Relational Data Synthesis using Generative Adversarial Networks: A Design Space Exploration

Experiment and Analysis Paper

Ju Fan†, Tongyu Liu†, Guoliang Li§, Junyou Chen†, Yuwei Shen†, Xiaoyong Du†

† *Renmin University of China, Beijing, China* § *Tsinghua University, Beijing, China*
{fanj,ltyzzz,kanamemadokam,rmdxsyw,duyong}@ruc.edu.cn; liguoliang@tsinghua.edu.cn

ABSTRACT

The proliferation of big data has brought an urgent demand for privacy-preserving data publishing. Traditional solutions to this demand have limitations on effectively balancing the tradeoff between privacy and utility of the released data. Thus, the database community and machine learning community have recently studied a new problem of relational data synthesis using generative adversarial networks (GAN) and proposed various algorithms. However, these algorithms are not compared under the same framework and thus it is hard for practitioners to understand GAN's benefits and limitations. To bridge the gaps, we conduct so far the most comprehensive experimental study that investigates applying GAN to relational data synthesis. We introduce a unified GAN-based framework and define a space of design solutions for each component in the framework, including neural network architectures and training strategies. We conduct extensive experiments to explore the design space and compare with traditional data synthesis approaches. Through extensive experiments, we find that GAN is very promising for relational data synthesis, and provide guidance for selecting appropriate design solutions. We also point out of limitations of GAN and identify future research directions. We make all codes and datasets public for future research.

1. INTRODUCTION

The tremendous amount of big data does not automatically lead to be easily accessed. The difficulty in data access is still one of the top barriers of many data scientists, according to a recent survey [21]. In fact, organizations, such as governments and companies, have intention to publish data to the public or share data to partners in many cases, but they are usually restricted by regulation and privacy concerns. For example, a hospital wants to share its electronic health records to a university for research purpose. However, the data sharing must be reviewed by its legal department and institutional review boards to avoid disclosure of patient privacy, which usually takes several months without guarantee of approval [20].

To address the difficulties, *privacy-preserving data publish* has been extensively studied recently to provide a safer way for data sharing [6, 25, 11, 1, 46, 47]. However, the existing solutions suffer from the limitations on effectively balancing privacy and utility of the released data [31]. Therefore, efforts have been made recently in the database and machine learning communities to apply *generative adversarial networks (GAN)* to relational data synthesis [7, 42, 43, 31, 3, 9, 26]. The main advantages of GAN are as follows. First, different from the conventional methods [6, 25, 1, 46] that inject noise to the original data, GAN utilizes neural networks to generate "fake" data directly from noise. Thus, there is no *one-to-one* relationship between real and synthetic data, which reduces the risk of re-identification attacks [31]. Moreover, the adversarial learning mechanism of GAN enables the synthetic data to effectively preserve utility of the original data for supporting down-streaming applications, such as classification and regression.

However, compared with the success of using GAN for image generation [27], GAN-based relational data synthesis is still in its infancy stage. Despite some very recent attempts [7, 42, 43, 31, 3, 9, 26], as far as we know, the proposed methods are not compared under the same framework and thus it is hard for practitioners to understand GAN's benefits and limitations. To bridge the gaps, in this paper, we provide a comprehensive experimental study that examines applying GAN to relational data synthesis. We introduce a general framework that can unify the existing solutions for GAN-based data synthesis. Based on the framework, we conduct extensive experiments to systemically investigate the following two key questions.

Firstly, it remains an unresolved question on how to effectively apply GAN to relational data synthesis. It is worth noting that relational data has its own characteristics that make the adoption very challenging. (i) Relational data has mixed data types, including categorical and numerical attributes. (ii) Different attributes have correlations. (iii) Many real-world datasets have highly imbalanced data distribution. Thus, the state-of-the-art GAN design for image synthesis (e.g., DCGAN [35]) may not perform well for relational data. In this paper, we review the existing solutions that realize GAN, including neural network design and training strategies, and define a *design space* for GAN-based data synthesis by providing a categorization of the solutions. Through exploring the design space, we systemically evalu-

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

ate the solutions on datasets with various types and provide insightful experimental findings.

The second question is whether GAN is more helpful than the existing approaches to relational data synthesis. To answer this, this paper considers various baseline approaches, including a representative deep generative model, variational auto-encoder (VAE) [23, 37], and the state-of-the-art data synthesis approach using statistical models [46, 47]. To provide a comprehensive comparison, we evaluate their performance on both privacy and the utility of the synthetic data. Moreover, we also examine whether GAN can support provable privacy protection, i.e., differential privacy [13]. Based on the comparison, we analyze the benefits and limitations of applying GAN to relational data synthesis.

To summarize, we make the following contributions.

(1) We conduct so far the most comprehensive experimental study for applying GAN to relational data synthesis. We formally define the problem and review the existing approaches (Section 2). We introduce a unified framework and define a design space that summarizes the solutions for realizing GAN (Sections 3, 4 and 5), which can help practitioners to easily understand how to apply GAN.

(2) We empirically conduct a thorough evaluation to explore the design space and compare with the baseline approaches (Section 6). We make all codes and datasets in our experiments public at Github[1]. We provide extensive experimental findings and reveal insights on strength and robustness of various solutions, which provide guidance for an effective design of GAN.

(3) We find that GAN is highly promising for relational data synthesis, as it empirically provides better tradeoff between synthetic data utility and privacy. We point out its limitations and identify research directions (Section 8).

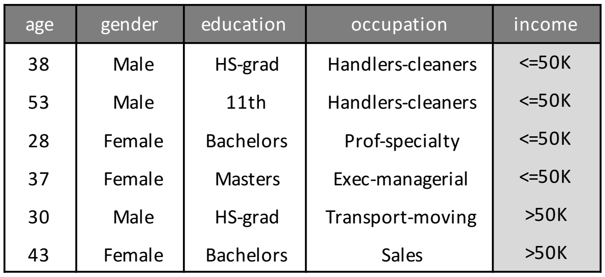
2. RELATIONAL DATA SYNTHESIS

2.1 Problem Formalization

This paper focuses on a relational table $\mathcal{T}$ of n records, i.e., $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$. We use $\mathcal{T}[j]$ to denote the j-th attribute (column) of table $\mathcal{T}$ and $t[j]$ to denote the value of record t's j-th attribute. In particular, we consider both categorical (nominal) and numerical (either discrete or continuous) attributes in this paper. Figure 1 shows an example table with six records and five attributes, where attribute `age` is numerical while other attributes are categorical.

We study the problem of synthesizing a "fake" table $\mathcal{T}'$ from the original $\mathcal{T}$, with the objective of preserving data utility and protecting privacy, as introduced as follows.

(1) *Data utility* is highly dependent to the specific need of the synthetic data for down-streaming applications. This paper focuses on the specific need on using the fake table to train machine learning (ML) models, which is commonly considered by recent works [7, 42, 43, 31, 3, 9, 26]. This means that an ML model trained on the fake table should achieve similar performance as that trained on $\mathcal{T}$. For simplicity, this paper considers classification models in supervised learning, such as decision tree and logistic regression.



age	gender	education	occupation	income
38	Male	HS-grad	Handlers-cleaners	<=50K
53	Male	11th	Handlers-cleaners	<=50K
28	Female	Bachelors	Prof-specialty	<=50K
37	Female	Masters	Exec-managerial	<=50K
30	Male	HS-grad	Transport-moving	>50K
43	Female	Bachelors	Sales	>50K

Figure 1: An example table for data synthesis.

Formally, we represent the original table as $\mathcal{T} = [X; Y]$, where each $x_i \in X$ and each $y_i \in Y$ respectively represent *features* and *label* of the corresponding record t_i. We use $\mathcal{T}$ to train a classifier $f : X \rightarrow Y$ that maps $x_i \in X$ to its predicted label $f(x_i)$. Then, we evaluate the performance of f on a test set $\mathcal{T}_{\texttt{test}} = [X_{\texttt{test}}; Y_{\texttt{test}}]$ using a specific metric $\texttt{Eval}(f|\mathcal{T}_{\texttt{test}})$. Some representative metrics include F1 score and Area Under the ROC Curve (AUC), which are defined in Section 6. Similarly, we can train a classifier f' on the synthetic table $\mathcal{T}'$ and evaluate the classifier on the same $\mathcal{T}_{\texttt{test}}$ to obtain its performance $\texttt{Eval}(f'|\mathcal{T}_{\texttt{test}})$. Thus, the utility of $\mathcal{T}'$ is measured by the difference between these two classifiers' performance metrics, i.e.,

$$\texttt{Diff}(\mathcal{T}, \mathcal{T}') = \mid \texttt{Eval}(f|\mathcal{T}_{\texttt{test}}) - \texttt{Eval}(f'|\mathcal{T}_{\texttt{test}}) \mid . \quad (1)$$

EXAMPLE 1 (SYNTHETIC DATA UTILITY). *Consider an example table $\mathcal{T}$ in Figure 1, where* `income` *indicates the label with two possible values:* 0 *(*`income` $\leq 50K$*) and* 1 *(*`income` $> 50K$*). We use $\mathcal{T}$ to train a data synthesizer G and generate a fake table $\mathcal{T}'$ via G. We train models f and f' to predict* `income` *on $\mathcal{T}$ and $\mathcal{T}'$ respectively, and evaluate these models on a new table $\mathcal{T}_{\texttt{test}}$ as the test set. We measure the performance difference of these two models as the utility difference* $\texttt{Diff}(\mathcal{T}, \mathcal{T}')$ *between the original $\mathcal{T}$ and synthetic table $\mathcal{T}'$. Intuitively, the lower the difference* $\texttt{Diff}(\mathcal{T}, \mathcal{T}')$ *is, the better the synthetic table preserves the data utility.*

(2) *Privacy risk* evaluation for synthetic table $\mathcal{T}'$ is also an independent research problem. This paper adopts two commonly-used metrics in the existing works [32, 26, 28], namely hitting rate and distance to the closest record (DCR). Intuitively, the metrics measure the likelihood that the original data records can be re-identified by an attacker. We will introduce their formal definitions in Section 6.

Remarks. Real-world practice in data synthesis may have various needs on preserving data utility, such as answering OLAP queries, training supervised ML models, supporting decision-makings, etc. While this paper only focuses on the specific need on training classification models, we will take the research on other needs in the future work.

2.2 Related Works for Data Synthesis

As synthetic data is useful in many applications, data synthesis has been extensively studied in the last decades. Existing approaches to data synthesis can be broadly classified into *statistical model* and *neural model*. The statistical approach aims at modeling a joint multivariate distribution for a dataset and then generating fake data by sampling from the distribution. To effectively capture dependence between variates, existing works utilize copulas [24, 33], Bayesian networks [46, 47], Gibbs sampling [32] and Fourier decompositions [4]. Neural models have been recently emerging to synthesize relational data. Existing works aim to use deep generative models to approximate the distribution of an original dataset. To this end, some studies devise deep de-noising autoencoders [14] and variational autoencoders

[1] https://github.com/ruclty/Daisy

(VAE) [40], while more attentions are paid on generative adversarial networks (GAN) [7, 42, 43, 31, 3, 9, 26].

However, despite the aforementioned attempts on GAN-based relational data synthesis, existing works have not systemically explored the design space, as mentioned previously. Thus, this paper conducts an experimental study to systemically investigate the design choices and compare with the state-of-the-art statistical approaches for data synthesis. Note that, besides data synthesis, private data release can also be achieved by anonymization [6, 25] and perturbation [11, 1]. However, the existing study [31] has shown that GAN-based data synthesis outperforms these techniques.

2.3 Generative Adversarial Networks (GAN)

Generative adversarial networks (GAN) [15, 27], are a kind of deep generative models, which have achieved breakthroughs in many areas, such as image generation [35, 30, 8], and sequence generation [45, 44]. Typically, GAN consists of a generator G and a discriminator D, which are competing in an adversarial process. The generator $G(\mathbf{z}; \theta_g)$ takes as input a random noise $\mathbf{z} \in \mathbb{R}^z$ and generates synthetic samples $G(\mathbf{z}) \in \mathbb{R}^d$, while the discriminator $D(\boldsymbol{t}; \theta_d)$ determines the probability that a given sample comes from the real data instead of being generated by G. Intuitively, the optimal D could distinguish real samples from fake ones, and the optimal G could generate indistinguishable fake samples which make D to randomly guess. Formally, G and D play a minimax game with value function $V(G, D)$, i.e.,

$$\min_G \max_D V(G, D) = \mathbb{E}_{\boldsymbol{t} \in p_{data}(\boldsymbol{t})}\big[\log D(\boldsymbol{t})\big] + \mathbb{E}_{\mathbf{z} \in p_z(\mathbf{z})}\big[1 - \log D(G(\mathbf{z}))\big], \quad (2)$$

where p_{data} is the distribution of the real samples transformed from our relational table $\mathcal{T}$ and p_z is the distribution of the input noise $\mathbf{z}$. Minibatch stochastic gradient descent algorithms can be applied to optimizing parameters θ_g and θ_d in generator G and discriminator D respectively.

3. GAN-BASED SYNTHESIS OVERVIEW

Relational data has its own characteristics that make the adoption of GAN to data synthesis challenging. First, relational data has *mixed* data types, including continuous, discrete and categorical attributes. However, GAN models each data tuple as a sample $\boldsymbol{t} \in \mathbb{R}^d$ of numerical values. Thus, it is non-trivial to transform a record into such a sample[2]. Second, different attributes in relational data usually have correlations. It remains challenging to enable the generator to capture such correlations. Third, most real-world data has highly *imbalanced* label distribution. Considering our example table in Figure 1, most records have `income` $\leq 50K$, while only a few with `income` $> 50K$. This increases the difficulty of relational data synthesis, especially for records with minority labels. To address these challenges, we introduce a framework that unifies the existing solutions for applying GAN to relational data synthesis, and present a design space that categorizes the solutions in Section 3.2.

3.1 Framework of GAN-based Synthesis

Figure 2 shows a unified framework of GAN-based relational data synthesis. It takes a relational table $\mathcal{T}$ as input and generates a table $\mathcal{T}'$ of synthetic data in three phases.

Phase I - Data Transformation. This phase aims at preparing *input data* for the subsequent GAN model training. Specifically, it transforms each record $t \in \mathcal{T}$ with mixed attribute types into a sample $\boldsymbol{t} \in \mathbb{R}^d$ of numerical values, which can be then fed into neural networks in GAN. As different neural networks may have different requirements for the input data, $\boldsymbol{t}$ can be in the form of either *matrix* (e.g., input of convolutional neural networks) or *vector* (e.g., input of fully-connected neural networks). Moreover, to address the issue of mixed attribute types, this phase should consider how to encode categorical attributes to numerical values, and normalize numerical values to appropriate ranges that fit the subsequent neural networks.

Phase II - GAN Model Training. This phase aims at training a deep generative model G. Specifically, G takes as input a random noise $\mathbf{z} \in \mathbb{R}^z$ and generates synthetic sample $\boldsymbol{t}' = G(\mathbf{z}) \in \mathbb{R}^d$. Meanwhile, a Sampler picks a sample $\boldsymbol{t}_i$ from the data prepared by the previous phase. Then, fed with both real and synthetic samples, our discriminator D determines the probability that a given sample is real. By iteratively applying minibath stochastic gradient descent, parameters of both G and D are optimized, and thus G could be improved towards generating indistinguishable samples that fool D. One key technical issue here is to design effective neural networks for G that can capture correlations among attributes. Moreover, considering imbalanced label distribution of real datasets, our framework also supports conditional GAN [30] that also feeds a target label to both generator and discriminator as their input, so as to "guide" them for generating records with the label.

Phase III - Synthetic Data Generation. This phase utilizes G, which is well trained in the previous phrase, to generate a synthetic table $\mathcal{T}'$. It repeatedly feeds G with the prior noise $\boldsymbol{z}$ (as well as target label), which generates a set of synthetic samples $\{\boldsymbol{t}'\}$. Next, it adopts the same data transformation scheme used in Phase I to convert the samples back into records that then compose $\mathcal{T}'$.

Example 2 (Framework). *Considering our example table in Figure 1, the framework firstly transforms each record in the table into a sample. Suppose that we adopt a simple scheme that transforms a record into a vector with ordinal encoding for categorical attributes*[3]. *Given this scheme, the first record in Figure 1 is transformed into* $[38, 0, 0, 0, 0]$. *Then, the framework uses the transformed samples to train the GAN model for obtaining an optimized generator G. Finally, it leverages G to generate samples, e.g.,* $[40, 1, 2, 1, 1]$, *and transforms the samples back to synthetic records, e.g.,* (40, `Female`, `Bachelors`, `Prof-specialty`, $> 50K$).

3.2 Categorization of Design Choices

We provide a categorization of design solutions for each component in our framework, as summarized in Figure 3.

Data transformation. We investigate three issues for data transformation: (1) representation form of sample $\boldsymbol{t}$, i.e., matrix or vector, (2) encoding schemes for categorical attributes, i.e., ordinal or one-hot encoding, and normalization schemes for numerical attributes, i.e., simple normalization or normalization using Gaussian Mixture Model (GMM). More details of data transformation are in Section 4.

[2] We use t and $\boldsymbol{t}$ to respectively denote a record in $\mathcal{T}$ and the d-dimension sample transformed from record t in the paper.

[3] We will discuss more complicated transformation later.

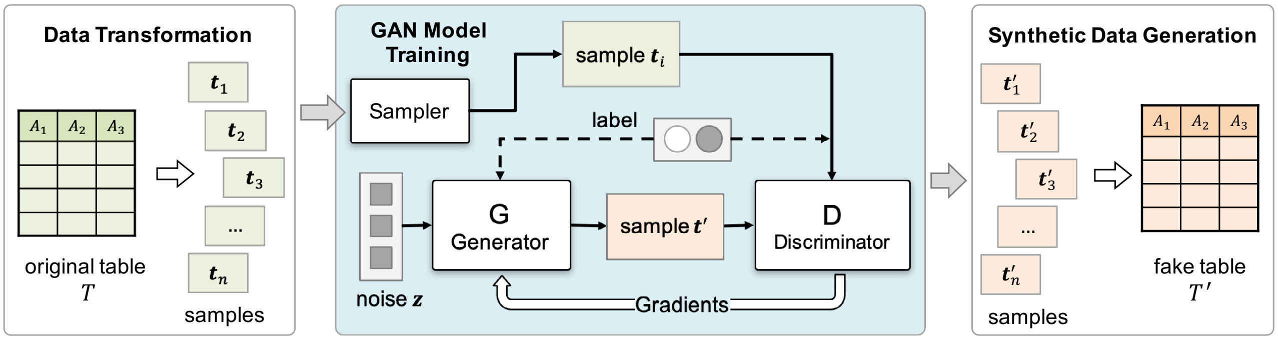


Figure 2: Framework of relational data synthesis using GAN.

Components		Design Solutions		
Data Transformation		*Sample Form:* (1) Matrix (2) Vector	*Categorical:* (1) Ordinal (2) One-hot	*Numerical:* (1) Norm (2) GMM
Neural Networks	Matrix as Input	*Generator:* (1) CNN-based	*Discriminator:* (1) CNN-based	
	Vector as Input	*Generator:* (1) MLP-based (2) LSTM-based	*Discriminator:* (1) MLP-based (2) LSTM-based	
Training Algorithm		(1) Vanilla + KL (2) WGAN		
Conditional GAN		*Condition:* (1) None (2) Label	*Sampling:* (1) Random (2) Label-aware	
Differential Privacy		(1) None (2) DPGAN (adding noise to gradients)		

Figure 3: A categorization of design solutions.

Neural networks. Neural network architecture is required to capture correlations among different attributes. To this end, we examine representative neural networks according to the form of input sample $\boldsymbol{t}$. Given matrix-formed samples, we examine Convolutional Neural Networks (CNN) for both generator G and discriminator D, in which G is a deconvolution process and D is a convolution process. Such solution has been proven to be effective for synthetic image generation [35]. For vector-formed samples, we can use fully-connected neural networks (MLP) [16] to realize G that uses multiple layers to transform random noise $\boldsymbol{z}$ to sample $\boldsymbol{t}'$. Alternatively, we can model record synthesis as a *sequence generation* process that generates attributes separately in sequential time-steps. In this case, we may choose some variant of recurrent neural networks, such as long short-term memory (LSTM) networks [17] to realize G. Similarly, we can also use MLP or LSTM to realize discriminator D. More details can be referred to Section 5.1.

Training algorithm. Minibatch-based stochastic gradient descent (SGD) strategy is applied to train GAN for optimizing the parameters in G and D. However, different training algorithms have been proposed with various loss functions and variants of SGD optimizer, such as `Adam` and `RMSProp`. This paper investigates two alternatives to train GAN: (1) the vanilla training algorithm [15] with an improved loss function and (2) Wasserstein GAN (WGAN) training [2]. See Section 5.2 for more details of these algorithms.

Conditional GAN. We examine the adoption of conditional GAN [30] that encodes a label as a condition vector $\boldsymbol{c}$ to guide G (D) to generate (discriminate) samples with the label. We evaluate the performance of GAN with/without label as a condition. Moreover, we also investigate different sampling strategies (i.e., Sampler in Figure 2): (1) random sampling as commonly used in GAN training, and label-aware sampling that gives fair opportunity for samples with different labels. See Section 5.3 for more details.

Differential privacy. This component ensures *differential privacy* [13], which is well-adopted formalization of data privacy, to our GAN-based framework. Intuitively, although G does not access the real data $\mathcal{T}$ (only D accesses $\mathcal{T}$ via Sampler), G may still implicitly disclose privacy information as the gradients for optimizing G is computed based on D. Thus, we adopt the DPGAN model [41] in the GAN training process, as elaborated in Section 5.4.

4. DATA TRANSFORMATION

Given a record t from our relational table $\mathcal{T}$, data transformation converts it into a sample $\boldsymbol{t} \in \mathbb{R}^d$. To this end, it processes each attribute $t[j]$ in t independently to transform $t[j]$ into a vector $\boldsymbol{t}_j$. Then, it generates $\boldsymbol{t}$ by combining all the attribute vectors. Next, we first describe the schemes for both categorical and numerical attributes, and then discuss how to combine multiple attributes.

Categorical attribute transformation. A categorical attribute is the one that has two or more categories as its values, e.g., `gender` with two categories `Female` and `Male`. In machine learning, a categorical attribute is typically preprocessed by label encoding methods. Specifically, we consider two commonly-used label encoding schemes.

1) Ordinal encoding. Given a categorical attribute $\mathcal{T}[j]$, this method assigns an ordinal integer value to each category of $\mathcal{T}[j]$, e.g., starting from 0 to $|\mathcal{T}[j]| - 1$ ($|\mathcal{T}[j]|$ is value domain of $\mathcal{T}[j]$). In such a way, after ordinal encoding, $\mathcal{T}[j]$ is equivalent to a discrete numeric attribute.

2) One-hot encoding. This method first assigns each category of categorical attribute $\mathcal{T}[j]$ with an integer, starting from 0 to $|\mathcal{T}[j]| - 1$. Then, it represents each category as a *binary vector* with all zero values, except that the index of the integer corresponding to the category is set as one.

For example, for `gender`, ordinal and one-hot respectively transform its two values into 0 and 1, and $(1, 0)$ and $(0, 1)$.

Numerical attribute transformation. We normalize values in a numerical attribute to $[-1, 1]$, so as to enable neural networks in G to generate values in the attribute using `tanh` as an activation function.

1) Simple normalization. Considering numerical attribute $\mathcal{T}[j]$, we use $\mathcal{T}[j].\max$ and $\mathcal{T}[j].\min$ to respectively denote the maximum and minimum values of the attribute. Then, given an original value v in $\mathcal{T}[j]$, we normalize it as

$$v_{\texttt{norm}} = -1 + 2 \cdot \frac{v - \mathcal{T}[j].\min}{\mathcal{T}[j].\max - \mathcal{T}[j].\min}. \tag{3}$$

For example, considering attribute age in Figure 1, value 43 of the last record is transformed into 0.2.

2) GMM-based normalization. Some existing studies [42, 43] propose to consider the *multi-modal* distribution of a numerical attribute $\mathcal{T}[j]$, to avoid some limitations of simple normalization, such as gradient saturation. To this end, they utilize a Gaussian Mixture model (GMM) to cluster values of $\mathcal{T}[j]$, and normalize a value by the cluster it belongs to. They first train a GMM with s components over the values of $\mathcal{T}[j]$, where the mean and standard deviation of each component i are denoted by $\mu^{(i)}$ and $\sigma^{(i)}$. Then, given a specific value v, they compute the probability distribution $(\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(s)})$ where $\pi^{(i)}$ indicates the probability that v comes from component i, and normalize v as

$$v_{\texttt{gmm}} = \frac{v - \mu^{(k)}}{2\sigma^{(k)}}, where\ k = \arg\max_{i} \pi^{(i)}. \qquad (4)$$

For example, suppose that the records in our example table can be clustered into two modes, i.e., "young generation" and "old generation" with Gaussian distributions $G(20, 10)$ and $G(50, 5)$ respectively. Then, given an age value 43, we first determine that it is more likely to belong to the old generation, and then normalize it into a vector $(1, -0.7)$ where 0 indicates the first mode and -0.7 is $v_{\texttt{gmm}}$.

Combination of multiple attributes. Once all attributes in t are transformed by the above schemes, we need to combine them together to generate the output sample $\boldsymbol{t}$. As $\boldsymbol{t}$ is used to feed the subsequent neural networks, its form depends on the input requirement of the neural networks.

1) Matrix-formed samples. For CNN-based neural networks, we follow the method in [31] to convert the attributes into a square matrix. For example, a record with 8 attributes is converted into a 3×3 square matrix after padding one zero. Note that such a method requires each attribute is transformed into one value instead of a vector (otherwise, the vector of an attribute may be split in the converted matrix). Thus, one-hot encoding and GMM-based normalization are not applicable for matrix-formed samples.

2) Vector-formed samples. For MLP-based and LSTM-based neural networks, we concatenate all the attribute vectors to generate a sample vector, i.e., $\boldsymbol{t} = \boldsymbol{t}_1 \oplus \boldsymbol{t}_2 \oplus \ldots \oplus \boldsymbol{t}_m$. Obviously, this method is compatible to all the attribute transform schemes described above.

EXAMPLE 3 (DATA TRANSFORMATION). *Let us consider the last record of our example table as shown in Figure 1. Suppose that we want to transform the record into a matrix-formed sample. In this case, only ordinal encoding and simple normalization can be used for attribute transformation. Thus, we obtain a square matrix* $((0.2, 1, 2), (4, 1, 0), (0, 0, 0))$. *On the other hand, suppose that we want to transform the record into a vector-formed sample and consider one-hot encoding and GMM-based normalization. The record is transformed into* $(\underline{1, -0.7}, \underline{0, 1}, \underline{0, 0, 1, 0}, \underline{0, 0, 0, 0, 1}, \underline{0, 1})$*, where the underlines indicate different attributes.*

Note that the above transformation process is reversible: after generating synthetic sample $\boldsymbol{t}'$ using G, we can apply these methods to reversely convert $\boldsymbol{t}'$ to a fake record.

5. GAN MODEL DESIGN

In this section, we present technical details of GAN design, including neural network architectures (Section 5.1), training algorithms (Section 5.2), conditional GAN (Section 5.3) and differential privacy preserving GAN (Section 5.4).

5.1 Neural Network Architectures

This section presents neural network (NN) architectures for realizing G and D in GAN.

5.1.1 CNN: convolutional neural networks.

CNN is utilized in the existing works for relational data synthesis [7, 31], which is inspired by the well-known DCGAN [35]. Generator G takes as input a prior noise $\boldsymbol{z}$, which is denoted by $\boldsymbol{h}_g^0$, and uses multiple de-convolutional layers $\boldsymbol{h}_g^l$ (i.e., fractionally strided convolution) to transform $\boldsymbol{z}$ to a synthetic sample in the form of matrix, i.e.,

$$\begin{aligned} \boldsymbol{h}_g^{l+1} &= \mathtt{ReLU}(\mathtt{BN}(\mathtt{DeConv}(\boldsymbol{h}_g^l))), \qquad (5) \\ \boldsymbol{t} &= \mathtt{tanh}(\mathtt{DeConv}(\boldsymbol{h}_g^L)), \end{aligned}$$

On the other hand, discriminator D takes as input a real/fake sample $\boldsymbol{t}$ in matrix form, which is denoted by $\boldsymbol{h}_d^0$. It applies multiple convolutional layers $\boldsymbol{h}_d^l$ to convert $\boldsymbol{t}$ to a probability indicating how likely $\boldsymbol{t}$ is real, i.e.,

$$\begin{aligned} \boldsymbol{h}_d^{l+1} &= \mathtt{LeakyReLU}(\mathtt{BN}(\mathtt{Conv}(\boldsymbol{h}_d^l))), \\ f &= \mathtt{sigmoid}(\mathtt{BN}(\mathtt{Conv}(\boldsymbol{h}_d^L))), \qquad (6) \end{aligned}$$

where Conv is a convolution function.

5.1.2 MLP: fully connected neural networks.

MLP is used in the existing works for relational data synthesis [9, 42]. Specifically, G takes as input a prior noise $\boldsymbol{z}$, which is also denoted by $\boldsymbol{h}^{(0)}$, and utilizes with L fully-connected layers, where each layer is computed by

$$\boldsymbol{h}^{l+1} = \phi\big(\mathtt{BN}(\mathtt{FC}_{|\boldsymbol{h}^l| \rightarrow |\boldsymbol{h}^{l+1}|}(\boldsymbol{h}^l))\big), \qquad (7)$$

where $\mathtt{FC}_{|\boldsymbol{h}^l| \rightarrow |\boldsymbol{h}^{l+1}|}(\boldsymbol{h}^l) = \boldsymbol{W}^l \boldsymbol{h}^l + \boldsymbol{b}^l$ with weights $\boldsymbol{W}^l$ and bias $\boldsymbol{W}^l$, ϕ is the activation function (we use ReLU in our experiments), and BN is the batch normalization [18].

The challenge here is how to make the output layer in G *attribute-aware.* We propose to generate each attribute vector $\boldsymbol{t}_j$ depending on the transformation method on the corresponding attribute $\mathcal{T}[j]$, i.e.,

$$\boldsymbol{t}_j = \begin{cases} \mathtt{tanh}(\mathtt{FC}_{|\boldsymbol{h}^L| \rightarrow 1}(\boldsymbol{h}^L)), & (C_1) \\ \mathtt{tanh}(\mathtt{FC}_{|\boldsymbol{h}^L| \rightarrow 1}(\boldsymbol{h}^L) \oplus \mathtt{softmax}(\mathtt{FC}_{|\boldsymbol{h}^L| \rightarrow |\boldsymbol{t}_j|-1}(\boldsymbol{h}^L), & (C_2) \\ \mathtt{softmax}(\mathtt{FC}_{|\boldsymbol{h}^L| \rightarrow |\boldsymbol{t}_j|}(\boldsymbol{h}^L)), & (C_3) \\ \mathtt{sigmoid}(\mathtt{FC}_{|\boldsymbol{h}^L| \rightarrow 1}(\boldsymbol{h}^L)), & (C_4) \end{cases}$$

where C_1 to C_4 respectively denote the cases of using simple normalization, mode-specific normalization, one-hot encoding and ordinal encoding as transformation on the attribute $\mathcal{T}[j]$ (see Section 4). For example, consider C_2, the mode-specific normalization, we first use $\mathtt{tanh}(\mathtt{FC}_{|\boldsymbol{h}^L| \rightarrow 1}(\boldsymbol{h}^L)$ to generate $v_{\texttt{gmm}}$ and then use $\mathtt{softmax}(\mathtt{FC}_{|\boldsymbol{h}^L| \rightarrow |\boldsymbol{t}_j|-1}(\boldsymbol{h}^L)$ to generate a one-hot vector indicating which component $v_{\texttt{gmm}}$ belongs to. After generating $\{\boldsymbol{t}_j\}$ for all attributes, we concatenate them to obtain $\boldsymbol{t}$ as a synthetic sample.

Discriminator D is an MLP that takes a sample $\boldsymbol{t}$ as input, and utilizes multiple fully-connected layers and a sigmoid output layer to classify whether $\boldsymbol{t}$ is real or fake.

5.1.3 LSTM: recurrent neural networks.

Existing work also utilizes LSTM, a representative variant of RNN, to realize G [43]. The basic idea is to formalize record synthesis as a *sequence generation* process: it models a record $\boldsymbol{t}$ as a sequence and each element of the sequence is an attribute $\boldsymbol{t}_j$. It uses LSTM to generate $\boldsymbol{t}$ at multiple timesteps, where the j-th timestep is used to generate $\boldsymbol{t}_j$. Let $\boldsymbol{h}^j$ and $\boldsymbol{f}^j$ respectively denote the hidden state and output of the LSTM at the j-th timestep. Then, we have

$$\begin{aligned}\boldsymbol{h}^{j+1} &= \texttt{LSTMCell}(\boldsymbol{z}, \boldsymbol{f}^j, \boldsymbol{h}^j),\\ \boldsymbol{f}^{j+1} &= \texttt{tanh}(\texttt{FC}_{|\boldsymbol{h}^{j+1})|\rightarrow|\boldsymbol{f}^{j+1})|}(\boldsymbol{h}^{j+1})),\end{aligned}$$

where $\boldsymbol{h}^0$ and $\boldsymbol{f}^0$ are initialized with random values.

Next, we compute attribute $\boldsymbol{t}_j$ from $\boldsymbol{f}^{j+1}$ by considering transformation method of the corresponding attribute. Specifically, for simple normalization, one-hot encoding and ordinal encoding, we compute $\boldsymbol{t}_j$ as follows.

$$\boldsymbol{t}_j = \begin{cases} \texttt{tanh}(\texttt{FC}_{|\boldsymbol{f}^j|\rightarrow 1}(\boldsymbol{f}^j)), & \text{simple normalization}\\ \texttt{softmax}(\texttt{FC}_{|\boldsymbol{f}^j|\rightarrow|\boldsymbol{t}_j|}(\boldsymbol{f}^j)), & \text{one} - \text{hot encoding}\\ \texttt{sigmoid}(\texttt{FC}_{|\boldsymbol{f}^j|\rightarrow 1}(\boldsymbol{f}^j)), & \text{ordinal encoding}\end{cases}$$

In the case that attribute $t[j]$ is transformed by GMM-based normalization, we use two timesteps to generate its sample $\boldsymbol{t}_j$: the first timestep j_1 generates the normalized value $v_{\texttt{gmm}} = \texttt{tanh}(\texttt{FC}_{|\boldsymbol{f}^{j_1}|\rightarrow 1}(\boldsymbol{f}^{j_1}))$, while the second timestep j_2 generates a vector that indicates which GMM component $v_{\texttt{gmm}}$ comes from, i.e., $\texttt{softmax}(\texttt{FC}_{|\boldsymbol{f}^{j_2}|\rightarrow|\boldsymbol{t}_j-1|}(\boldsymbol{f}^{j_2}))$. Then, we concatenate these two parts to compose $\boldsymbol{t}_j$.

Note that we use a typical *sequence-to-one* LSTM [39] to realize D. We omit the details due to the space limit.

5.2 GAN Training Algorithms

We apply the vanilla GAN training algorithm [15] to iteratively optimize parameters θ_d in D and θ_g in G as follows. Due to the space limit, we present the pseudo-codes of all training algorithms in our technical report [**?**]. < (add) >

$$\theta_d \leftarrow \theta_d + \alpha_d \nabla_{\theta_d} \frac{1}{m}\sum_{i=1}^{m}[\log D(\boldsymbol{t}^{(i)}) + \log(1 - D(G(\boldsymbol{z}^{(i)})))]$$

$$\theta_g \leftarrow \theta_g^{(t-1)} - \alpha_g \nabla_{\theta_g} \frac{1}{m}\sum_{i=1}^{m}\log(1 - D(G(\boldsymbol{z}^{(i)}))),$$

where m is the size of the minibatch and α_d (α_g) is the learning rate of discriminator D (generator G). In each iteration, the algorithm trains discriminator D and generator D alternately. First, it fixes G and trains D by sampling m noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p(\boldsymbol{z})$ and m real examples $\{\boldsymbol{t}^{(i)}\}_{i=1}^m \sim p_{data}(\boldsymbol{t})$ and updating θ_d with the Adam optimizer [22]. Second, it fixes D and trains G by sampling another set of noise samples and updating parameters θ_g.

One limitation of the algorithm is that it may not provide sufficient gradient to train G well in the early training iterations [15]. Thus, some existing work [43] introduces to consider the KL divergence between real and synthetic data to warm up model training of G. Formally, let $\texttt{KL}(\mathcal{T}[j], \mathcal{T}'[j])$ denote the KL divergence regarding attribute $\mathcal{T}[j]$ between the sampled real examples $\{\boldsymbol{t}^{(i)}\}_{i=1}^m$ and synthetic samples $\{G(\boldsymbol{z}^{(i)})\}_{i=1}^m$. We optimize G by considering the original loss and KL divergences regarding all attributes, i.e.,

$$\mathcal{L}_G = \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] + \sum_{j=1}^{|\mathcal{T}|}\texttt{KL}(\mathcal{T}[j], \mathcal{T}'[j]), \quad (8)$$

We also evaluate Wasserstein GAN [2] for training our data synthesizer. Different from the original GAN, Wasserstein GAN removes the `sigmoid` function of D and changes the gradient optimizer from Adam to RMSProp. It uses the loss functions of D and G as

$$\begin{aligned}\mathcal{L}_D &= -\mathbb{E}_{\boldsymbol{t}\sim p_{data}(\boldsymbol{t})}[D(\boldsymbol{t})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))]\\ L(G) &= -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))].\end{aligned}$$

It uses T_g iterations to optimize G. In each G's training iteration, it first uses T_d iterations to train D, and then trains G. In particular, it clips the parameters θ_d of D into an interval $[-c_p, c_p]$ after each training iteration of D.

5.3 Conditional GAN

The imbalanced label distribution in real-world data may result in *insufficient training* for records with minority labels [42]. Thus, some studies [42] apply conditional GAN [30] to data synthesis. The basic idea is to encode label as a *condition vector* $\boldsymbol{c} \in \mathbb{R}^c$ and feed $\boldsymbol{c}$ to both generator and discriminator as an additional input. We respectively represent the generator and the discriminator as $G(\boldsymbol{z}|\boldsymbol{c};\theta_g) \in \mathbb{R}^d$ and $D(\boldsymbol{t}|\boldsymbol{c};\theta_d)$. Then, generator G would like to generate samples conditioned on $\boldsymbol{c}$ which can perfectly fool discriminator D, while D wants to distinguish real samples with condition $\boldsymbol{c}$ from synthetic ones. Formally, the objective of the minimax game can be represented as follows [30].

$$\begin{aligned}\min_G \max_D V(G, D) =& \mathbb{E}_{\boldsymbol{t}\in p_{data}(\boldsymbol{t})}\big[\log D(\boldsymbol{t}|\boldsymbol{c})\big]\\ &+ \mathbb{E}_{\mathbf{z}\in p_z(\mathbf{z})}\big[1 - \log D(G(\mathbf{z}|\boldsymbol{c}))\big]. \quad (9)\end{aligned}$$

A naïve way to train conditional GAN is to simply modify the vanilla training algorithm by extracting label y of each sampled real example $\boldsymbol{t}^{(i)}$ and feeding y to both D and G. However, due to the highly imbalanced label distribution, the minority label may not have sufficient training opportunities. To overcome the obstacle, we introduce *label-aware* data sampling into model training. The idea is to sample minibatches of real examples by considering labels as a condition, instead of uniformly sampling data. Specifically, in each iteration, the algorithm considers every label in the real data, and for each label, it samples records with corresponding label for the following training of D and G. Using this method, we can ensure that data with minority labels also have sufficient training opportunities.

5.4 Differential Privacy Preserving GAN

We apply DPGAN [41] to enable our data synthesizer to have theoretical guarantees on differential privacy. The basic idea is to add noise to the gradients used to update parameters θ_d to make discriminator D differentially private, since D accesses the real data and has the risk of disclosing privacy information. Then, according to the *post-processing* property of differential privacy, a differentially private D will also enables G differentially private, as parameters θ_g are updated based on the output of D. Overall, DPGAN follows the framework of Wasserstein GAN training, with a minor modifications. When training D, for each sampled noise $\boldsymbol{z}^{(i)}$ and real example $\boldsymbol{t}^{(i)}$, it adds Gaussian noise $N(0, \sigma_n^2 c_g^2 I)$ to the gradient $\nabla_{\theta_d}[D(\boldsymbol{t}^{(i)}) - D(G(\boldsymbol{z}^{(i)}))]$, where σ_n is the noise scale and c_g is a user-defined bound on the gradient of Wasserstein distance with respect to parameters θ_d (see the original paper [41] for details of c_g).

6. EVALUATION METHODOLOGY

Table 1: Dataset statistics: #Size is the number of records, #Cat and #Num are the number of numerical and categorical attributes, and #Labels is the number of unique labels for classification.

Dataset	#Size	#Num	#Cat	#Labels
Adult	41292	6	8	2
Covertype	116204	10	2	7
Sdata1	120000	2	0	2
Sdata2	60000	0	8	2

6.1 Datasets

For our evaluation, we select two real datasets `Adult` and `Covertype` used in the existing works for GAN-based data synthesis [42, 43, 31]. These two datasets are representative: (1) They have different ratios between the numbers of numerical and categorical attributes; (2) They are used to train different classifiers, i.e., binary and multi-class.

(1) `Adult` dataset. This dataset contains the personal information of $41,292$ individuals extracted from the 1994 US census with 8 categorical attributes, such as `Workclass` and `Education` and 6 numerical attributes, such as `Age` and `Hours-per-Week`. We use this dataset to train a binary classifier: we use attribute `Income` as label and predict whether a person has income larger than $50K$ per year (positive) or not (negative). The label distribution is skew, i.e., the ratio between positive and negative labels is 0.34. This dataset is downloaded from the UCI Machine Learning Repository[4].

(2) `Covertype` dataset. This dataset contains the information of $116,204$ forest records obtained from US Geological Survey (USGS) and US Forest Service (USFS) data. It includes 2 categorical attributes, `Wild-area` and `Soil-type`, and 10 numerical attributes, such as `Elavation` and `Slope`. We consider a multi-class classification problem on the dataset: we use attribute `Cover-type` with 7 distinct values as label and predicts forest cover-type from other cartographic variables. The label distribution is also very skew, e.g., there are 46% records with label 2 while only 6% records with label 3. This dataset is also downloaded from the UCI Machine Learning Repository.

The real datasets have *mixed* data types. To provide in-depth analysis on synthesis performance on different attribute types, we also use two synthetic datasets generated by simulation, each of which only has one data type.

(3) `SDataNum` dataset. This dataset is used to evaluate the generation of records with purely *numerical* attributes. To this end, we use 25 two-dimension Gaussian Distribution $N(u_i, v_i), (u_i, v_i) \in \{(u, v)|u, v \in \{-4, -2, 0, 2, 4\}\}$ to simulate two numerical attributes, where each record in dataset is randomly sampled from the 25 Gaussian Distribution components, then we assign labels according to the components. We consider a binary classification problem that predicts the label from the numerical attributes.

(4) `SDataCat` dataset. This dataset is used to evaluate the generation of records with purely *categorical* attributes. We generate 9 categorical attributes as follows. We use a chain Bayesian network, and the number of value for each attribute (except the label column) and the conditional probability corresponding to each value are generated randomly. We consider a binary classification problem that predicts a label from these simulated categorical attributes.

The statistical information of the datasets is summarized in Table 1. Detailed attribute information in these datasets is described in our technical report [**?**]. < (**add**) >

6.2 Evaluation Framework

We implement our GAN-based relational data synthesis framework, as shown in Figure 2, using PyTorch [34]

To evaluate the performance of the data synthesis framework, we split a dataset into training set $\mathcal{T}_{\texttt{train}}$, validation set $\mathcal{T}_{\texttt{valid}}$ and test set $\mathcal{T}_{\texttt{test}}$ with ratio of 4:1:1 respectively. Next, we train a data synthesizer realized by our GAN-based framework on the training set $\mathcal{T}_{\texttt{train}}$ to obtain the optimized parameters for discriminator and generator. Specifically, we first perform *hyper-parameter search*, which will be described later, to determine the hyper-parameters of the model. Then, we run a training algorithm (see Section 5.2) for parameter optimization. We divide the training iterations in the algorithm evenly into 10 *epochs* and evaluate the model performance after each epoch on the validation set $\mathcal{T}_{\texttt{valid}}$. We select the model snapshot with the best performance and generate a synthetic relational table $\mathcal{T}'$.

After obtaining $\mathcal{T}'$, we compare it with the original table $\mathcal{T}_{\texttt{train}}$ on both ML training utility and privacy protection.

Evaluation on ML training utility. We train a classifier f' on the fake table $\mathcal{T}'$, while also training another classifier f on the training set $\mathcal{T}_{\texttt{train}}$. In our experiments, we consider the following four types of classifiers for evaluation. (1) Decision Tree (DT): A simple classifier for classification by training a decision tree model. We adopt 2 decision trees with max depth 10 and 30 respectively. (2) Random Forest (RF): A classifier that uses multiple decision trees to train and predict. We adopt two random forests with max depth 10 and 20 respectively. (3) AdaBoost: It uses an iterative algorithm to train different classifiers(weak classifiers), then gathers them to form a stronger final classifier for classification. (4) Logical Regression (LR): A generalized linear regression model which uses gradient descent method to optimize the classifier for classification.

We evaluate the performance of a trained classifier f' on the test set $\mathcal{T}_{\texttt{test}}$. As label distribution of a dataset may be highly imbalanced, accuracy is inadequate for performance evaluation. Thus, we use the F1 score, which is the harmonic average of precision and recall, as the evaluation metric for the classifier. In particular, for binary classifier, we measure the F1 score of the positive label, which is much fewer but more important than the negative label. For the multi-class classifier on the `Covertype` dataset, we measure the F1 score of label 3, which is a rare label among all the seven labels. Meanwhile, we also measure the F1 score of classifier f which is trained on the real data $\mathcal{T}_{\texttt{train}}$. We evaluate the performance of a data synthesizer by measuring the difference `Diff` of the F1 scores between f' and f, as shown in Equation (1). The smaller the difference is, the better $\mathcal{T}'$ is for ML training. Note that we also consider area under the ROC curve (AUC) as evaluation of classifiers, and obtain similar trends with that of F1 score.

Evaluation on Privacy Protection. We adopt the following two metrics, which are widely used in the existing works [32, 26, 28] for privacy evaluation.

1) *Hitting Rate*: It measures how many records in the original table $\mathcal{T}_{\texttt{train}}$ can be hit by a synthetic record in $\mathcal{T}'$. To measure hitting rate, we first randomly sample 5000 synthetic records from $\mathcal{T}'$. For each sampled record, we measure

[4] http://archive.ics.uci.edu/ml/index.php

Table 2: Evaluating neural networks of generator G on synthetic data utility (`Adult` dataset).

Classifier	CNN	MLP				LSTM			
		`nrm +ord`	`nrm +hot`	`gmm +ord`	`gmm +hot`	`nrm +ord`	`nrm +hot`	`gmm +ord`	`gmm +hot`
DT10	0.475	0.062	0.062	0.056	0.040	0.069	0.113	0.088	**0.032**
DT30	0.485	0.071	**0.049**	0.077	0.094	0.059	0.167	0.088	0.062
RF10	0.417	0.035	0.038	0.029	0.018	0.136	0.050	0.054	**0.015**
RF20	0.458	0.060	0.066	0.053	0.034	0.125	0.047	0.051	**0.006**
AdaBoost	0.217	0.066	0.059	0.029	0.042	0.219	0.025	0.064	**0.009**
LR	0.047	0.018	0.088	0.018	0.013	0.012	0.009	**0.006**	0.012

Table 3: Evaluating neural networks of generator G on synthetic data utility (`Covertype` dataset).

Classifier	MLP				LSTM			
	`nrm +ord`	`nrm +hot`	`gmm +ord`	`gmm +hot`	`nrm +ord`	`nrm +hot`	`gmm +ord`	`gmm +hot`
DT10	0.190	0.170	0.566	0.241	0.130	0.107	0.402	**0.079**
DT30	0.534	0.327	0.752	0.437	0.419	0.606	0.652	**0.305**
RF10	0.165	0.123	0.455	0.155	**0.111**	0.198	0.259	0.113
RF20	0.342	0.253	0.648	0.264	0.247	0.312	0.491	**0.197**
AdaBoost	0.091	0.070	0.321	**0.029**	0.056	0.036	0.098	0.038
LR	0.130	0.058	0.516	0.113	0.076	0.369	0.378	**0.043**

the proportion of records in $\mathcal{T}_{\texttt{train}}$ that are *similar* to this synthetic record. We regard two records are similar if and only if 1) the values of each categorical attribute are the same, and 2) the difference between values of each numerical attribute is within a threshold. In our experiment, this threshold is set as the range of the attribute divided by 30.

2) *Distance to the closest record (DCR)*: This metric measures whether the synthetic data is weak from re-identification attacks [32, 26]. Given a record t in the original table $\mathcal{T}_{\texttt{train}}$, we find the synthetic record from $\mathcal{T}'$ that is closest to t in Euclidean distance. Note that a record with DCR=0 means that $\mathcal{T}'$ leaks its real information, and the larger the DCR is, the better the privacy protection is. To measure DCR, we calculate the distance after attribute-wise normalization to make sure that each attribute contributes the distance equally. We sample 3000 records from the original table $\mathcal{T}_{\texttt{train}}$, and find the the nearest synthetic record in $\mathcal{T}'$ for each of these records. Then, we compute the the average distance between the real record to its closest synthetic record.

6.3 Data Synthesis Methods

GAN-based methods. We implement the design choices for each component in our framework (see Figure 3). We use the source code provided by the authors of [31] to implement the CNN-based model[5]. We use the hyper-parameter settings provided by the code to train the model. Moreover, the code provides three privacy settings. When evaluating the ML training utility, we choose the settings of the weakest privacy protection to achieve the best synthetic data utility. On the other hand, We implement the MLP-based and LSTM-based models by ourselves using PyTorch to enable the flexibility of adapting different transformation schemes for comprehensive evaluation. Also, we implement the variants of training algorithms, conditional GAN and DPGAN, which are described in Section 5.

Statistical methods. We compare our GAN-based framework with a state-of-the-art statistical data synthesis method `PrivBayes` [46, 47], using the source code downloaded here[6]. As `PrivBayes` has theoretical guarantee on differential privacy [13], we vary the privacy parameter ϵ to examine the tradeoff between privacy protection and data utility. According to the original papers [46, 47], we run `PrivBayes` in multiple times and report the average result.

Variational Autoencoder (VAE). We also implement variational autoencoder (VAE), which is another representative deep generative model [23, 37] for relational data synthesis. To train a VAE, we adopt the loss function that consists of both the reconstruction loss and the KL divergence [10]. In our work, we use the binary cross-entropy (BCE) loss for categorical attributes and the mean squared error (MSE) loss for numerical attributes.

6.4 Hyper Parameter Search

Hyper parameter search is very important for neural networks. We adopt the method in a recent empirical study for GAN models [27] for hyper parameter search. Given a GAN model, we firstly generate a set of candidate hyper parameter groups, each of which includes the hyper-parameters for model training. Then, we train the model for several times, and at each time, we randomly select a hyper-parameter group and evaluate the trained model on the validation set $\mathcal{T}_{\texttt{valid}}$. Based on this, we select the hyper-parameter group that results in a model with the best performance.

All the experiments are conducted on a server with 2TB disk, 40 CPU cores (Intel Xeon CPU E5-2630 v4 @ 2.20GHz), one GPU (NVIDIA TITAN V) and 512GB memory, and the version of Python is 3.6.5.

7. EVALUATION RESULTS

7.1 Evaluating GAN-based Framework

This section explores the design space of our GAN-based framework shown in Figure 3. We focus on the results on synthetic data utility due to the space limit, and will report privacy results in the next section.

7.1.1 Evaluation on Neural Networks

[5] https://github.com/mahmoodm2/tableGAN

[6] https://sourceforge.net/projects/privbayes/

Table 4: Evaluating neural networks of generator G on synthetic data utility (SDataNum dataset).

Classifier	CNN	MLP		LSTM	
		nrm	gmm	nrm	gmm
DT10	0.444	0.190	0.166	0.146	**0.052**
DT30	0.296	0.146	0.150	0.128	**0.069**
RF10	0.545	0.178	0.143	0.140	**0.080**
RF20	0.566	0.172	0.147	0.132	**0.050**
AdaBoost	0.544	0.090	0.040	0.171	**0.004**
LR	0.182	0.007	0.010	**0.002**	0.005

Table 5: Evaluating neural networks of generator G on synthetic data utility (SDataCat dataset).

Classifier	CNN	MLP		LSTM	
		ord	hot	ord	hot
DT10	0.177	0.073	0.032	**0.013**	**0.013**
DT30	0.219	0.074	0.033	**0.013**	**0.013**
RF10	0.178	0.073	0.024	**0.004**	0.014
RF20	0.181	0.067	0.026	**0.006**	0.007
AdaBoost	0.122	0.076	0.008	**0.007**	**0.007**
LR	0.115	0.070	0.010	**0.001**	0.010

We evaluate the neural networks, CNN, MLP and LSTM that realize the generator G in our framework. Notice that, for MLP and LSTM, we fix the discriminator D as MLP (see Figure **??**). We also evaluate the LSTM-based discriminator and obtain inferior result (the result is included in our technical report [**?**] due to the space limit). Tables 2, 3, 4 and 5 report the experimental results on the four datasets, where nrm, gmm, ord, and hot respectively denote simple normalization, GMM-based normalization, ordinal encoding and one-hot encoding. Note that the CNN model is not evaluated on the Covertype dataset, as the original code in [31] is not designed for multi-class classification.

Observed from the results, LSTM achieves the best performance in most of the cases, and outperforms CNN and MLP. This suggests the *sequence generation* mechanism in LSTM, which generates a record attribute by attribute, is more adequate for relational data synthesis. Firstly, as illustrated in Figure **??**, each attribute is generated from a separated noise $\boldsymbol{z}$, which avoids the disturbance among different attributes. Secondly, LSTM does not generate an attribute from scratch. Instead, it generates an attribute based on the "understanding" of previous attributes, i.e., the hidden state $\boldsymbol{h}$ and previous output $\boldsymbol{f}$, and thus it can capture the column dependency. For instance, consider our example table in Figure 1. Suppose that we have partly generated a record with 11th as its education. LSTM can learn to lower the likelihood of generating the occupation values that needs high education background for the record.

CNN achieves inferior performance in relational data synthesis, which is quite different from the scenario of image synthesis [35]. This is, on the one hand, because the matrix input of CNN is only compatible with simple normalization and ordinal encoding, as discussed in Section 4, which is not effective for representing numerical and categorical data. Moreover, the convolution/deconvolution operation in CNN is usually effective for data with *feature locality.* For example, features, which are locally close to each other in the matrix of an image, may also be semantically correlated, e.g., composing an object in the image. However, relational data does not possess such feature locality.

Table 6: An extreme example of mode collapse.

age	workclass	fnlwgt	education	...	income
51	Private	167317	Prof-school	...	$\leqslant 50K$
51	Private	167317	Prof-school	...	$\leqslant 50K$
21	Private	167317	Prof-school	...	$\leqslant 50K$
32	Private	167317	Prof-school	...	$\leqslant 50K$

Finding 1: LSTM with appropriate transformation schemes generates the best synthetic data on ML training utility, achieving $7\% - 90\%$ less F1 difference than that of the second best model, MLP.

For ease of presentation, in the remainders of this section, we use LSTM with one-hot encoding and GMM-based normalization as its default setting.

7.1.2 Evaluation on GAN Training

We first evaluate the *robustness* of MLP-based and LSTM-based generator wrt. hyper parameter settings. Recall that we generate a set of candidate hyper parameter settings and randomly select one setting when training a model (Section 6.4). Given a setting, we divide the training iterations evenly into 10 *epochs* and generate a snapshot of synthetic table after each epoch. Then, we evaluate the F1 score of a classifier trained on each synthetic table snapshot.

Figure 4 shows the experimental results on the two real datasets. We have a surprising observation that the LSTM-based generator performs badly in some hyper parameter settings. For example, on the Adult dataset, the F1 score drops sharply to 0 after the few early epochs in 4 out of 6 hyper parameter settings. To investigate the reason, we sample some records from the inferior synthetic table snapshot and find the result as illustrated in Table 6: generator G only produces duplicated samples, rather than outputting diverse synthetic records. This is a commonly encountered failure in GAN training, called *mode collapse* [38, 29]. The reason is that generator G may not get sufficient gradients in the early training iterations and thus overfits to a few training records. On the other hand, MLP-based generator is robust against various hyper parameter settings. Moreover, it achieves moderate results on F1 score, although its best case is worse than that of LSTM-base generator.

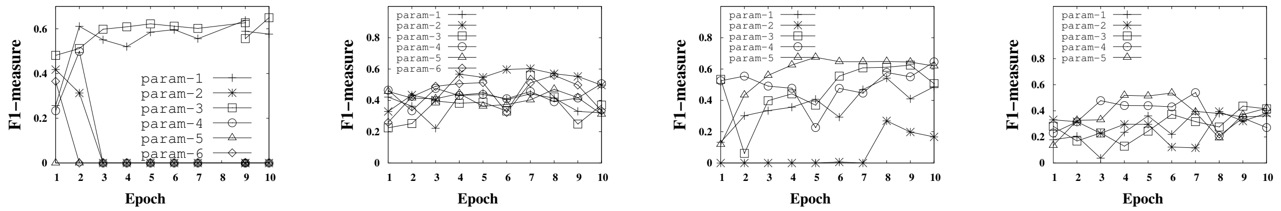
Finding 2: MLP is more robust against hyper parameter settings and achieves moderate results, while LSTM is more likely to result in mode collapse if its hyper parameters are not well tuned.

Moreover, we compare VanillaTrain (with KL divergence) with Wasserstein GAN training (Algorithm 2). As observed from Figure 5, Wasserstein GAN training does not have advantage over vanilla GAN training, which is different from the image synthesis scenarios. Thus, we use the latter to train GAN models throughout our experiments.

Finding 3: The Vanilla GAN training algorithm performs better than Wasserstein GAN training in the scenario of relational data synthesis.

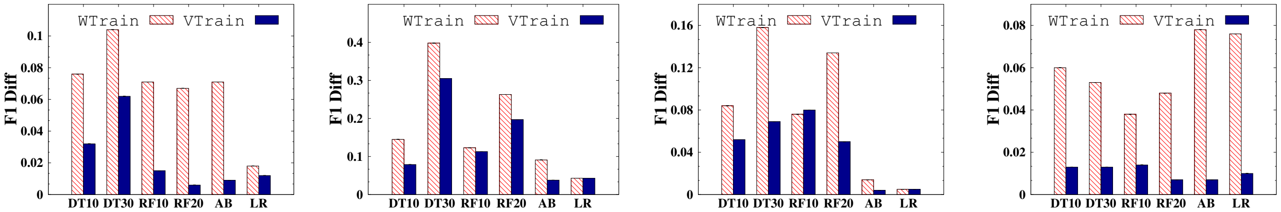
7.1.3 Evaluation on Conditional GAN

This section investigate whether conditional GAN is helpful to address the challenge of imbalance label distribution.



(a) LSTM-based G (`Adult`). (b) MLP-based G (`Adult`). (c) LSTM-based G (`Covertype`). (d) MLP-based G (`Covertype`).

Figure 4: Evaluating GAN model training on various hyper-parameter settings.



(a) `Adult` dataset. (b) `Covertype` dataset. (c) `SDataNum` dataset. (d) `SDataCat` dataset.

Figure 5: Comparison between Wasserstein GAN training and Vanilla GAN training

We compare the original GAN, conditional GAN trained by uniform data sampling and condition GAN trained by label-aware data sampling, which are denoted by `VGAN`, `CGAN-V` and `CGAN-C` respectively. The result is shown in Figure 6. We can see that `CGAN-V` gains very limited improvements over `GAN`, and sometimes it performs worse than `GAN` as shown in Figure 6(a). This is because that `VTrain` uses the random strategy to sample each minibatch of real records. Due to the high label imbalance, records with minority labels (e.g., positive label on the `Adult` dataset) may have less chances to be sampled, leading to insufficient training opportunities for the minority labels. On the contrary, `CGAN-C` solves this problem by sampling records conditioned on given labels. This label-aware sampling method can provide fair training opportunities for data with different labels.

Finding 4: Conditional GAN plus label-aware data sampling is helpful to address the challenge of imbalance label distribution and improves the ML training utility of synthetic data.

7.1.4 *Evaluation on Data Transformation*

We also find that data transformation in preprocessing does affect the overall utility of synthetic data. Observed from Tables 2, 3, 4 and 5, GMM-based normalization and one-hot encoding achieve the best performance in most of the cases. To provide in-depth analysis, we further examine whether the value distribution of a synthetic attribute is similar to that of its counterpart real attribute. We report the results on `SDataNum` and `SDataCat` to purely evaluate numerical and categorical attributes respectively. We also find similar observations on the two real datasets `Adult` and `Covertype`, and we include their results in [**?**].

Figure 7 shows the distribution for numerical attributes using the violin plots. LSTM with `gmm` can generate the attribute having the most approximate distribution to their counterpart real attribute, and it is remarkably effective for the attribute with *multi-modal* distribution. This is attributed to the Gaussian Mixture model used in this method, which is more powerful to represent multi-modal attributes. Moreover, it also outperforms MLP with `gmm`. This is because that LSTM uses two time steps to generate normalized value $v_{\texttt{gmm}}$ and components probabilities $\{\pi^{(i)}\}$ separately, which is shown more effective than generating them together in MLP. Figure 8 shows the distribution for categorical attributes. We can see that one-hot is significantly better than ordinal embedding. This is because values in a categorical attribute usually do not have ordinal relationships, and thus a single number is insufficient for attribute representation.

Finding 5: Data transformation does affect ML training utility of synthetic data: GMM-based normalization performs better than simple normalization, especially for numerical attributes with multi-modal distribution; One-hot encoding is better than ordinal encoding for categorical attributes.

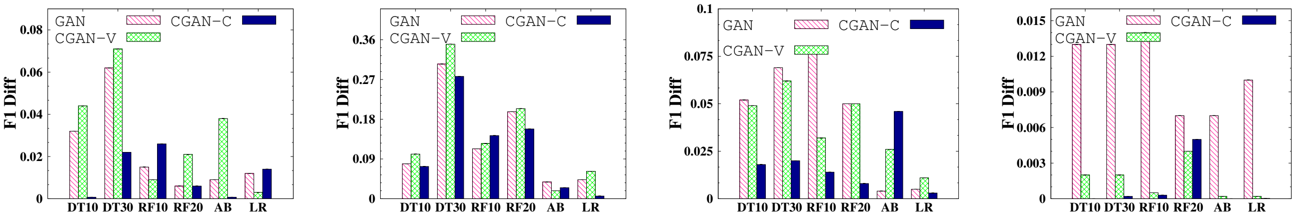
7.1.5 *Effect of Synthetic Data Size*

Recall that the synthetic table $\mathcal{T}'$ is generated by repeatedly calling generator $G(\mathbf{z};\theta_g)$ where each call produces one synthetic record. This section evaluates whether the size $|\mathcal{T}'|$, i.e., the number of calling G, would affect the ML training utility of $\mathcal{T}'$. Table 7 reports the F1 difference `Diff` when varying the ratio between sizes of synthetic $\mathcal{T}'$ and real $\mathcal{T}_{\texttt{train}}$ tables. We observe that, with the increase of synthetic table size, the performance of classifier DT is improved, as more samples can be used for training the classifier. However, the improvement is not very significant due to the fact that increasing synthetic data size does not actually inject more *information*: synthetic tables with varying sizes are from a generator G with the same set of parameters θ_g.

7.2 Comparing Data Synthesis Methods

This section compares GAN with `VAE` and `PrivBayes`, which are described in Section 6.3. Note that we use the conditional GAN, which achieves the best performance as reported previously, as the default setting of GAN.

7.2.1 *Evaluation on Synthetic Data Utility*



(a) `Adult` dataset. (b) `Covertype` dataset. (c) `SDataNum` dataset. (d) `SDataCat` dataset.

Figure 6: Evaluating conditional GAN on relational data synthesis.

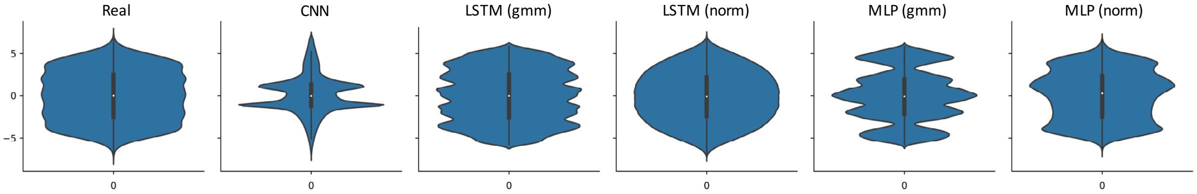


Figure 7: Evaluating value distribution of synthetic numerical attributes (`SDataNum`).

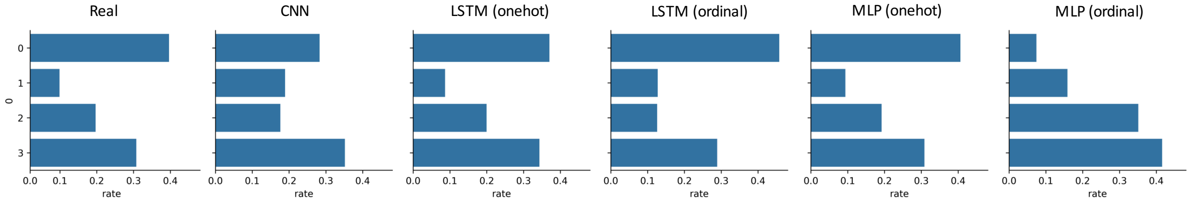
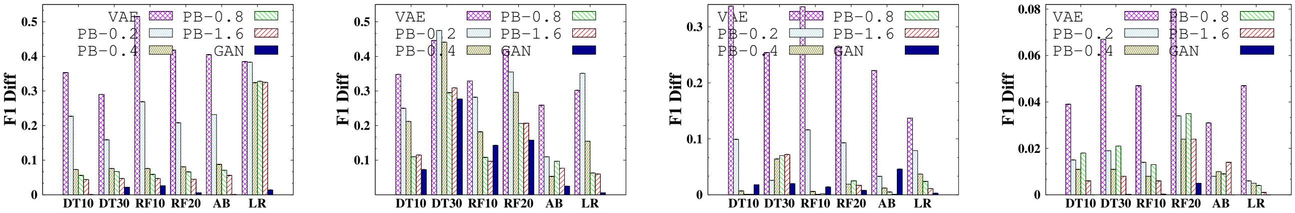


Figure 8: Evaluating value distribution of synthetic categorical attributes (`SDataCat`).



(a) `Adult` dataset. (b) `Covertype` dataset. (c) `SDataNum` dataset. (d) `SDataCat` dataset.

Figure 9: Comparison of different approaches for relational data synthesis.

Table 7: Effect of size ratio between synthetic and original tables (using DT10 as classifier).

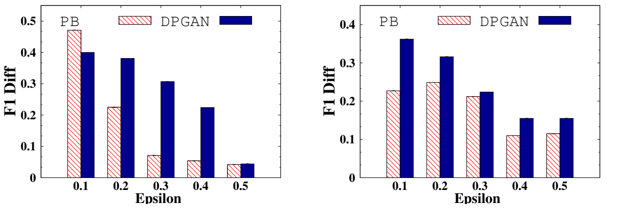
Dataset	Size ratio: $\lvert\mathcal{T}'\rvert/\lvert\mathcal{T}_{\mathtt{train}}\rvert$			
	50%	100%	150%	200%
`Adult`	0.073	0.032	0.028	**0.024**
`Covertype`	0.088	0.079	0.117	**0.064**
`SDataNum`	0.079	0.052	**0.024**	0.033
`SDataCat`	0.013	0.013	**0.007**	0.012

Table 8: Comparison between `GAN` and `PrivBayes` on privacy, i.e., Hitting Rate and DCR.

Method	Hitting Rate (%)		DCR	
	`Adult`	`Covertype`	`Adult`	`Covertype`
`PrivBayes-0.1`	0.49	0.002	0.164	0.106
`PrivBayes-0.2`	0.88	0.006	0.147	0.094
`PrivBayes-0.4`	2.16	0.022	0.123	0.082
`PrivBayes-0.8`	4.40	0.056	0.112	0.073
`PrivBayes-1.6`	4.64	0.070	0.110	0.069
`GAN`	0.30	0.500	0.113	0.072

Figure 9 shows the experimental results on synthetic data utility. First, `VAE` generates synthetic data with the most inferior utility for ML training. This result is similar to that of image synthesis [12]: the images synthesized by VAE is much worse than that generated by GAN. This is because the low dimensional latent variable may not be sufficient to capture the complex relational data. Second, with the increase of privacy parameter ϵ, the result of `PrivBayes` becomes better. This is because ϵ is used to control the privacy level: the larger the ϵ, the lower the privacy level.

Observed from the figures, our GAN-based framework significantly outperforms `PrivBayes` on preserving data utility for ML training. For example, the F1 difference achieved by `GAN` is $45-98\%$ and $10-90\%$ smaller than that achieved



(a) Adult dataset. (b) Covertype dataset.

Figure 10: Comparing DPGAN and PrivBayes on varying privacy levels (using DT10 as classifier).

by PrivBayes with the lowest privacy level ($\epsilon = 1.6$) on the Adult and Covertype datasets respectively. This is mainly attributed to their different data synthesis mechanisms. PrivBayes aims at approximating a joint multivariate distribution of the original table, which may not perform well if the data distribution is complex. On the contrary, GAN adopts a discriminator D and utilizes the adversarial training mechanism to optimize generator G. The experimental result shows that the adversarial mechanism is useful for synthesizing relational data.

Finding 6: GAN significantly outperforms VAE and PrivBayes on synthetic data utility. For some classifiers, the F1 difference of the synthetic data wrt. the original data achieved by GAN is smaller than that of VAE and PrivBayes by an order of magnitude.

7.2.2 Evaluation on Privacy

Table 8 compares GAN with PrivBayes on protecting privacy against the risk of re-identification, measured by Hitting Rate and DCR introduced in Section 6.2.

Firstly, we can see that, on the Adult dataset, GAN achieves much lower hitting rate than PrivBayes. For example, even compared with PrivBayes with the highest privacy level $\epsilon = 0.1$, GAN reduces the hitting rate by 39%. On the Covertype dataset, GAN achieves very low hitting rate 0.5%, i.e., only 25 out of 5000 sampled synthetic record can hit similar records in the original table. We notice that, on the Covertype dataset, the hitting rate of GAN is higher than that of PrivBayes. This is because most of the attributes on Covertype are *numerical* attributes and PrivBayes discretizes the domain of each numerical attribute into a fixed number of equi-width bins [46, 47], and thus a synthetic numerical value is seldom similar to the original one. Secondly, considering the metric DCR, GAN provides comparable overall performance to PrivBayes, and even outperforms PrivBayes with moderate privacy level ($\epsilon = 0.8$ or 1.6). This above results validate our claim that GAN can reduce the risk of re-identification as there is no *one-to-one* relationship between real and synthetic records.

Finding 7: Empirically, the GAN-based data synthesis framework shows better tradeoff between ML training utility and protecting privacy against the risk of re-identification, as there is no one-to-one relationship between original and synthetic records.

We also investigate whether the current solution DPGAN for GAN with differential privacy (DP) guarantee (see Section 5.4) is effective for relational data synthesis. Figure 10 reports the experimental results on varying privacy level ϵ. We can see that DPGAN cannot beat PrivBayes at almost all privacy levels on the Adult and Covertype datasets. This is because DPGAN adds noise to the gradients for updating parameters of D and then uses D to update parameters of G. This process may make the adversarial training ineffective, as D now has limited ability to differentiate real/fake samples. The experimental result also implies better solutions for DP preserving GAN need to be invented.

Finding 8: The current solution for differential privacy (DP) preserving GAN cannot beat traditional data synthesis methods with DP guarantees.

8. CONCLUSION & FUTURE DIRECTION

In this paper, we have conducted so far the most comprehensive experimental study for applying GAN to relational data synthesis. We introduced a unified framework and defined a design space of the solutions that realize GAN, including data transformation, neural network architecture, and training strategy. We empirically conducted a thorough evaluation to explore the design space and compare GAN with conventional approaches to data synthesis. Based on our experimental findings, we summarize the following key insights that provide guidance to the practitioners who want to apply GAN to develop a relational data synthesizer.

Overall Evaluation for GAN. GAN proves highly promising for relational data synthesis. It generates synthetic data with very good utility on ML model training and sometimes outperforms the baseline approaches by one order of magnitude (Finding 6). It also achieve competitive performance on protecting privacy against the risk of re-identification (Finding 7). However, GAN has limitations on providing provable privacy protection: the current solution cannot produce superior data utility when preserving differential privacy.

Neural Network Selection. For ordinary users with limited knowledge on deep learning, we suggest them to use MLP to realize GAN, as MLP is more robust and can achieve moderate or above results without parameter tuning (Finding 2). In contrast, for expert users who want to spend sufficient efforts to finetune parameters, we recommend LSTM that can achieve the best performance given proper parameters and data transformation schemes.

Model Training Strategy. We reveal that one should carefully train the GAN model according to the characteristics of their original datasets. Representative solutions to boost model training include adding KL divergence in the loss function for warm-up (Finding 3) and leveraging conditional GAN for datasets with highly imbalanced data distribution (Finding 4). We also suggest the users to pay much attention to the mode collapse phenomenon (Finding 2).

Relational Data Representation. Data transformation that converts original records to recognized input of GAN, such as matrix and vector, does affect the overall performance (Finding 4), which shows that the representation of relational data is also very important. This may imply an interesting future work that co-trains GAN and record representation through a hybrid optimization framework.

We also identify several future directions in GAN-based relational data synthesis that may be worthy of exploration.

(1) Providing provable privacy protection. We have shown that GAN has limitations on providing provable privacy protection, such as differential privacy. Although enabling GAN to support differential privacy is a hot research topic in ML [41, 19], this problem is very challenging, because adding noises to the adversarial training in GAN may drastically affect parameter optimization in G and D. Therefore, it calls for new solutions to equip GAN-based data synthesis with provable privacy protection.

(2) Capturing dependency among columns. Finding 1 reveals that LSTM achieves superior performance as its sequence generation mechanism can *implicitly* capture dependency among columns using the hidden state of LSTM. On the other hand, the DB community has long studied how to model column dependency *explicitly* by providing solutions like functional dependency (FD) [36] and conditional functional dependency (CFD) [5]. Despite some preliminary attempt [7], it still remains an unsolved question that how to combine techniques from the two communities to improve the synthetic data quality for the GAN-based framework.

(3) Supporting more utility definitions. This paper focuses on the specific need on training classification models in ML to define the utlity of synthetic data. However, relational data synthesis should be able to support a variety of applications, including more ML tasks, such as regression and clustering, OLAP queries, such as count and average, and so on. For example, some recent works discuss data synthesis for supporting approximate query processing (AQP) [40]. However, there is a highly demand to conduct more thorough research.

9. REFERENCES

[1] D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *PODS*, 2001.

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *CoRR*, abs/1701.07875, 2017.

[3] M. K. Baowaly, C. Lin, C. Liu, and K. Chen. Synthesizing electronic health records using improved generative adversarial networks. *JAMIA*, 26(3):228–241, 2019.

[4] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *PODS*, pages 273–282, 2007.

[5] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.

[6] J. Brickell and V. Shmatikov. The cost of privacy: destruction of data-mining utility in anonymized data publishing. In *SIGKDD*, pages 70–78, 2008.

[7] H. Chen, S. Jajodia, J. Liu, N. Park, V. Sokolov, and V. S. Subrahmanian. Faketables: Using gans to generate functional dependency preserving tables with bounded real data. In *IJCAI*, pages 2074–2080, 2019.

[8] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, pages 2172–2180, 2016.

[9] E. Choi, S. Biswal, B. A. Malin, J. Duke, W. F. Stewart, and J. Sun. Generating multi-label discrete electronic health records using generative adversarial networks. *CoRR*, abs/1703.06490, 2017.

[10] C. Doersch. Tutorial on variational autoencoders. *CoRR*, abs/1606.05908, 2016.

[11] J. Domingo-Ferrer. A survey of inference control methods for privacy-preserving data mining. In *Privacy-Preserving Data Mining - Models and Algorithms*, pages 53–80. 2008.

[12] V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. C. Courville. Adversarially learned inference. In *ICLR*, 2017.

[13] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[14] L. Gondara and K. Wang. MIDA: multiple imputation using denoising autoencoders. In *PAKDD*, pages 260–272, 2018.

[15] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[16] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics. Springer, 2009.

[17] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

[19] J. Jordon, J. Yoon, and M. van der Schaar. PATE-GAN: generating synthetic data with differential privacy guarantees. In *ICLR*, 2019.

[20] J. H. Jr, L. O. Gostin, and P. Jacobson. Legal issues concerning electronic health information: privacy, quality, and liability. *Jama*, 282.

[21] Kaggle. The state of data science and machine learning, 2017. `https://www.kaggle.com/surveys/2017`.

[22] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[23] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[24] H. Li, L. Xiong, L. Zhang, and X. Jiang. Dpsynthesizer: Differentially private data synthesizer for privacy preserving data sharing. *PVLDB*, 7(13):1677–1680, 2014.

[25] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115, 2007.

[26] P. Lu, P. Wang, and C. Yu. Empirical evaluation on synthetic data generation with generative adversarial network. In *WIMS*, pages 16:1–16:6, 2019.

[27] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. Are gans created equal? A large-scale study. In *NeurIPS*, pages 698–707, 2018.

[28] J. M. Mateo-Sanz, F. Sebé, and J. Domingo-Ferrer. Outlier protection in continuous microdata masking. In *Privacy in Statistical Databases*, pages 201–215, 2004.

[29] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. *CoRR*, abs/1611.02163, 2016.

[30] M. Mirza and S. Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

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

[32] Y. Park and J. Ghosh. Pegs: Perturbed gibbs samplers that generate privacy-compliant synthetic data. *Trans. Data Privacy*, 7(3):253–282, 2014.

[33] N. Patki, R. Wedge, and K. Veeramachaneni. The synthetic data vault. In *DSAA*, pages 399–410, 2016.

[34] PyTorch Developers. Tensors and dynamic neural networks in python with strong gpu acceleration. `https://pytorch.org`.

[35] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

[36] R. Ramakrishnan and J. Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003.

[37] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286, 2014.

[38] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *NIPS*, pages 2226–2234, 2016.

[39] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[40] S. Thirumuruganathan, S. Hasan, N. Koudas, and G. Das. Approximate query processing using deep generative models. *CoRR*, abs/1903.10000, 2019.

[41] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou. Differentially private generative adversarial network. *CoRR*, abs/1802.06739, 2018.

[42] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. Modeling tabular data using conditional GAN. *CoRR*, abs/1907.00503, 2019.

[43] L. Xu and K. Veeramachaneni. Synthesizing tabular data using generative adversarial networks. *CoRR*, abs/1811.11264, 2018.

[44] L. Yang, S. Chou, and Y. Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. In *ISMIR*, pages 324–331, 2017.

[45] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017.

[46] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: private data release via bayesian networks. In *SIGMOD*, pages 1423–1434, 2014.

[47] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. *ACM Trans. Database Syst.*, 42(4):25:1–25:41, 2017.

Table 9: Attributes details of the `Adult` Dataset

Type	Attributes	Details
Numerical	Age	17.0 – 90.0
	Fnlwgt	13492.0 – 1490400.0
	Education-num	1.0 – 16.0
	Capital-gain	0.0 – 99999.0
	Capital-loss	0.0 – 4356.0
	Hours-per-week	1.0 – 99.0
Categorical	Workclass	7
	Education	16
	Marital-status	7
	Occupation	14
	Relationship	6
	Race	5
	Sex	2
	Native-country	40
Label	Income	2

Table 10: Attributes details of the `Covertype` Dataset

Type	Attributes	Details
Continuous	Elevation	1860.0 – 3858.0
	Aspect	0.0 – 360.0
	Slope	0.0 – 66.0
	Hori-distance-hyd	0.0 – 1397.0
	Vert-distance	−173.0 – 601.0
	Hori-distance-road	0.0 – 7116.0
	Hillshade-9am	0.0 – 254.0
	Hillshade-noon	0.0 – 254.0
	Hillshade-3pm	0.0 – 254.0
	Hori-distance-fire	0.0 – 7173.0
Categorical	Wild-area	4
	Soil-type	40
Label	Cover-type	7

Table 11: Attributes details of the `SDataNum` Dataset

Type	Attributes	Details
Continuous	Col-0	−7.5032 – 7.3784
	Col-1	−7.6817 – 7.6576
Label	Col-2	2

Table 12: Attributes details of the `SDataCat` Dataset

Type	Attributes	Details
Categorical	Col-0	4
	Col-1	2
	Col-2	11
	Col-3	4
	Col-4	3
	Col-5	3
	Col-6	4
	Col-7	3
Label	Col-8	2

APPENDIX

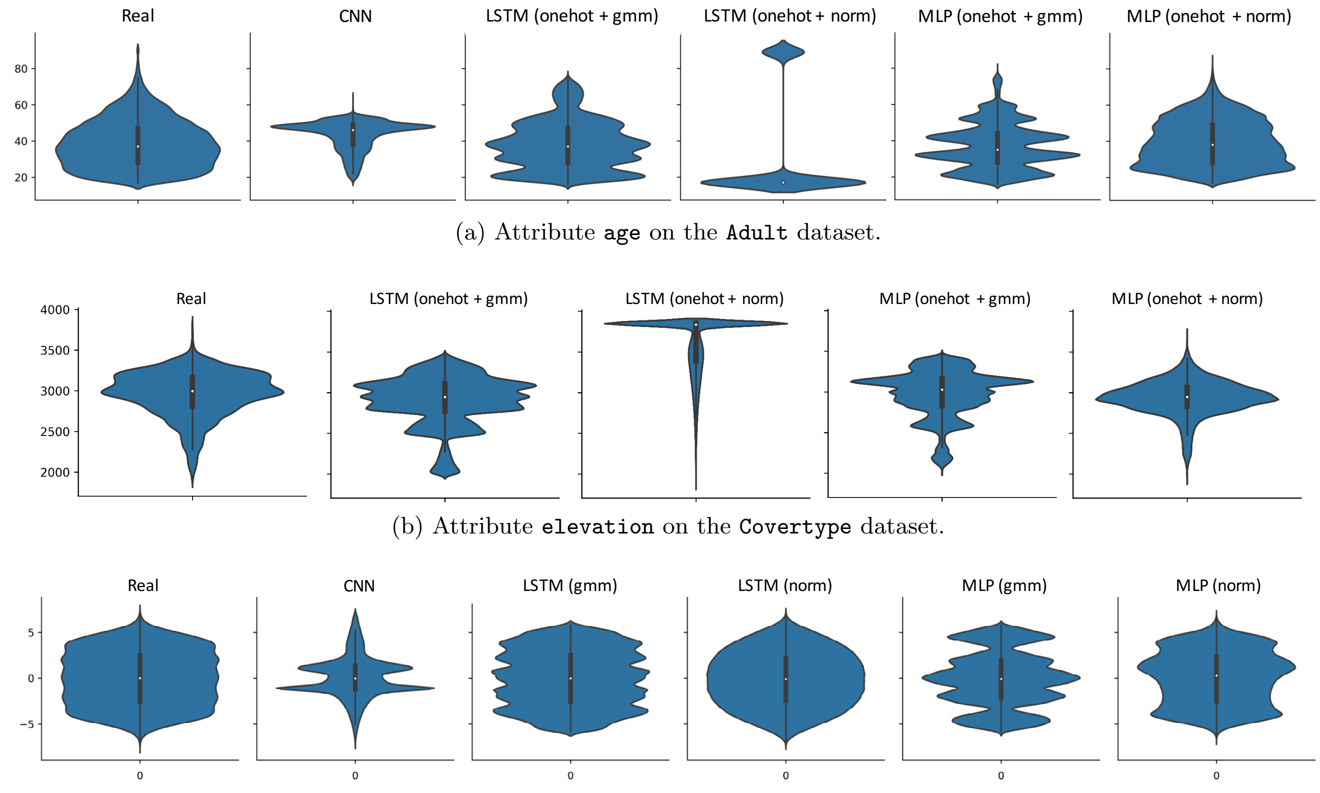
A. ATTRIBUTE INFORMATION OF DATASETS

The attribute information of the four datasets used in our experiments is described in Tables 9, 10, 11 and 12.

Figure 11 shows the distribution for numerical attributes using the violin plots, where we fix categorical attribute encoding as one-hot and focus on comparing simple (`nrm`) and GMM-based normalization (`gmm`). As observed, LSTM equipped with `gmm` can generate attributes having the most approximate distribution to their counterpart real attributes, and it is remarkably effective when the attribute has *multi-modal* distribution, e.g., `col0` on the `SDataNum` dataset. This is attributed to the Gaussian Mixture model used in this method, which is more powerful to represent multi-modal attributes. Moreover, it also outperforms MLP with `gmm`. This is because that LSTM uses two time steps to generate normalized value $v_{\texttt{gmm}}$ and components probabilities $\{\pi^{(i)}\}$ separately, which is shown more effective than generating them together in MLP. Figure 12 shows the distribution for categorical attributes. We can see that one-hot is significantly better than ordinal embedding. This is because values in a categorical attributes, such as `education` and `ocupation`, usually do not have ordinal relationships, and thus a single number is insufficient to represent the attribute.

Table 13: Evaluating neural network structures for the GAN model on the `Adult` dataset(With LSTM discriminator).

Classifier	MLP				LSTM			
	nrm +ord	nrm +hot	gmm +ord	gmm +hot	nrm +ord	nrm +hot	gmm +ord	gmm +hot
DT10	0.099	0.151	0.096	0.157	0.125	**0.085**	0.104	0.165
DT30	0.136	0.136	0.079	0.156	0.076	**0.069**	0.166	0.131
RF10	**0.041**	0.126	0.125	0.118	0.141	0.071	0.085	0.117
RF20	0.107	0.232	0.142	0.139	0.123	**0.083**	0.118	0.115
AdaBoost	0.105	0.277	0.126	0.143	0.089	0.126	**0.074**	0.156
LR	0.093	0.019	**0.008**	0.265	0.065	0.133	0.013	0.055

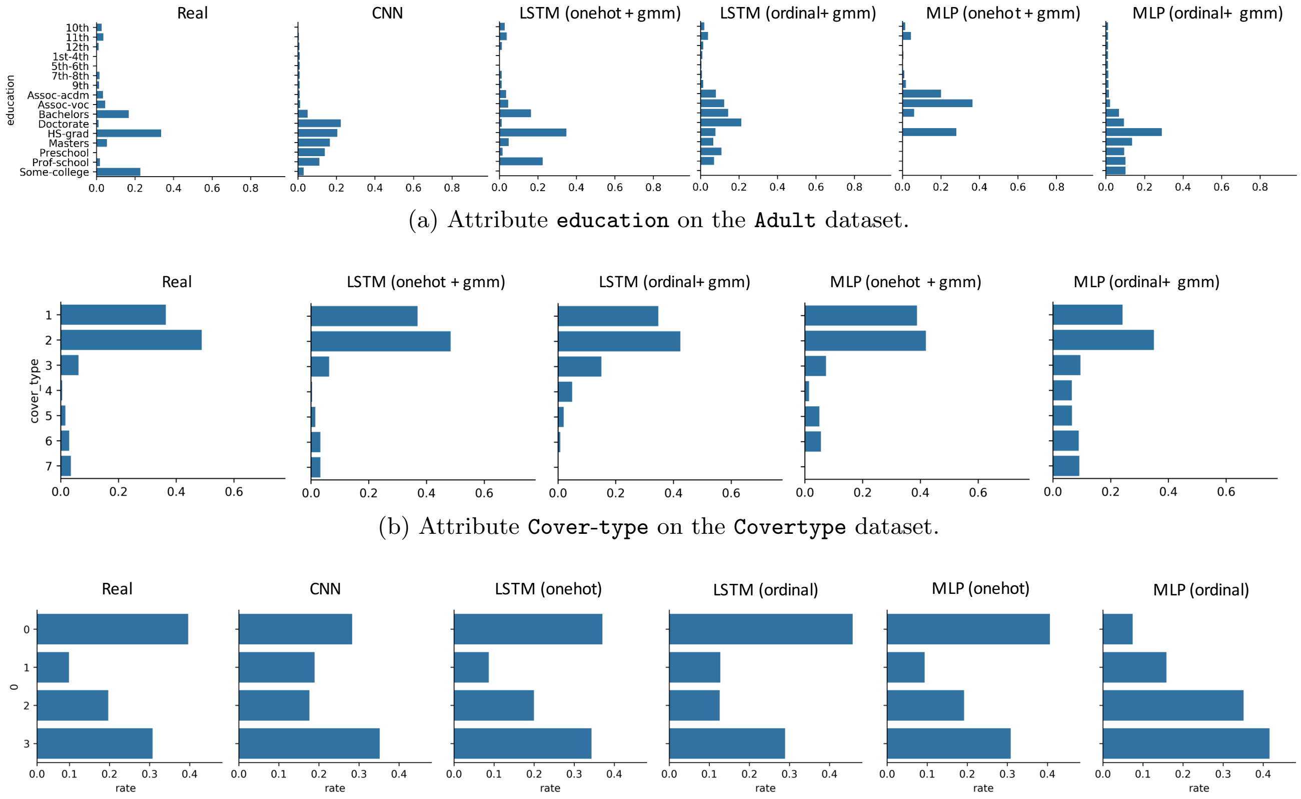


(a) Attribute `age` on the `Adult` dataset.

(b) Attribute `elevation` on the `Covertype` dataset.

(c) Attribute `col0` on the `SDataNum` dataset.

Figure 11: Evaluating value distribution of numerical attributes synthesized by different models.



(a) Attribute `education` on the `Adult` dataset.

(b) Attribute `Cover-type` on the `Covertype` dataset.

(c) Attribute `col0` on the `SDataCat` dataset.

Figure 12: Evaluating value distribution of categorical attributes synthesized by different models.

Algorithm 2: WGANTrain (m, α_d, α_g, T_d, T_g, c_p)

```
Input: $m$: batch size; $\alpha_d$: learning rate of $D$; $\alpha_g$: learning
       rate of $G$; $T_d$: number of iterations for $D$; $T_g$:
       number of iterations for $G$; $c_p$, clipping parameter
Output: $G$: Generator; $D$: Discriminator
Initialize parameters $\theta_d^{(0)}$ for $D$ and $\theta_g^{(0)}$ for $G$
for training iteration $t_1 = 1, 2, \dots, T_g$ do
    /* Using $T_d$ iterations to train $D$ */
    for training iteration $t_2 = 1, 2, \dots, T_d$ do
        Sample noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ from noise prior
        $p_z(\boldsymbol{z})$
        Sample samples $\{\boldsymbol{t}^{(i)}\}_{i=1}^{m}$ from real data $p_{data}(\boldsymbol{t})$
        $\bar{g}_1 \leftarrow \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [D(\boldsymbol{t}^{(i)}) - D(G(\boldsymbol{z}^{(i)}))]$
        $\theta_d^{(t_2)} \leftarrow \theta_d^{(t_2-1)} + \alpha_d \cdot \texttt{RMSProp}(\theta_d^{(t_2-1)}, \bar{g}_1)$
        $\theta_d^{(t_2)} \leftarrow \texttt{clip}(\theta_d^{(t_2)}, -c_p, c_p)$
    /* Training generator $G$ */
    Sample noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ from noise prior $p_z(\boldsymbol{z})$
    $\bar{g}_2 \leftarrow -\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} D(G(\boldsymbol{z}^{(i)}))$
    $\theta_g^{(t_1)} \leftarrow \theta_g^{(t_1-1)} - \alpha_g \cdot \texttt{RMSProp}(\theta_g^{(t_1-1)}, \bar{g}_2)$
return $G$, $D$
```

Algorithm 3: CGANTrain+ (m, α_d, α_g, T, Ω)

```
Input: $m$: batch size; $\alpha_d$: learning rate of $D$; $\alpha_g$: learning
       rate of $G$; $T$: number of training iterations; $\Omega$: label
       domain in real data
Output: $G$: Generator; $D$: Discriminator
Initialize parameters $\theta_d^{(0)}$ for $D$ and $\theta_g^{(0)}$ for $G$
for training iteration $t = 1, 2, \dots, T$ do
    for each label $y$ in $\Omega$ do
        Encode label $y$ as condition vector $\boldsymbol{c}$
        /* Training discriminator $D$ */
        Sample $m$ noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ from prior $p_z(\boldsymbol{z})$
        Sample $m$ samples $\{\boldsymbol{t}^{(i)}\}_{i=1}^{m}$ with label $y$ from real
        data $p_{data}(\boldsymbol{t}|y)$
        $\bar{g}_1 \leftarrow$
        $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(\boldsymbol{t}^{(i)}, \boldsymbol{c}) + \log(1 - D(G(\boldsymbol{z}^{(i)}, \boldsymbol{c}), \boldsymbol{c}))]$
        $\theta_d^{(t)} \leftarrow \theta_d^{(t-1)} + \alpha_d \cdot \texttt{Adam}(\theta_d^{(t-1)}, \bar{g}_1)$
        /* Training generator $G$ */
        Sample $m$ noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ from prior $p(\boldsymbol{z})$
        $\bar{g}_2 \leftarrow \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(\boldsymbol{z}^{(i)}, \boldsymbol{c}), \boldsymbol{c}))$
        $\theta_g^{(t)} \leftarrow \theta_g^{(t-1)} - \alpha_g \cdot \texttt{Adam}(\theta_g^{(t-1)}, \bar{g}_2)$
return $G$, $D$
```

Algorithm 4: DPGANTrain (m, α_d, α_g, T_d, T_g, c_p, c_g, σ_n)

```
Input: $m$: batch size; $\alpha_d$: learning rate of $D$; $\alpha_g$:
       learning rate of $G$; $T_d$: number of iterations for
       $D$; $T_g$: number of iterations for $G$; $c_p$: clipping
       parameter; $c_g$: bound on the gradient; $\sigma_n$:
       noise scale
Output: $G$: Generator; $D$: Discriminator
Initialize parameters $\theta_d^{(0)}$ for $D$ and $\theta_g^{(0)}$ for $G$
for training iteration $t_1 = 1, 2, \dots, T_g$ do
    /* Using $T_d$ iterations to train $D$ */
    for training iteration $t_2 = 1, 2, \dots, T_d$ do
        Sample noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ from prior $p_z(\boldsymbol{z})$
        Sample samples $\{\boldsymbol{t}^{(i)}\}_{i=1}^{m}$ from real data $p_{data}(\boldsymbol{t})$
        for each $i$ do
            $g_1(\boldsymbol{t}^{(i)}, \boldsymbol{z}^{(i)}) \leftarrow \nabla_{\theta_d} [D(\boldsymbol{t}^{(i)}) - D(G(\boldsymbol{z}^{(i)}))]$
        $\bar{g}_1 \leftarrow \frac{1}{m} (\sum_{i=1}^{m} g_1(\boldsymbol{t}^{(i)}, \boldsymbol{z}^{(i)}) + N(0, \sigma_n^2 c_g^2 I))$
        $\theta_d^{(t_2)} \leftarrow \theta_d^{(t_2-1)} + \alpha_d \cdot \texttt{RMSProp}(\theta_d^{(t_2-1)}, \bar{g}_1)$
        $\theta_d^{(t_2)} \leftarrow \texttt{clip}(\theta_d^{(t_2)}, -c_p, c_p)$
    /* Training generator $G$ */
    Sample noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ from prior $p_z(\boldsymbol{z})$
    $\bar{g}_2 \leftarrow -\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} D(G(\boldsymbol{z}^{(i)}))$
    $\theta_g^{(t_1)} \leftarrow \theta_g^{(t_1-1)} - \alpha_g \cdot \texttt{RMSProp}(\theta_g^{(t_1-1)}, \bar{g}_2)$
return $G$, $D$ ;
```