GDTS: GAN-based Distributed Tabular Synthesizer

GDTS: GAN-based Distributed Tabular Synthesizer

Zilong Zhao
TU Delft
Delft, Netherlands
z.zhao-8@tudelft.nl

Robert Birke
University of Turin
Turin, Italy
birke@ieee.org

Lydia Y. Chen
TU Delft
Delft, Netherlands
lydiaychen@ieee.org

***Abstract*—Generative Adversarial Networks (GANs) are typically trained to synthesize data, from images and more recently tabular data, under the assumption of directly accessible training data. While learning image GANs on Federated Learning (FL) and Multi-Discriminator (MD) systems has just been demonstrated, it is unknown if tabular GANs can be learned from decentralized data sources. Different from image GANs, state-of-the-art tabular GANs require prior knowledge on the data distribution of each (discrete and continuous) column to agree on a common encoding – risking privacy guarantees. In this paper, we propose GDTS, a distributed framework for GAN-based tabular synthesizer. GDTS provides different system architectures to match the two training paradigms termed GDTS_FL and GDTS_MD. Key to enable learning on distributed data is the proposed novel privacy-preserving multi-source feature encoding to capture the global data properties. In addition GDTS encompasses a weighting strategy based on table similarity to counter the detrimental effects of non-IID data and a validation pipeline to easily assess and compare the performance of different paradigms and hyper parameters. We evaluate the effectiveness of GDTS in terms of synthetic data quality, and overall training scalability. Experiments show that GDTS_FL achieves better statistical similarity and machine learning utility between generated and original data compared to GDTS_MD.**

***Index Terms*—Tabular GAN, federated learning, tabular data, Non-IID**

I. Introduction

Generative Adversarial Networks (GANs) [5] are an emerging methodology to synthesize data, ranging from images [11], [12], to text [19], to tables [16], [22]. Key to training GANs are two competing neural networks, i.e., generator and discriminator, where the former iteratively generates synthetic data and the latter judges its quality. During the training process, the discriminator needs to access the original data and provide feedback to the generator by comparing it with the generated data. However, such a privilege of direct data access may no longer be taken for granted due to the ever increasing concerns for data privacy. In response to such a demand, the federated learning (FL) and multi-discriminator (MD) GAN paradigms emerge. FL features decentralized local processing, under which machine learning (ML) models can first be trained in parallel on clients' local data and subsequently be securely aggregated by the federator. As such, the local data is not directly accessed, except by the owner, and only the intermediate local model is shared. MD leverages the nature of GAN training where only the discriminator needs to see the real data. Therefore, it locates one discriminator at each clients' with access to the local (real) data and centrally hosts the generator. Prior art has explored these two paradigms for image data [6], [7], but none tackles tabular data even if it is the most dominant data type in industries [1].

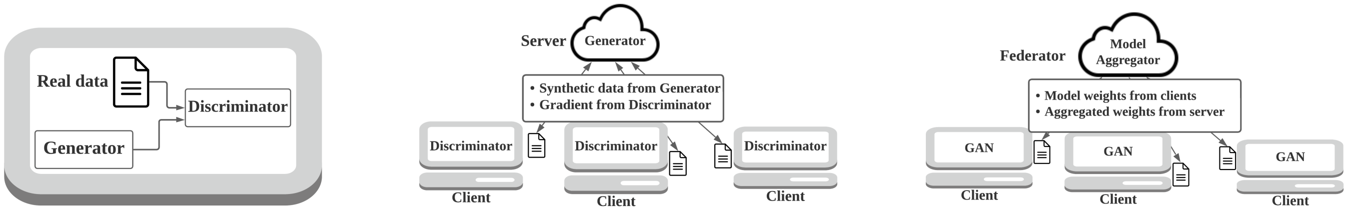
Compared to image GANs, training state-of-the-art tabular GANs (e.g., TVAE [22], CTGAN [22], CTAB-GAN [23], CTAB-GAN [24], and IT-GAN [13]) from decentralized data sources in a privacy-preserving manner presents toe crucial additional challenges rooted in the global data properties. Images have a standard encoding, i.e. pixels of one/three color channels for black-and-white/color images. Tables lack such a priori knowledge and tabular GANs must explicitly model each column, be it continuous or categorical, via data-dependent encodings which leverage global data properties.Hence, the first challenge is determine global column encoders from non-identically independently distributed (non-IID) local data in a privacy-preserving manner[1]. The second challenge is that the convergence speed of GANs critically depends on how local models are merged [9]. For image GANs [18], the merging weights are determined jointly by the data quantity and (dis)similarity of a single class distribution across clients. Tabular GANs instead must account for all columns i.e., differences in every column across clients.

In this paper, we aim to design a first of its kind framework, GDTS, that provides an infrastructure for privacy-preserving training of tabular GANs on distributed data. GDTS uses the PyTorch RPC framework to implement both FL and MD training paradigms and adds algorithmic features that address fine-grained privacy-preserving column modeling. First, the novel feature encoding scheme of GDTS can reconstruct the entire continuous column distribution via bootstrapping each client's partial information without privacy leakage. Secondly, GDTS injects noise, formed by the definition of differential privacy [4], to categorical column statistics to avoid individual privacy leakage while building global feature encoder for categorical column. Additionally, for the FL case, a more precise weighting scheme can effectively merge local models by considering the quantity and distribution dissimilarity for every column across all clients.

After implementing FL and MD paradigm through GDTS, we extensively evaluate GDTS_FL and GDTS_MD on a vast number of client scenarios with disparate data distributions. Specifically, GDTS_FL and GDTS_MD are compared with

[1] Privacy-preserving solutions refer to ones that do not require full knowledge of the local data.

Author's preprint of Z. Zhao, R. Birke and L. Y. Chen, "GDTS: GAN-Based Distributed Tabular Synthesizer,"
2023 IEEE 16th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, 2023, pp. 570-576, doi: 10.1109/CLOUD60044.2023.00078.



(a) Centralized GAN (b) Multi-Discriminator structure (c) Federated Learning structure

Fig. 1: The architecture choices for training GANs.

three baselines including the centralized approach and three ablated variants of GDTS_FL. Each of the ablations shows the efficacy of the corresponding feature component. The evaluation is performed on four commonly used machine learning datasets where the statistical similarity between generated and real data are reported as evaluation metrics. Under an unbalanced amount of local data among the clients, GDTS_FL converges significantly faster than its variants. And for scenarios where data across clients is non-IID, the convergence of GDTS_FL is not only stable but also produces synthetic data with 44.74% higher ML utility compared to GDTS_MD.

II. Decentralized Table Synthesizer

We propose a framework for decentralized tabular data synthesizing, GDTS, which can train a GAN-based tabular data generator on decentralized data sources. There are two paradigms of decentralized learning, GDTS_MD and GDTS_FL, that show different advantages in model and system performance of training of tabular GANs.

Overview Fig. 2 shows the high-level workings of GDTS. At the start all data column statistics are collected and processed by the global feature encoding component (details in Sec.II-B). After unifying the encoding, GDTS_MD or GDTS_FL can be chosen to train a distributed tabular GAN synthesizer. Each paradigm has its unique functionality that is specifically designed for tabular GAN training. G, D, S and C denotes generator, discriminator, server and client. GDTS contains a validation pipeline to assess the quality of synthetic data, i.e., machine learning utility and statistical similarity.

Threat Model To design privacy-preserving algorithm, we first define the threat model considered in this paper. Even though tabular datasets are retained locally on each client, the values used for building global feature encoders in GDTS are still subject to revealing sensitive information if not designed carefully, leading to risk of data leakage. Based on several previous risk analysis studies [8], [15], [21] in federated learning, our proposed GDTS architecture focuses on the semi-honest model. We assume that the clients and server are honest but curious, adhering to the GDTS protocol but potentially seeking additional information during computation. Collusion between clients and server is not considered.

A. Training Architecture Designs

1) **GDTS_MD:** uses one single central generator and multiple discriminators distributed across the clients as illustrated in Fig. 1b. Such a structure ensures that the client's data does not need to leave the clients' machines. The downside of the MD structure is that it induces significant communication overhead between the generator and discriminator, i.e., sending synthesized data to all discriminators, and returning the discriminators' gradients to the generator per each training epoch. Communication burden can be lowered by allowing multiple local training epochs for each global training round. However, client discriminators tend to overfit to their local data with increasing training epochs. To counter that, GDTS_MD allows clients to randomly swap their models in a peer-to-peer way every several epochs.

2) **GDTS_FL:** is composed of multiple GANs (composed of both a discriminator and a generator), one at each client (see Fig. 1c). Each GAN has access only to the data of the corresponding client. Each client first trains his GAN using the local data for a number of epochs, then sends the GAN model to the server hosting the federator. The key roles of the federator are: (i) during initialization to determine the GANs architecture to use; and (ii) during training to collect and aggregate the locally trained GANs models into a global GAN model; and (iii) redistribute the global GAN model to all the clients. Communication occurs when clients upload their model weights to the federator and when the federator redistributes the updated weights. Such a communication cycle is commonly referred to as global *training round* in FL studies. The resulting overhead depends on the local epochs and global rounds needed to achieve the desired model performance. Compared to the MD paradigm, this overhead tends to be lower as transferring model weights is typically more efficient than transferring synthesized data. Moreover, the FL paradigm has a stronger scalability relative to the number of clients, as the computation complexity of model aggregation is lower than training the generator network. Finally, the FL paradigm allows to easily weight local models against each other since local models are all available at the same time for aggregation. Weighting helps to accelerate the training convergence under skewed data distributions among clients.

B. Privacy-Preserving Feature Encoding

Our privacy-preserving model initialization comprise three steps (see Fig. 3a): 1) collecting statistics from clients to the server, 2) creating the column encoders, and 3) encoding local data at the clients. Fig. 3b illustrates the process detail of creating global encoders for categorical and continuous

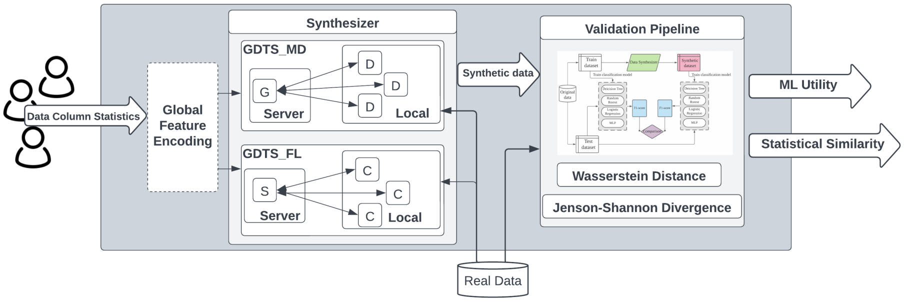


Fig. 2: System overview of GDTS: decentralizing GANs training and validation pipeline.

columns. To protect the column statistics in step 1, The Laplace Mechanism of differential privacy [4] is used. In this paper, when we use the Laplace mechanism, we always choose $\epsilon = 1$ as the privacy budget.

Step 1. Each of the P clients extracts the statistical properties of the local data and sends them to the server. The information sent is different based on the column type. For any categorical column j, each client i first computes the category frequency distribution X_{ij}, then samples a Laplacian noise vector ξ_{ij} according to the Laplace mechanism, and finally sends $\hat{X_{ij}} = X_{ij} + \xi_{ij}^X$ as the local category frequency distribution to server. For any continuous column j, each client i fits and sends in the parameters of the VGM model VGM_{ij}. In addition, Each client i also sends in $\hat{N_i} = N_i + \xi_i^N$ where N_i is the number of local data and ξ_i^N is a Laplacian noise.

Step 2. The category frequency distribution is used in two ways. First, the server uses all distinct categories to build the label encoder LE_j for column j. A label encoder is a table which maps all possible distinct values of a categorical column into their corresponding rank in one-hot encoding. Second, the frequency information is used to build an aggregated global frequency distribution X_j for column j. We use N_{all} denotes the sum of $\hat{N_i}$ across all clients. The global label frequency distribution X_j, $\hat{N_i}$ and N_{all} are needed to estimate the similarity of clients' local data for computing the clients' weights for model aggregation in GDTS_FL. For continuous variables, the local VGM models VGM_{ij} are used to estimate the global distribution of column j. The server uses VGM_{1j}, VGM_{2j}, ..., VGM_{Pj} to sample the datasets D_{1j}, D_{2j}, ..., D_{Pj} with $\hat{N}_1$, $\hat{N}_2$, ..., $\hat{N}_P$ data points where P is the number of clients. We define D_j = { D_{1j}, D_{2j}, ..., D_{Pj} }, then the server uses D_j to fit a new global VGM model VGM_j for column j[2]. VGM_j is used as the final encoder for column j.

Step 3. The server distributes all the column encoders LE_j and VGM_j to each client. Clients use this information to encode the local data and initialize the local models. Models initialized using the same encoders will have the same input/output layers.

C. Training GDTS_FL and GDTS_MD

For training of GDTS_FL, we propose a novel table-similarity aware weighting scheme. After model initialization, the federator hosted on the server uses the collected global data statistics to pre-compute weights for each client. These weights are used in training during the model aggregation (shown in Fig. 3c) to smooth convergence in the presence of skewed data across clients. The weights calculation process is illustrated in Fig. 4, the calculation details are as follows:

Step 0 is to build a $P \times Q$ divergence matrix $\boldsymbol{S}$ where P is the number of clients and Q is the number of columns. Each matrix element S_{ij} is the divergence between distribution of column j in client i and the estimated global distribution of column j. The metric used depends on the type of column.

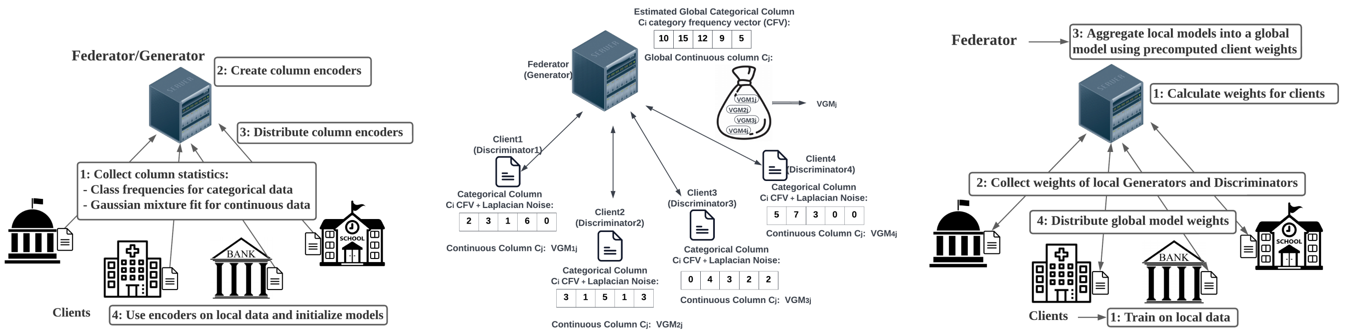
Categorical columns use the Jensen-Shannon Divergence (JSD) [14]. The JSD between two probability vectors p and q is defined as $\sqrt{\frac{D(p||m)+D(q||m)}{2}}$ where m is the point-wise mean of p and q, and D is the Kullback-Leibler divergence [10]. The JSD distance metric is symmetric and bounded between 0 and 1 enabling a hassle-free interpretation of results. For each categorical column j and client i we compute S_{ij} as JSD between X_{ij} and X_j, i.e., $S_{ij} = \text{JSD}(X_{ij}, X_j)$.

Continuous columns use the Wasserstein Distance (WD) [17]. The first Wasserstein distance between two distributions u and v is defined as $WD(u,v) = inf_{\pi \in \Gamma(u,v)} \int_{\mathbb{R}\times\mathbb{R}} |x-y| d\pi(x,y)$ where $\Gamma(u,v)$ is the set of probability distributions on $\mathbb{R} \times \mathbb{R}$ whose marginals are u and v on the first and second factors, respectively. It can be interpreted as the minimum cost to transform one distribution into another where the cost is given by amount of distribution to shift times the distance it must be shifted. For each continuous column j, we use the datasets D_{ij} created previously for each client i and D_j to compute S_{ij} as the WD between VGM_{ij} and VGM_j.

Step 1 normalizes the matrix $\boldsymbol{S}$ across the P clients for each table column j. This is done by dividing each matrix element by the sum of the elements in the corresponding matrix column. This step maintains the relative divergence between different clients with respect to the global column data distribution while allowing to give the same importance to all columns (all columns sum up to 1).

Step 2 aggregates the divergence across the different table columns j. This is done via a sum along the rows of the matrix. For each client i the resulting score SS_i can already represent the divergence between client and global data distribution, but it does not yet take into account possible difference in the amount of local data available at each client.

[2] It might be possible to fit the global model directly from the parameters of the local models by, e.g., adapting [2]. This is left for future work.


Federator/Generator
2: Create column encoders
3: Distribute column encoders
1: Collect column statistics:
- Class frequencies for categorical data
- Gaussian mixture fit for continuous data
SCHOOL
BANK
Clients
4: Use encoders on local data and initialize models
Estimated Global Categorical Column
Ci category frequency vector (CFV):
10 15 12 9 5
Federator
(Generator)
Global Continuous column Cj:
VGM1j
VGM2j
VGM3j
VGM4j
VGMj
Client1
(Discriminator1)
Categorical Column
Ci CFV + Laplacian Noise:
2 3 1 6 0
Continuous Column Cj: VGM1j
Client2
(Discriminator2)
Categorical Column
Ci CFV + Laplacian Noise:
3 1 5 1 3
Continuous Column Cj: VGM2j
Client3
(Discriminator3)
Categorical Column
Ci CFV + Laplacian Noise:
0 4 3 2 2
Continuous Column Cj: VGM3j
Client4
(Discriminator4)
Categorical Column
Ci CFV + Laplacian Noise:
5 7 3 0 0
Continuous Column Cj: VGM4j
Federator
3: Aggregate local models into a global model using precomputed client weights
1: Calculate weights for clients
2: Collect weights of local Generators and Discriminators
4: Distribute global model weights
SCHOOL
BANK
Clients
1: Train on local data

(a) Collect distributions, create encoders

(b) Global feature encoding of GDTS.

(c) Training of GDTS_FL

Fig. 3: Initialization and training process of GDTS_FL and GDTS_MD.

Step 3 fuses the divergence in data values and data quantity at each client. Step 3 first normalizes the divergence metric between 0 and 1 across the clients. Then it uses the complement to represent similarity instead of divergence and combines it with the ratio of local data available with respect to the global data, i.e., $\frac{\hat{N}_i}{N_{all}}$. The resulting SD_i take into account differences in both number of values and distribution of values of the local vs. global data.

Step 4 computes the final weights W_i. The W_i for each client i are obtained by passing the SD_i to a softmax function. W_i is the weight that the federator will use when aggregating the model from client i.

For training of GDTS_MD, after the step of global feature encoding, all clients can train their discriminators with local data. The generator takes the turn to link with each discriminator to compose a GAN and accumulates gradients. The generator does not update its weights until it calculates one round of gradients from all discriminators. At the end of each training round, to avoid discriminators overfit on local data, all discriminators randomly swap their weights with each other in a peer-to-peer way.

D. Validation Pipeline

To evaluate the quality of the synthesized data, we design a validation pipeline considering two key properties of the data: (1) statistical similarity and (2) machine learning utility.

1) Statistical similarity: We use two metrics to measure the statistical similarity between the real and synthetic data:

Average Jensen-Shannon divergence (Avg-JSD). Used for categorical columns. First, we compute the JSD between the synthetic and real data for each categorical column. Second, we average the obtained JSDs to obtain a compact comprehensible score, abbreviated as Avg-JSD.

Average Wasserstein distance (Avg-WD). Used for continuous columns[3]. Unlike JSD, WD is unbounded and can vary greatly depending on the scale of the data. To make the WD scores comparable across columns, before computing the WD we fit and apply a min-max normalizer to each continuous column in the real data and apply the same normalizer to the corresponding columns in the synthetic data. We average all column WD scores to obtain the final score.

2) Machine learning utility: F1-score is used to quantify the ML utility between the synthesized and real data

F1-score difference. We compare the performance achieved by 4 widely used ML algorithms: decision tree, random forest, logistic regression and MLP. This list can be easily adapted. The original data is split into *train dataset* and *test dataset*. We use the *train dataset* to train all GANs algorithms and synthesize a dataset of the same size. Then we use again the *train dataset* and the *synthetic dataset* to train two separate instances of the 4 ML models listed above and evaluate them by the same *test dataset*. The ML utility is measured via difference in F1-score between each model pair trained by *train dataset* and the *synthetic dataset*.

III. Experimental Analysis

Our algorithm GDTS_MD and GDTS_FL are evaluated on four commonly used datasets, and compared with centralized approach. Statistical similarity and ML utility are evaluated between real and synthetic data. Three ablation analyses are implemented to highlight the efficacy of the proposed client weighting strategies of GDTS_FL and the influence of Laplacian noise injected during categorical encoder initialization for GDTS_FL. A training time analysis is reported in the end, to show the time efficiency of GDTS_MD and GDTS_FL.

A. Experimental Setup

Datasets. We test our algorithm on four commonly used machine learning datasets. Adult, Covertype and Intrusion – are from the UCI machine learning repository [3], and Credit is from Kaggle [20]. Due to our computational limitation, we randomly sample 40k and 10k data as train and test datasets from each of above datasets.

Baselines. We compare GDTS_MD and GDTS_FL against 3 baselines: (i) **GDTS_FL without similarity weights**, (ii) **GDTS_FL without Laplacian noise** and (iii) **centralized approach** abbreviated as **GDTS_FL\SW**, **GDTS_FL\LN** and **Centralized**, respectively. **GDTS_FL\SW** removes the data

[3] We use WD over JSD for continuous columns since JSD is not well-defined when the synthetic values lie outside of the original value range from the real dataset, i.e., the KL divergence is not defined when comparing the similarity of probability distributions with non-overlapping support.

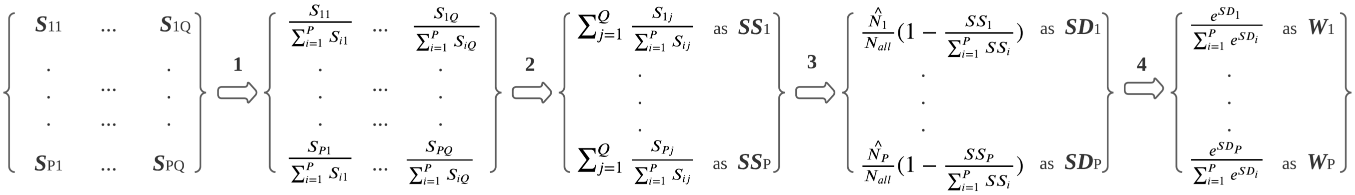

S_{11} ... S_{1Q}
S_{P1} ... S_{PQ}
1
$\frac{S_{11}}{\sum_{i=1}^{P} S_{i1}}$ ... $\frac{S_{1Q}}{\sum_{i=1}^{P} S_{iQ}}$
$\frac{S_{P1}}{\sum_{i=1}^{P} S_{i1}}$ ... $\frac{S_{PQ}}{\sum_{i=1}^{P} S_{iQ}}$
2
$\sum_{j=1}^{Q} \frac{S_{1j}}{\sum_{i=1}^{P} S_{ij}}$ as SS_1
$\sum_{j=1}^{Q} \frac{S_{Pj}}{\sum_{i=1}^{P} S_{ij}}$ as SS_P
3
$\frac{\hat{N}_1}{N_{all}}(1 - \frac{SS_1}{\sum_{i=1}^{P} SS_i})$ as SD_1
$\frac{\hat{N}_P}{N_{all}}(1 - \frac{SS_P}{\sum_{i=1}^{P} SS_i})$ as SD_P
4
$\frac{e^{SD_1}}{\sum_{i=1}^{P} e^{SD_i}}$ as W_1
$\frac{e^{SD_P}}{\sum_{i=1}^{P} e^{SD_i}}$ as W_P

Fig. 4: Weights calculation of GDTS_FL. Starting with a divergence matrix: (1) normalizes scores by column; (2) aggregates the scores per client (by row); (3) incorporates differences in local data quantities; and, (4) performs the final weight normalization.

similarity factor in the calculation of the aggregation weights for GDTS_FL, the aggregation weight of client i equals to the ratio of the number of data rows to the global number of data rows, i.e., $\frac{\hat{N}_i}{N_{all}}$. **GDTS_FL\LN** sends in the information such as category frequency vector and number of data rows without injecting Laplacian noise in feature encoding step. The aim of all the experiments is to learn a CTGAN model from distributed clients using above mentioned frameworks on the basis of CTGAN's default settings for encoding features [22]. We use VGM encoders for each continuous column and one-hot-encoding for categorical columns.

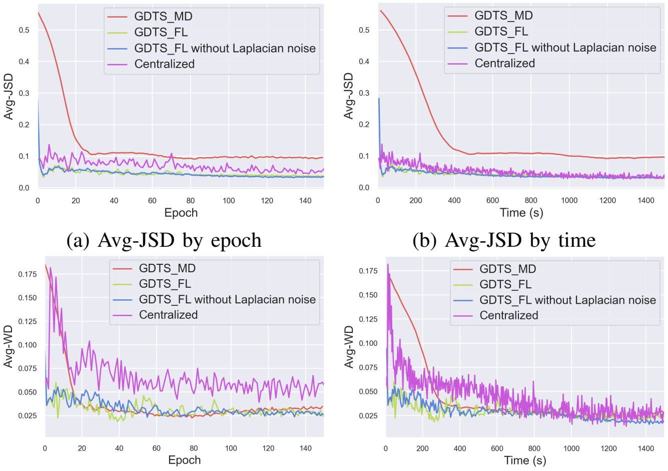
GDTS_MD clients swap discriminator models with each other at the end of each training epoch [7]. For a fair comparison with GDTS_MD, we force GDTS_FL and its variants to share the model weights with the federator at the end of each training epoch. Due to this, the notion *per round* commonly referred in FL studies equals to *per epoch* in this paper. Due to the different learning speed per epoch of the all frameworks, for a fair comparison we preset the number of epochs for each algorithm to ensure that the total training time is similar. We use 500 epochs for centralized approach, GDTS_FL and its variants, and 150 epochs for GDTS_MD. We repeat each experiment 3 times and report the average.

Testbed. Experiments are run under Ubuntu 20.04 on two machines. Each machine is equipped with 32 GB memory, RTX 2080 Ti GPU and 10-core Intel i9 CPU. The machine are interconnected via 1G Ethernet links. One machine hosts the server, the other all the clients. When not otherwise stated, both federator and clients use the GPU for training.

Evaluation Metrics Recall the validation pipeline in Sec. II-D. Two factors are considered (1) statistical similarity and (2) machine learning utility. Avg-JSD and Avg-WD are used for evaluating statistical similarity between real and synthetic data for categorical and continuous columns. And difference of F1-score between real and synthetic data is reported for ML utility.

B. Result Analysis

We first designs an experiment where all the clients contain the whole dataset, this is to test the performance of each framework under the ideal case. Then we implement a scenario where data on clients are IID, but quantities of data are highly imbalanced across all the clients. The objective of this experiment is to show the effect of our model aggregation weighting method. In the end, an ablation analysis is designed where one of the clients has much higher amount of data, but the data quality is low. We want to show the efficacy of table-similarity aware weighting method in the calculation of model aggregation weights.



(a) Avg-JSD by epoch

(b) Avg-JSD by time

(c) Avg-WD by epoch

(d) Avg-WD by time

Fig. 5: GDTS_MD, GDTS_FL, GDTS_FL\LN and Centralized: 5 clients each with a *complete* data copy.

1) Ideal case of full dataset: This experiment uses one server (or federator) and 5 clients. Each client is provided with a copy of the full real dataset. This represents the ideal case with perfectly identical clients, i.e. each client has identical IID data. We compare in particular GDTS_FL, GDTS_MD and Centralized. Since in this case the aggregation weights in GDTS_FL are the same, we skip the baselines: GDTS_FL\SW. In order to show the influence of Laplacian noise injected in feature encoding step, we include the baseline: GDTS_FL\LN.

Results for the Intrusion dataset are shown in Fig. 5. Avg-JSD and Avg-WD are presented by both epoch and time (in seconds) as different architectures spend vastly different time per epoch. For categorical columns, GDTS_FL converges faster both by epoch and by time (see Fig.5a and 5b). Moreover, the Avg-JSD of GDTS_MD converges quite slowly after epoch 24. For continuous columns, from the perspective of number of epochs, Avg-WD for GDTS_FL converges faster at the beginning, then becomes slightly worse than the Avg-WD for GDTS_MD (see Fig. 5c). However, inspecting the result by time, GDTS_FL not only converges faster, but also achieves a slightly lower Avg-WD than the other two architectures

TABLE I: Results for GDTS_MD, GDTS_FL, GDTS_FL\LN and Centralized: 5 clients each having a complete data copy.

Dataset	**Avg JSD**				**Avg WD**				**F1-score diff.**			
	MD	**FL**	**FL\LN**	**Centr.**	**MD**	**FL**	**FL\LN**	**Centr.**	**MD**	**FL**	**FL\LN**	**Centr.**
Adult	0.072	0.061	**0.059**	0.117	0.014	0.014	**0.012**	0.015	0.101	0.050	**0.026**	0.056
Covertype	0.035	0.021	**0.018**	0.075	0.022	0.022	**0.021**	0.086	0.493	0.390	**0.376**	0.460
Credit	0.083	0.004	**0**	0.012	**0.006**	**0.006**	**0.006**	0.041	0.064	0.070	**0.050**	0.075
Intrusion	0.095	0.032	**0.031**	0.032	0.027	0.024	**0.020**	0.026	0.085	0.076	**0.074**	0.075

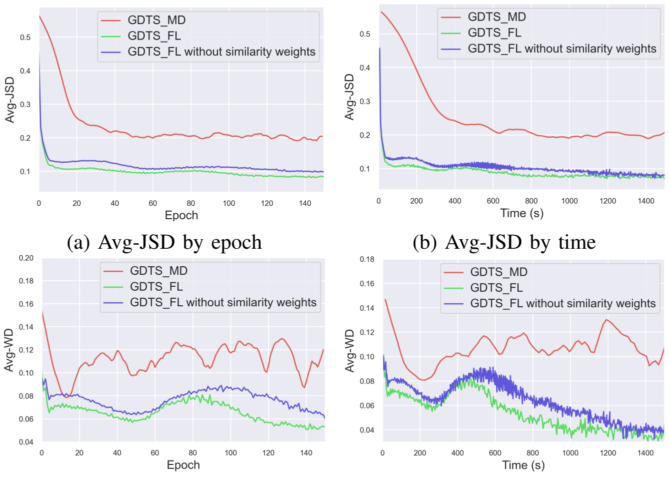
TABLE II: Final similarity for GDTS_MD, GDTS_FL and GDTS_FL without similarity weights: 4 clients have 10k sampled *IID* data, 1 client has 40k rows of *Non IID* data.

Dataset	**Avg JSD**			**Avg WD**			**F1-score diff.**		
	MD	**FL**	**FL\SW**	**MD**	**FL**	**FL\SW**	**MD**	**FL**	**FL\SW**
Adult	0.454	**0.152**	0.265	0.132	**0.034**	0.040	0.230	**0.084**	0.117
Covertype	0.097	**0.056**	0.067	0.151	**0.054**	0.062	0.636	**0.476**	0.496
Credit	0.076	**0.030**	0.068	0.041	**0.012**	0.017	0.165	**0.149**	0.165
Intrusion	0.210	**0.071**	0.075	0.131	**0.039**	0.047	0.557	**0.108**	0.148

(see Fig. 5d) in the end. The performance gap between the centralized approach and GDTS_FL may look counter-intuitive. However, similar results are reported by FeGAN [6]. The reason is that, GDTS_FL can see the data five times per epoch as compared to the centralized approach which only sees it once. This boosts the diversity of samples seen by GDTS_FL thereby providing superior performance. Regards to the impact of Laplacian noise, it is minor.

We summarize the final statistical similarity and ML utility results of all three approaches and all four datasets in Tab. I. The scores are taken at the time in seconds when Centralized finishes 500 epochs training. One can see that excluding GDTS_FL\LN, GDTS_FL consistently achieves same or higher similarity (lower Avg-JSD and Avg-WD values) than the other two approaches. Similar results are obtained for ML utility. F1-score difference shows that GDTS_FL outperforms (lower F1-score difference) GDTS_MD and Centralized on all datasets except Intrusion where GDTS_FL is slightly worse than Centralized. GDTS_FL\LN marginally improves the performance of GDTS_FL on all metrics, which is expected. Because its calculation of aggregation weights is more accurate. Meanwhile, we also observe that the influence is not significant, that is because we choose a less strict privacy budget (i.e., ϵ) in Laplace mechanism. A lower privacy budget can further worsen the aggregation weights calculation.

2) Imbalanced amount of Non IID data: Recall the weights calculation process in Fig. 4. The SD_i is composed of two parts: (1) the ratio of the number of data rows locally available at the client i to global number of data rows, i.e., $\frac{\hat{N}_i}{N_{all}}$; and (2) the similarity calculated between the local data distribution of client i and the global distribution, i.e., $1 - \frac{SS_i}{\sum_{i=1}^{P} SS_i}$. The contribution of data number ratio part is intuitive. Therefore in this ablation analysis, we design a scenario where for GDTS_FL, the client weights are only calculated using data number ratio of each client, without using the similarity weights. To better show the importance of similarity weights, we design a specific scenario for this experiment. Still with 5 clients, 4 of them containing 10k IID data sampled from original data, the last client is modified to contain 40k rows of data by repeating one row sampled from the original dataset 40k times. One can imagine, this last client has a large number of rows, but contains little information in them. Fig. 6 shows the results on Intrusion dataset. One can already notice that this scenario badly hits GDTS_MD since it treats all clients equally while updating the generator's weights. Moreover, for the results in Fig. 6c and 6d, one can see the client with 40k repeated data introduces oscillation to the curves of GDTS_FL with and without the similarity weights. As expected, the curve for GDTS_FL\SW naturally performs worse than GDTS_FL. Results in Tab. II (scores are taken at the time when GDTS_MD finishes 150 epochs training) show that GDTS_FL averagely outperforms GDTS_MD and GDTS_FL\SW by 44.74% and 17.24 on F1-score with all datasets. Therefore, similarity weights in GDTS_FL give more stability for model convergence.



(a) Avg-JSD by epoch

(b) Avg-JSD by time

(c) Avg-WD by epoch

(d) Avg-WD by time

Fig. 6: GDTS_MD, GDTS_FL and GDTS_FL\SW: 4 clients have 10k *IID* data, 1 client has 40k sampled *Non IID* data.

IV. Conclusion

This paper proposes, GDTS, a first of its kind decentralized architecture and prototype for tabular GANs, overcoming specific challenges related to tabular data. Two main features of GDTS are (i) privacy preserving feature encoding to enable model initialization across heterogeneous data sources, and (ii) table-similarity aware weighting for merging local models. We extensively evaluate two variants of GDTS, GDTS_MD and GDTS_FL, using a SOTA tabular GAN and compare it with four baselines. Our results show that GDTS can generate synthetic tabular data with high similarity to the original data, even in the challenging case of Non-IID data among clients.

References

[1] S. O. Arik and T. Pfister. Tabnet: Attentive interpretable tabular learning. *arXiv preprint arXiv:1908.07442*, 2019.

[2] P. Bruneau, M. Gelgon, and F. Picarougne. Parameter-based reduction of gaussian mixture models with a variational-bayes approach. In *IEEE International Conference on Pattern Recognition ICPR*, pages 1–4, 2008.

[3] D. Dua and C. Graff. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2017.

[4] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, aug 2014.

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, page 2672–2680, Cambridge, MA, USA, 2014.

[6] R. Guerraoui, A. Guirguis, A.-M. Kermarrec, and E. Le Merrer. Fegan: Scaling distributed gans. In *ACM/IFIP Middleware, 2002*, 2020.

[7] C. Hardy, E. Le Merrer, and B. Sericola. Md-gan: Multi-discriminator generative adversarial networks for distributed datasets. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 866–877, 2019.

[8] B. Hitaj, G. Ateniese, and F. Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 603–618, 2017.

[9] J. Huang, R. Talbi, Z. Zhao, S. Boucchenak, L. Y. Chen, and S. Roos. An exploratory analysis on users' contributions in federated learning. In *2020 Second IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pages 20–29, 2020.

[10] J. M. Joyce. *Kullback-Leibler Divergence*, pages 720–722. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[11] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405, 2019.

[12] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of stylegan. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8107–8116, 2020.

[13] J. Lee, J. Hyeong, J. Jeon, N. Park, and J. Cho. Invertible tabular gans: Killing two birds with one stone for tabular data synthesis. *Advances in Neural Information Processing Systems*, 34:4263–4273, 2021.

[14] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[15] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE symposium on security and privacy (SP)*, pages 691–706. IEEE, 2019.

[16] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim. Data synthesis based on generative adversarial networks. *Proc. VLDB Endow.*, 11(10):1071–1083, 2018.

[17] A. Ramdas, N. G. Trillos, and M. Cuturi. On wasserstein two-sample testing and related families of nonparametric tests. *Entropy*, 19(2), 2017.

[18] M. Rasouli, T. Sun, and R. Rajagopal. Fedgan: Federated generative adversarial networks for distributed data. *CoRR*, abs/2006.07228, 2020.

[19] S. Semeniuta, A. Severyn, and S. Gelly. On accurate evaluation of gans for language generation. *arXiv preprint arXiv:1806.04936*, 2018.

[20] M. L. G. ULB. Kaggle - anonymized credit card transactions labeled as fraudulent or genuine. https://www.kaggle.com/mlg-ulb/creditcardfraud, 2018.

[21] H. Wu, Z. Zhao, L. Y. Chen, and A. Van Moorsel. Federated learning for tabular data: Exploring potential risk to privacy. In *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, pages 193–204. IEEE, 2022.

[22] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. Modeling tabular data using conditional gan. In *Advances in Neural Information Processing Systems, 2019*, volume 32, pages 7335–7345. Curran Associates, Inc., 2019.

[23] Z. Zhao, A. Kunar, R. Birke, and L. Y. Chen. Ctab-gan: Effective table data synthesizing. In *Proceedings of The 13th Asian Conference on Machine Learning*, volume 157, pages 97–112, 17–19 Nov 2021.

[24] Z. Zhao, A. Kunar, R. Birke, and L. Y. Chen. Ctab-gan+: Enhancing tabular data synthesis. *arXiv preprint arXiv:2204.00401*, 2022.