G_k(z_d, c_k)$ and $G_k(z_d, c_k)$ to D_{ij} for $j \in \mathbb{J}_k$;
      for $j \in \mathbb{J}_k$ **do**
        Train D_{kj};
        Calculate $\mathcal{L}_{D_{kj}}(G_k)$;
      end
      if *Synchronous* **then**
        Train G_k via Eq. (1);
      end
      if *Asynchronous* **then**
        Train G_k via Eq. (2);
      end
      $I_{loc} \leftarrow I_{loc} - 1$, $first$=FALSE;
    end
    Send G_k to the server;
  end
  Update G based on Eq. (3) ; // global aggregation
  Update classifier C on $\{G(z, c_k), c_k | k \in \mathbb{K}\}$;
  Calculate loss $\mathcal{L}_{inv}(G)$ and $\mathcal{L}_{spe}(G)$ with C;
  for $k \in \mathbb{K}$ **do**
    $G_k \leftarrow G$; // distributed G to communities
  end
  $I_{glo} \leftarrow I_{glo} - 1$;
end
Return G

VI. Feature-Label Non-related Data Generation

In real applications, not all distributed communities ideally have the same feature or same label spaces. The data distribution often has different feature/label spaces across the IoT devices. For example, some devices in one community collect a set of features and labels, while other devices in another collect different features and labels under the same domain. Moreover, these features may have different data types, such as numerical, boolean, and categorical data. To provide a clear sense, tabular data is the most commonly encountered data type with these specific properties, where the column (attribute) types are numerical, categorical, boolean, etc.

In Fig. 4, some communities collect users' data with features (*i.e.,* columns) $[age, job, balance, loan]$ and the others collect data with columns $[age, education, pay/h, rent]$. The different columns will cause different distributions of tables in both feature and label spaces among different communities, which brings the problem of **feature-label non-related** scenario. To get a full understanding of the financial data, it is necessary to combine these tables with one generator to synthesize a complete tabular dataset with full columns $[age, education, job, pay/h, balance, rent, loan]$. Therefore, handling such different features, different labels, and mixed-

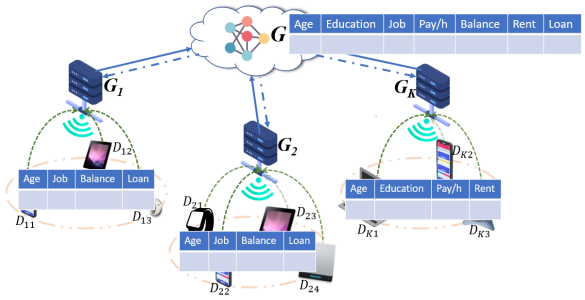


Fig. 4. The illustration of feature-label non-related data generation.

type data is our focus in this section.

To achieve this purpose, we need to solve three subproblems: (i) designing data representation for mixed-type data; (ii) formatting conditional generator and discriminator; (iii) using the proposed 3-layer framework to learn feature-label non-related data distribution.

A. Data Representation

It's non trivial to formulate feature-label non-related data because of the following challenges: (i) the existence of mixed type numerical data and categorical data requires different functions during generating data, *e.g.,* tanh and softmax function; (ii) the numerical values in each feature are not Gaussian distribution in most cases, which need more complex distribution to model these values; (iii) in categorical features, the categories are not balanced, where certain categories appear much more frequently than others.

Without loss of generality, we use the tabular data as an example of feature-label non-related data and denote its rows as follows. Suppose a table has numerical type columns as a set $\mathbb{N} = \{N_1, N_2, \ldots, N_{|\mathbb{N}|}\}$ and categorical type columns set $\mathbb{C} = \{C_1, C_2, \ldots, C_{|\mathbb{C}|}\}$, where $|\mathbb{N}|$ and $|\mathbb{C}|$ are the size of numerical column set and categorical column set, respectively. Each column is considered as a random variable, and the to-be-learned joint distribution of tabular data is represented by $P(N_1, \ldots, N_{|\mathbb{N}|}, C_1, \ldots, C_{|\mathbb{C}|})$. Then, the r-th row (record) of this table is denoted by $\{n_{r,1}, \ldots, n_{r,|\mathbb{N}|}, c_{r,1}, \ldots, c_{r,|\mathbb{C}|}\}$, where $n_{r,1}$ until $c_{r,|\mathbb{C}|}$ are the original data of each column in the table. We need to pre-process these numerical and categorical type data to scalar or vector so that they can be used as input for learning models.

1) Numerical Data Representation: Existing research has found that the numerical data in a column emerges as a multi-mode distribution instead of a standard gaussian distribution. Thus, we adopt a mode-specific normalization to represent the numerical data in each column. For each numerical column $N_i \in \{N_1, \ldots, N_{|\mathbb{N}|}\}$, we denote it independently. Each value in this column N_i is represented as a one-hot vector indicating the mode, and a scalar indicating the normalized value with the mode. This normalization method includes three steps as follows. (i) The variational Gaussian mixture model (VGM) [44] is used to fit the column values to a Gaussian mixture. The learned Gaussian mixture of column N_i is represented as $P_{N_i}(n_{r,i}) = \sum_{m=1}^{M} w_m \mathcal{N}(n_{r,i}; \mu_m, \sigma_m)$, where M is the number of modes, μ_m and σ_m are the mean and standard deviation of the m-th gaussian distribution. (ii) For each value $n_{r,i}$ in the numerical column N_i, the probability density that $n_{r,i}$ comes from the m-th gaussian distribution is calculated as $p_m = w_m \mathcal{N}(n_{r,i}; \mu_m, \sigma_m)$. (iii) Based on the probability density, we can select a gaussian distribution and use its mean value to normalize the value $n_{r,i}$. For example, if the m-th mode is selected, the numerical value $n_{r,i}$ is represented by an one-hot vector $\nu_{r,i} = [0, \ldots, 1, \ldots, 0]$ where the m-th element is 1, and a normalized value $\gamma_{r,i} = \frac{n_{r,i} - \mu_m}{\sigma_m}$. Following this idea, each numerical value $n_{r,i}$ can be represented as $\nu_{r,i}|\gamma_{r,i}$ by appending the vector and scalar. And then, all numerical columns of the r-th row $\{n_{r,1}, \ldots, n_{r,|\mathbb{N}|}\}$ can be modeled as $x_r^{num} = \nu_{r,1}|\gamma_{r,1} \oplus \nu_{r,2}|\gamma_{r,2} \oplus \cdots \oplus \nu_{r,|\mathbb{N}|}|\gamma_{r,|\mathbb{N}|}$, where $\oplus$ is concatenating operation.

2) Categorical Data Representation: Generally, each categorical data can be easily represented by a one-hot vector, where the length of the vector is the number of categories in this column. While, in the distributed scenario, a local client may not have all categories of a column, which will cause mismatch in representing the one-hot vector for global data generation. Thus, local clients need to upload some crucial but non-private information (*i.e.,* categorical column names and category names of each column) to the server for system initialization. Specifically, each client j in community k can construct a small table with format $T_{kj}[col_{name}][cat_{name}]$, where col_{name} is the name of the categorical column and cat_{name} is the category names appeared in this column without repeat. On the server side, we can combine all tables from clients and create a whole table $T[col_{name}][cat_{name}]$ to guide one-hot vector encoding. Therefore, each categorical column data $c_{r,i}$ in column $C_i \in \{C_1, C_2, \ldots, C_{|\mathbb{C}|}\}$ is represented as a one-hot vector $\beta_{r,i} = [0, \ldots, 1, \ldots, 0]$, where the length of $\beta_{r,i}$ is the number of categories in column C_i calculated from the full table $T[col_{name}][cat_{name}]$. Based on the one-hot vector, all categorical values of the r-th row $\{c_{r,1}, \ldots, c_{r,|\mathbb{N}|}\}$ is denoted as $x_r^{cat} = \beta_{r,1} \oplus \beta_{r,2} \oplus \cdots \oplus \beta_{r,|\mathbb{C}|}$.

Finally, the representation of a complex row data $x_r = \{n_{r,1}, \ldots, n_{r,|\mathbb{N}|}, c_{r,1}, \ldots, c_{r,|\mathbb{C}|}\}$ is modeled as Eq. (7)

$$x_r = x_r^{num} \oplus x_r^{cat}. \tag{7}$$

B. Condition of Mixed Data Generation

Our goal is to use one generator on the server to generate data with different features and label distributions. So, a naive generator can not handle this problem without more auxiliary information. There are two challenges we need to solve: (i) Constructing different columns for different clients; (ii) Generating imbalanced distribution as real categorical data.

1) Column Condition: The initialized generator on the server is a global generator that can generate a fixed size of $|\mathbb{C}|+|\mathbb{N}|$ columns. While different columns appear in different local clients, which requires a column condition to indicate what columns should be generated and evaluated in output. We design a condition vector con_k^{col} with the same length as x_r, which is filled with 0 and 1 to represent the existence of a specific column. For example, if the 1st numerical column appears in the real tabular data, the corresponding position of $\nu_{r,1}|\gamma_{r,1}$ in the vector con_k^{col} is filled with 1. If the $2nd$

categorical column does not exist, the corresponding position of $\beta_{r,2}$ is filled with 0. The column condition vector con_k^{col} will be used in the generators and discriminators to produce those appeared columns.

2) Imbalanced Data Condition: The elements in a categorical column are not evenly distributed. For example, in column "$education$", there will be much more "high school" values than "doctorate". Such imbalanced data distribution is hard to learn if we randomly sample data points to train the model [45]. Therefore, during training, an imbalanced data condition con_{kj}^{imb} is useful to capture the real distribution of columns in local clients. The imbalanced data condition con_{kj}^{imb} has the same length as categorical column representation x_r^{cat} and is filled with 0 and 1 as follows: (i) a categorical column is randomly selected with equal probability; (ii) one category of this column is selected based on the frequency distribution within this column; (iii) the selected category is represented as a one-hot vector of this column, and other columns in con_{kj}^{imb} are filled with 0. For instance, when the first category of the 3rd column is chosen, the value of con_{kj}^{imb} in the corresponding location of $\beta_{r,3}$ becomes $[1, 0, \ldots, 0]$ and all other columns $\beta_{r,i\neq 3}$ are filled with 0. When we select the condition con_{kj}^{imb} in this method, the sampled training data and generated data can have very similar class distribution as real tabular data for categorical columns.

The column condition con_k^{col} and imbalanced data condition con_{kj}^{imb} will be used together during data generation and discrimination as discussed in Section VI-C below.

C. Conditional Generator and Discriminator

1) Generator: Same as the previous scenario, we adopt conditional GANs to generate and discriminate tabular data. In each community, the edge generator G_k takes latent vectors z and con_k^{col} as input for data generation, where the column condition con_k^{col} indicates what columns are needed on local clients. During the local community training process, the generated data $G_k(z_d, con_k^{col})$ and $G_k(z_g, con_k^{col})$, are used for updating the discriminators and generator. While, due to different columns in each community, only the columns with value 1 in con_k^{col} are valid in local data, which requires extracting these columns from generated data $G_k(z_d, con_k^{col})$ and $G_k(z_g, con_k^{col})$ before sending to local clients. We design a column extracting matrix M_k^{col} based on the column condition con_k^{col} to reduce the column of raw generated data. M_k^{col} is a $|con_k^{col}| \times |con_k^{col}|_1$ matrix, where $|con_k^{col}|$ is the length of condition con_k^{col} and $|con_k^{col}|_1$ is the number of 1s in con_k^{col}. The matrix M_k^{col} is made up by following steps: (i) making a diagonal square matrix with con_k^{col} as the diagonal elements; (ii) removing all-zero columns and the remaining matrix is M_k^{col}. For example, let $con_k^{col} = [1, 0, 1, 0, 1]$, then M_k^{col} is constructed as

$$M_k^{col} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}^{\top} \tag{8}$$

Therefore, for generated raw data $G_k(z_d, con_k^{col})$ and $G_k(z_g, con_k^{col})$, we can multiply M_k^{col} on the right side to get the same tabular data size as local real data. And then, the processed data $G_k(z_d, con_k^{col}) \cdot M_k^{col}$ and $G_k(z_g, con_k^{col}) \cdot M_k^{col}$ are sent to local client for training.

2) Discriminator: Upon receiving two batches of processed generated data, the discriminator D_{kj} first calculates loss and updates parameters through maximizing Eq. (9)

$$\begin{aligned} &\max_{D_{kj}} \mathcal{L}(D_{kj}) = \mathbb{E}_{x\sim p_{kj}(x)}[\log D_{kj}(x|con_{kj}^{imb})] + \\ &\mathbb{E}_{z\sim p_z(z)}[\log(1 - D_{kj}(G_k(z_d, con_k^{col}) \cdot M_k^{col}|con_{kj}^{imb}))]. \end{aligned} \tag{9}$$

When D_{kj} is trained, the discriminator loss of G_k can be calculated based on the other batch of data $G_k(z_g, con_k^{col}) \cdot M_k^{col}$ as follows.

$$\mathcal{L}_{D_{kj}}^{dis}(G_k) = \mathbb{E}_{z\sim p_z(z)}[\log(1 - D_{kj}(G_k(z_g, con_k^{col}) \cdot M_k^{col}|con_{kj}^{imb}))]. \tag{10}$$

Besides, we need to ensure that the generated categorical data is as imbalanced as real data. So, the categorical part x_r^{cat} of data should be consistent between real and generated data with the given condition con_{kj}^{imb}. When $\beta_{r,i}$ of generated data is equal to the corresponding position of con_{kj}^{imb}, the generated categorical data is the same as the sampled training data. Thus, we can use cross-entropy to evaluate the loss of generator G_k on categorical data as follows.

$$\mathcal{L}_{D_{kj}}^{gen}(G_k) = \frac{1}{B} CE(\beta_{r,i}, \beta'_{r,i}), \tag{11}$$

where B is the batch size of generated data and $\beta'_{r,i}$ is the corresponding elements of $\beta_{r,i}$ in the generated data.

Finally, the total loss of generator G_k is calculated by adding Eq. (10) and Eq. (11) together as shown in Eq. (12), which was transmitted back to the local community to update G_k.

$$\mathcal{L}_{D_{kj}}(G_k) = \mathcal{L}_{D_{kj}}^{dis}(G_k) + \mathcal{L}_{D_{kj}}^{gen}(G_k). \tag{12}$$

The training process of feature-label non-related data generation is shown in Algorithm 3.

VII. Experiments

1) Experiment Settings: **Datasets.** In **feature-related** data generation, two datasets are used in the experiments, including (i) simulated Gaussian mixed data and (ii) image data, including MNIST, Fashion-MNIST, and EMNIST datasets.

In the **label-related** data generation, 5 dataset pairs are used, including (i) MNIST and inverse MNIST dataset; (ii) MNIST and edge MNIST dataset; (iii) MNIST and USPS dataset; (iv) sketch and photo handbag dataset; and (v) sketch and photo shoe dataset.

For the feature-label **non-related** scenario, 6 tabular datasets are adopted as benchmarks with various numbers of numerical and categorical columns, including Adult, Census, CovType, Intrusion, Credit, and Alarm datasets.

A. Evaluation on Feature Related Data Generation

Baselines. The centralized GAN model, federated GAN [33], and multi-discriminator GAN [35] are used as the baselines of feature related data generation.

The experiment results of feature-related scenario for 3 datasets combination [MNIST+F-MNIST+EMNIST] is presented in Fig. 5 respectively. The generated images on i.i.d data is marked in green square and non-i.i.d. data is marked in red square. As can be seen from the results of Fig. 5,

(a) Centralized GAN. (b) Federated GAN. (c) Multi-disc GAN. (d) Feature (sync). (e) Feature (async).

Fig. 5. Feature related data generation results on MNIST, Fashion-MNIST, and EMNIST (A-J) dataset in i.i.d./non-i.i.d. setting.

Algorithm 3: Feature-Label Non-related Generation.

Input: $\mathbb{K}$, $\mathbb{J}_k$, I_{glo} and I_{loc}
Output: G

Collect columns' names and categories of columns;
Initialize G, G_k, and D_{kj} with $j \in \mathbb{J}_k$;
while $I_{glo} > 0$ **do** // global training
  for $k \in \mathbb{K}$ **do**
    while $I_{loc} > 0$ **do** // local training
      Generate data $G_k(z_d, con_k^{col}) \cdot M_k^{col}$ and $G_k(z_g, con_k^{col}) \cdot M_k^{col}$;
      Send two batches of data to D_{kj} for $j \in \mathbb{J}_k$;
      for $j \in \mathbb{J}_k$ **do**
        Update D_{kj} via Eq. (9);
        Calculate $\mathcal{L}_{D_{kj}}(G_k)$ based on Eq. (12);
      end
      if *Synchronous* **then**
        Update G_k via Eq. (1);
      end
      if *Asynchronous* **then**
        Update G_k via Eq. (2);
      end
      $I_{loc} \leftarrow I_{loc} - 1$;
    end
    Send G_k to the server;
  end
  Update G based on Eq. (3) ; // global aggregation
  for $k \in \mathbb{K}$ **do**
    $G_k \leftarrow G$; // distribute G to communities
  end
  $I_{glo} \leftarrow I_{glo} - 1$;
end
Return G

the centralized GAN model can not generate complete data from all of the three data distribution. Specifically, it can only produce digits and some clothes as the centralized GAN is notorious for model collapse in complex dataset. Among the 4 distributed generators, federated GAN can generate most of the data but it can not perform well well in certain labels, such as row 4. In multi-discriminator GAN, the results shows a very similar performance to our feature-related generation, but it achieved a slightly worse statistic score as shown in the supplementary materials. Our feature-related data generation with an async updating strategy looks good in data generation since it can produce all classes but the image quality is a bit lower. In summary, our feature-related generation method in either sync or async scenario can earn satisfied performance in the i.i.d. situation.

Jumping to non-i.i.d. data, it is obvious that our feature-related data generation with sync updating strategy outperforms the other baselines in comparison. In particular, the performance degradation from i.i.d. to non-i.i.d. on our method is negligibly small, which justified the general application scenario of our method.

B. Evaluation on Label Related Data Generation

Baselines. In the label-related generation experiment, a self-built federated GANGAN [33] and the centralized GAN modelRegGAN [22] are used as the baselines.

Two new datasets are employed in this evaluation. The Edge MNIST dataset and the USPS dataset are presented in Fig. 6, and Fig. 7, where the result of i.i.d. data is in green square, and non-i.i.d.data is in red square. For the centralized model, RegGAN excels the best generation quality as all of the data sources are collected in a central server and trained. The conditional GAN framework behind it can process the different domain information. Thus, the RegGAN is used as the baseline upper bound for comparison. In the three distributed generator frameworks, the federated GAN and our label-related data generation (sync) showed the extremely similar data quality, and the label-related data generation (async) was a little bit lower. However, it is worth noting that in the federated GAN method, the generated data in each column is not in the same label sometimes because of the lack of our designated domain spe/inv loss. Therefore, the federated GAN can not guarantee such multi-domain generation in one shot. While in our label-related data generation, either the sync or async setting would produce the same semantic data (i.e., same label or object).

Similar observations can be obtained from the non-i.i.d. data in the green boxes. Though the data generation quality of federated GAN is comparable to our methods, the unique domain classifier showcase the effect in generating such multi-source data distribution. In summary, our proposed label-related data generation method, especially in the sync setting, can achieve not only better data quality but also maintain the semantic label information.

C. Evaluation on Feature-Label Non-related Data Generation

Baselines. In this scenario, federated GAN and a state-of-the-art method HTGAN [41] are selected as baselines. Since the baselines are not able to handle different feature and label spaces, we randomly split the whole dataset to each local client to construct a realistic scenario.

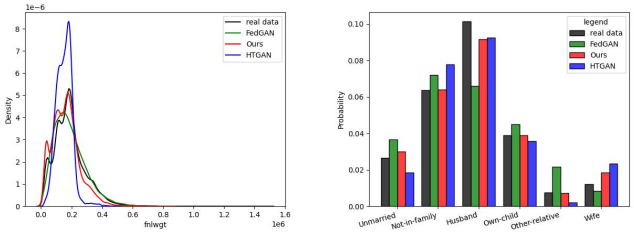
We evaluate the quality of synthetic tabular data from column distribution similarity and machine learning efficacy.

(a) RegGAN. (b) Federated GAN. (c) Label-related (sync). (d) Label-related (async).

Fig. 6. Label-related data generation results on MNIST and USPS dataset in i.i.d./non-i.i.d. setting.

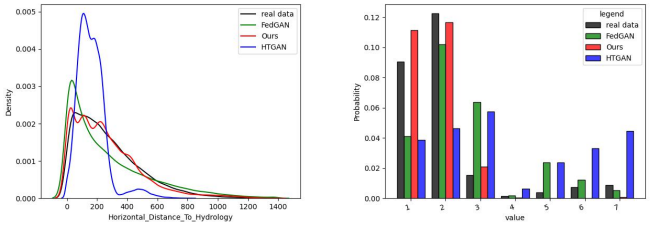
(a) RegGAN. (b) Federated GAN. (c) Label-related (sync). (d) Label-related (async).

Fig. 7. Label-related data generation results on MNIST and edge MNIST dataset in i.i.d./non-i.i.d. setting.



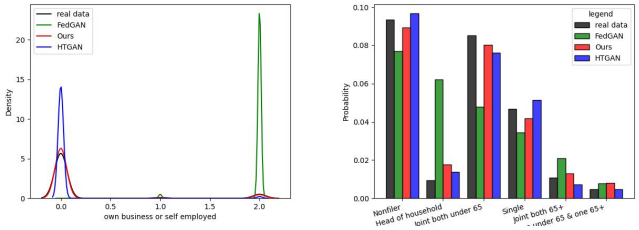
(a) 'fnlwgt' column distribution (b) 'relation' column distribution

Fig. 8. Selected column distribution comparison on Adult dataset.



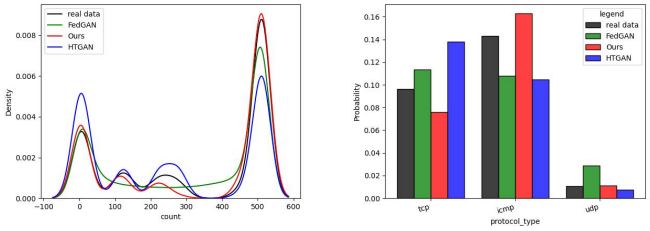
(a) 'hydrology' column distribution (b) 'value_label' column distribution

Fig. 10. Selected column distribution comparison on CovType dataset.



(a) 'employment' column distribution (b) 'tax_filer' column distribution

Fig. 9. Selected column distribution comparison on Census dataset.



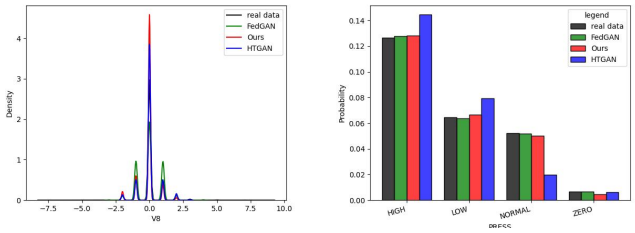
(a) 'count' column distribution (b) 'protocoltype' column distribution

Fig. 11. Selected column distribution comparison on Intrusion dataset.

1) Columns Distribution Similarity: We compared the probability distributions of the same column in the real tabular data and the synthetic data of different models. The comparison results are visualized from Fig. 8 to Fig. 12, which can clearly illustrate the statistical similarity between real data and our synthetic data against the baselines.

Due to the page limit, we can not show the distribution of all columns. Therefore, we display one numerical and one categorical column as representative. In Fig. 8(a), the black curve is the probability density distribution of real data for numerical column 'fnlwgt'. We can see that the density distribution of our model in the red curve is very similar to the real distribution, and can also recover different modes of real tabular data. While the generated result of FedGAN in the green curve, it only generates data with one mode surrounding the high-density area. This is because the original GAN is easy to fall into mode collapse when facing this multi-mode data without proper representation. The density curve of HTGAN can capture the multi-mode information of real data in density distribution, but the probability density does not fit very well. The reason is that HTGAN uses a federated Gaussian mixture model to estimate the local data, which causes more errors in local data representation. For the categorical column, the frequency histogram of 'relation' is shown in Fig. 8(b). Similarly, our generated data in red color has a closer frequency to the real data in the black bar for each category in the column, which is better than FedGAN and HTGAN. The results on both numerical and categorical columns indicate better synthetic data quality of our method. The visual performance of another 5 datasets is plotted in Fig. 9, Fig. 10, Fig. 11, and Fig. 12. And we can draw similar conclusions from these datasets by observing these numerical and categorical column distributions.

2) Machine Learning Efficacy: The generated synthetic data is supposed to be used for downstream tasks, which are mainly machine learning models for classification. Thus, we use the synthetic data to train machine learning models and then test these models on real data. We use the F1 score and macro-F1 score as the metrics for binary classification models and multi-classification models, respectively. For both metrics, the higher the score, the better the utility for the synthetic data. In Table I, we show the F1 scores and macro-F1 scores of the machine learning models and compare the quality of synthetic data. The



(a) 'V8' column distribution (b) 'Press' column distribution

Fig. 12. Selected column comparison on Credit and Alarm datasets.

TABLE I
QUANTITATIVE COMPARISON OF FEDGAN, OUR MODEL, AND HTGAN ON ADULT, CENSUS, COVTYPE, INTRUSION, CREDIT DATASETS. THE ALARM DATASET HAS NO LABEL AND THUS NOT COMPARED

	Binary Classification (F1)			Multi-Classification (macro-F1)	
Model	Adult	Census	Credit	CovType	Intrusion
Real Data	0.6851	0.5679	0.7515	0.6151	0.8128
FedGAN	0.5686	0.4490	0.6641	0.4226	0.4359
Ours	0.6770	0.5341	0.7319	0.5613	0.7345
HTGAN	0.6528	0.5113	0.7142	0.5090	0.6672

average F1 scores of different classifiers trained on real data and generated data are shown in the "Binary Classification" column of Table I. We can observe that for the Adult dataset, the F1 score of real data is the highest since the classifiers are trained and tested on the original real data. Among the three generative models, our method reaches 0.6770 F1 score, which is the best and very close to the real data performance. This reveals that our method can achieve quite similar performance on downstream classification applications as real data. Besides, in the two baselines, FedGAN fails with the lowest F1 score because of its bad generative capability, and HTGAN has a slightly lower F1 score than ours due to the same reason we've seen in the column distribution comparison. The result of the F1 score on other datasets can also be found in Table I, which expresses the same trend as the Adult dataset. The same testing procedure is adopted in multi-classification as binary classification, where we train different classifiers on the generated data and then evaluate the macro-F1 scores on real data. For the CovType dataset, the macro-F1 score of our method is 0.5613, which is the best among all generated data and much higher than FedGAN and HTGAN. The reason is that we use the column condition and imbalanced data condition to improve the data generation in the conditional generator and discriminator. The comparison of the macro F1-score on the Intrusion dataset also agrees that our generated data has better quality in multi-classification tasks. The results in Table I demonstrate that our method outperforms the two baselines in the feature-label not-related data generation.

D. Evaluation on Complexity vs. Performance

Considering that the generated model will be deployed on IoT devices, the computation cost and efficiency should be important given the limited resources. We explore the state-of-the-art Quantized FL methods [46] during the training process and then deploy the trained generative model on the physical development board through quantization to evaluate the actual performance for inference time efficiency and generated data quality. This physical board Maix-III is an IoT terminal device equipped with an AX620 chip, an AI SoC chip with an NPU that has a computing power of 3.6TOPs@INT8, a high energy efficiency ratio, and low power consumption, which can build an AI operating environment at a lower cost.

TABLE II
THE COMPARISON OF MODEL PERFORMANCE AND EFFICIENCY THROUGH QUANTIZATION.

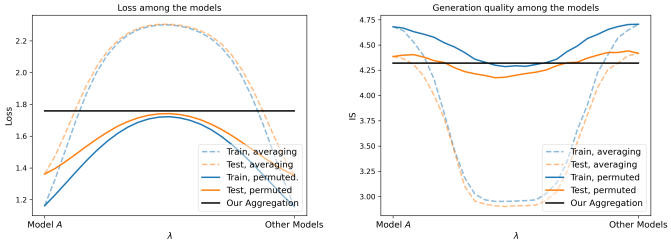
	Time/batch	IS ↑	FID ↓	Accuracy
Training D w/o quantization	498.25(ms)	-	-	-
Training D w/ quantization	542.99(ms)	-	-	-
Inference G w/o quantization	275.47(ms)	4.31	14.02	90.65%
Inference G w/ quantization	24.35(ms)	4.12	17.74	86.75%

Without loss of generality, we evaluate the proposed distributed generative model framework in the feature-related scenario, and all of the statistics presented in Table. II are calculated based on the MNIST/Fashion-MNIST/EMNIST datasets with fixed input dimension (28×28) and batch size 32.

It's worth noting that in our framework, each IoT device only needs to train a discriminator model D_{kj}. Therefore we only evaluate the training time for D_{kj}. For the trained generative model, we evaluate the inference time and generated data quality. Based on the results in Table. II, we find that the quantization operation during training IoT discriminator D will increase the time cost. The possible reason is that there are extra operations included in the quantization operation such as parameter mapping and calculation quantization. However, the overall training time for the discriminator w/ or w/o quantization is around 0.5s, which is very efficient and decent for resource-limited devices. The quantization operation stands out in the inference time of generator G, which is about 11.5× faster than w/o quantization. But in real evaluation, it also costs a one-time initialization time (1094.88ms) to enable the inference process with quantization. As for the generated data quality, since the quantitation operation tradeoff precision for efficiency, the IS, FID, and accuracy of generated data are slightly reduced but not much, which is still acceptable for general application. In summary, we argue that for small datasets on IoT devices, adopting quantization during the model training stage may not be always beneficial for all tasks due to the extra cost of quantization operation. A better option of quantization may be in the development stage after the model has been trained on more powerful devices, which will drastically increase inference speed with minor utility loss.

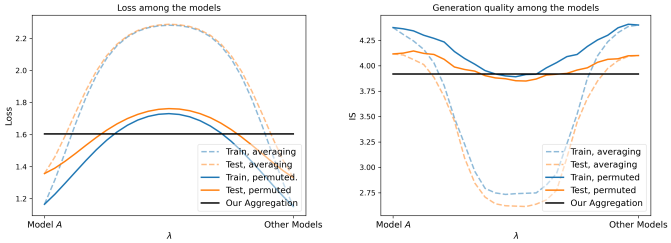
E. Evaluation on Permutation Invariance

The global training process of our proposed framework is like a model-merging approach. Based on the findings of paper [47] about permutation invariance in discriminative models, we conjecture that there might be a positive impact of permutation invariance on merging generation models. We adopt the "weight matching" [47] mechanism to evaluate the impact of permutation by merging our local community gen-



(a) impact on loss (b) impact on generation quality

Fig. 13. The impact of permutation invariance in feature-related generation



(a) impact on loss (b) impact on generation quality

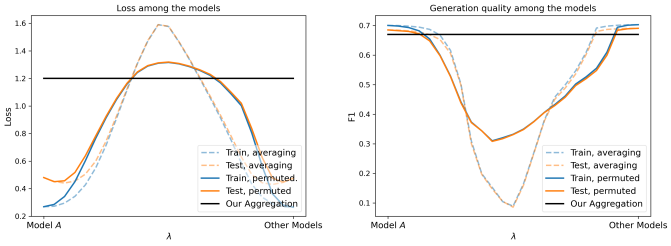
Fig. 14. The impact of permutation invariance in label-related generation

eration models for training and testing, where the comparison involves permuted aggregation, averaging aggregation, and our proposed loss-based aggregation. The experiment results are shown in Fig. 13, Fig. 14, and Fig. 15.

It can be seen from Fig. 13 that the permutation invariance does impact generative model merging. In Fig. 13(a), the permuted invariance aggregation method achieves smaller loss values in the peak compared with the naive averaging aggregation method, i.e., FedAvg. However, our loss-based aggregation (irrelevant to λ) is very close to the permuted invariance, though slightly worse. This phenomenon demonstrates that our aggregation method can also find a good aggregation approach in terms of loss value without permutation invariance. Besides, the generated data quality comparison is shown in Fig. 13(b). The data quality (IS) of permuted invariance aggregation is much better than that of the naive averaging aggregation, similar to the loss function. However, our loss-based aggregation can achieve comparative (sometimes better) data generation results. Fig. 14 and Fig. 15 depict the same pattern for label-related and feature-label non-related data generation. In summary, the performance in loss and generated data quality confirms that the permutation invariance somehow exists in generative models and it outperforms the averaging aggregation methods in both metrics. In addition, our loss-based aggregation has a slightly worse loss than permutation invariance but with a similar data generation quality. We assume this is caused by the following reasons: (i) In the original analysis of permutation invariance, different training subsets are from the same dataset and may lead to close local minima. In our problem, the datasets of different community generators are from different distributions (different features, labels, or both). Therefore, the local minima of the different communities might not be very close, causing worse performance for permutation invariance projection. (ii) Our loss-based aggregation method already considered a designated weighting strategy, assigning more weight to the generators with better generation quality. Based on this, our method can still generate good data compared with the optimization-based permutation invariance method.

F. Evaluation on Asynchronous Updating

We conduct the ablation study on the proposed asynchronous updating strategy and report the running time of different approaches to reach convergence in Table III, in which [Async w/o lr] is the proposed Async without adjusting



(a) impact on loss (b) impact on generation quality

Fig. 15. The impact of permutation invariance in feature-label non-related

learning rate based on staleness, and [Async w/o dw] is the proposed Async without dynamic weight adjustment by staleness. As can be seen from this table, the Sync method has the highest computation cost across all three scenarios. This is expected given that in the synchronous protocol, the community generator has to wait for the slow IoT device to finish and send their updates. Among the comparison of asynchronous strategies, Async is the most time-efficient compared with the other two, while [Async w/o dw] is closer to Async in running time. From the empirical results, we conclude that the dynamic learning rate (lr) is a promising strategy and effective when there are delays in IoT devices. [Async w/o lr] is slightly better than [Async w/o dw] in the feature-label non-related scenario, for which we suspect that in this scenario, a smaller model dimension converges faster and overtakes the learning rate effect. For larger datasets and complex models, we observe that the learning rate is more important than dynamic weighing in terms of time efficiency.

VIII. Conclusion & Future Work

In this work, the generative adversarial network, a classic generative AI model, is studied from the view of distributed computation. Considering the realistic case where distributed data possesses feature and label correlations, authors model a distributed generative framework to handle explored scenarios. Three detailed methods for solving the feature-related, label-related, and feature-label non-related scenarios are designed and implemented to compare with existing state-of-the-art models, illustrating the superior performance in dedicated applications. To comprehensively evaluate the proposed framework, intensive experiments are conducted on aspects such as data quality, time efficiency, computation complexity, and so on. In future work, we are working towards extending the proposed framework to more general data such as text and

TABLE III
COMPUTATION TIME (IN MINUTES) TO REACH CONVERGENCE. THE IOT DEVICE DELAY OF EACH CLIENT WAS SET TO BE A RANDOM VALUE BETWEEN 10 TO 60 SECONDS. **BEST** AND SECOND

	Sync	Async	Async w/o lr	Asyng w/o dw
Feat-R	260.86	**241.72**	251.36	244.97
Label-R	338.48	**320.24**	330.45	326.54
Feat-Label Non-R	181.36	**155.81**	158.46	160.92

video, to broaden the generative AI field.

REFERENCES

[1] Y. Lv, Y. Chen, L. Li, and F.-Y. Wang, "Generative adversarial networks for parallel transportation systems," *IEEE Intelligent Transportation Systems Magazine*, vol. 10, no. 3, pp. 4–10, 2018.

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

[3] T. Ganokratanaa, S. Aramvith, and N. Sebe, "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)*. IEEE, 2019, pp. 1395–1399.

[4] Y. Lee, J. Yun, Y. Hong, J. Lee, and M. Jeon, "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)*. IEEE, 2018, pp. 1–4.

[5] J. Lee and K. Park, "Gan-based imbalanced data intrusion detection system," *Personal and Ubiquitous Computing*, vol. 25, no. 1, pp. 121–128, 2021.

[6] T. Zixu, K. S. K. Liyanage, and M. Gurusamy, "Generative adversarial network and auto encoder based anomaly detection in distributed iot networks," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–7.

[7] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[8] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[9] M. Al-Rubaie and J. M. Chang, "Privacy-preserving machine learning: Threats and solutions," *IEEE Security & Privacy*, vol. 17, no. 2, pp. 49–58, 2019.

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

[11] Z. Xiong, W. Li, and Z. Cai, "Federated generative model on multi-source heterogeneous data in iot," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 9, 2023, pp. 10 537–10 545.

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

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

[14] Z. Xiong, W. Li, Q. Han, and Z. Cai, "Privacy-preserving auto-driving: a gan-based approach to protect vehicular camera data," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 668–677.

[15] J. Yoon, J. Jordon, and M. Schaar, "Radialgan: Leveraging multiple datasets to improve target-specific predictive models using generative adversarial networks," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5699–5707.

[16] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8789–8797.

[17] H. Li, J. C. Paetzold, A. Sekuboyina, F. Kofler, J. Zhang, J. S. Kirschke, B. Wiestler, and B. Menze, "Diamondgan: unified multi-modal generative adversarial networks for mri sequences synthesis," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2019, pp. 795–803.

[18] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[19] Y. Li, X. Tian, M. Gong, Y. Liu, T. Liu, K. Zhang, and D. Tao, "Deep domain generalization via conditional invariant adversarial networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 624–639.

[20] S. Zhao, M. Gong, T. Liu, H. Fu, and D. Tao, "Domain generalization via entropy regularization," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[21] M.-Y. Liu and O. Tuzel, "Coupled generative adversarial networks," *Advances in neural information processing systems*, vol. 29, pp. 469–477, 2016.

[22] X. Mao and Q. Li, "Unpaired multi-domain image generation via regularized conditional gans," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 2553–2559.

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

[24] C. Chen, W. Xie, Y. Wen, Y. Huang, and X. Ding, "Multiple-source domain adaptation with generative adversarial nets," *Knowledge-Based Systems*, vol. 199, p. 105962, 2020.

[25] A. Gonzalez-Garcia, J. Van De Weijer, and Y. Bengio, "Image-to-image translation for cross-domain disentanglement," *arXiv preprint arXiv:1805.09730*, 2018.

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

[27] C. Hardy, E. Le Merrer, and B. Sericola, "Gossiping gans: Position paper," in *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, 2018, pp. 25–28.

[28] I. P. Durugkar, I. Gemp, and S. Mahadevan, "Generative multi-adversarial networks," in *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=Byk-VI9eg

[29] 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.

[30] R. Yonetani, T. Takahashi, A. Hashimoto, and Y. Ushiku, "Decentralized learning of generative adversarial networks from non-iid data," *arXiv preprint arXiv:1905.09684*, 2019.

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

[32] A. Triastcyn and B. Faltings, "Federated generative privacy," *arXiv preprint arXiv:1910.08385*, 2019.

[33] C. Fan and P. Liu, "Federated generative adversarial learning," in *Chinese Conference on Pattern Recognition and Computer Vision (PRCV)*. Springer, 2020, pp. 3–15.

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

[35] Q. Chang, H. Qu, Y. Zhang, M. Sabuncu, C. Chen, T. Zhang, and D. N. Metaxas, "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*, 2020, pp. 13 856–13 866.

[36] H. Qu, Y. Zhang, Q. Chang, Z. Yan, C. Chen, and D. Metaxas, "Learn distributed gan with temporary discriminators," in *European Conference on Computer Vision*. Springer, 2020, pp. 175–192.

[37] A. Ferdowsi and W. Saad, "Generative adversarial networks for distributed intrusion detection in the internet of things," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[38] Z. Zhao, R. Birke, A. Kunar, and L. Y. Chen, "Fed-tgan: Federated learning framework for synthesizing tabular data," *arXiv preprint arXiv:2108.07927*, 2021.

[39] L. Xu and K. Veeramachaneni, "Synthesizing tabular data using generative adversarial networks," *arXiv preprint arXiv:1811.11264*, 2018.

[40] Z. Zhao, H. Wu, A. Van Moorsel, and L. Y. Chen, "Gtv: Generating tabular data via vertical federated learning," *arXiv preprint arXiv:2302.01706*, 2023.
[41] S. Duan, C. Liu, P. Han, X. Jin, X. Zhang, T. He, H. Pan, and X. Xiang, "Ht-fed-gan: Federated generative model for decentralized tabular data synthesis," *Entropy*, vol. 25, no. 1, p. 88, 2022.
[42] S. Duan, C. Liu, P. Han, T. He, Y. Xu, and Q. Deng, "Fed-tda: Federated tabular data augmentation on non-iid data," *arXiv preprint arXiv:2211.13116*, 2022.
[43] Z. Xiong, H. Xu, W. Li, and Z. Cai, "Multi-source adversarial sample attack on autonomous vehicles," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 3, pp. 2822–2835, 2021.
[44] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.
[45] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[46] Z. Xiong, W. Li, Y. Li, and Z. Cai, "Exact-fun: An exact and efficient federated unlearning approach," in *2023 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2023, pp. 1439–1444.
[47] S. K. Ainsworth, J. Hayase, and S. Srinivasa, "Git re-basin: Merging models modulo permutation symmetries," *arXiv preprint arXiv:2209.04836*, 2022.