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,kanamemadoka,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 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 [32]. 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 regulations and privacy concerns. For example, a hospital wants to share its electronic health records (EHR) 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 [31].

To address the difficulties, *privacy-preserving data publishing* has been extensively studied recently to provide a safer way for data sharing [14, 37, 21, 8, 59, 60]. However, the existing solutions suffer from the limitations on effectively balancing privacy and utility of the released data [43]. Therefore, efforts have been made recently in the database and machine learning communities to apply *generative adversarial networks (GAN)* to relational data synthesis [16, 55, 56, 43, 11, 18, 38]. The main advantages of GAN are as follows. First, different from the conventional methods [14, 37, 8, 59] 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 [43]. 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 clustering.

However, compared with the success of using GAN for image generation [39], GAN-based relational data synthesis is still in its infancy stage. Despite some very recent attempts [16, 55, 56, 43, 11, 18, 38], 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 [47]) 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) [34, 49], and the state-of-the-art data synthesis approach using statistical models [59, 60]. 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 [23]. 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).

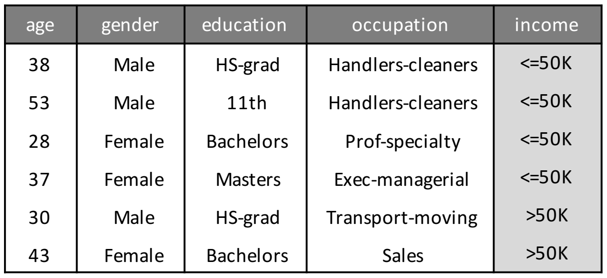
2. RELATIONAL DATA SYNTHESIS

2.1 Problem Formalization

This paper focuses on a relational table $\mathcal{T}$ of n records, i.e., $\mathcal{T} = \{t_1, t_2, \dots, t_n\}$. 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. 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.

(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 [16, 55, 56, 43, 11, 18, 38]. 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. 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 $f : X \to Y$ 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). Similarly, we can train a classifier f' on the synthetic table $\mathcal{T}'$ and evaluate the classifier

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



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 relational table, with one numerical attribute age, three categorical attributes gender, education and occupation and a label income.

on the same $\mathcal{T}_{\texttt{test}}$ to obtain its performance $\texttt{Eval}(f'|\mathcal{T}_{\texttt{test}})$. The utility of $\mathcal{T}'$ is measured by the difference between these two classifiers' performance metrics, i.e.,

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

EXAMPLE 1 (SYNTHETIC DATA UTILITY). *Consider an example table $\mathcal{T}$ in Figure 1, where label* income *has two unique values: 0 (*income $\leq 50K$*) and 1 (*income $> 50K$*). We use $\mathcal{T}$ to train a 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 test table $\mathcal{T}_{\texttt{test}}$. We measure the performance difference of these two models as $\texttt{Diff}(\mathcal{T}, \mathcal{T}')$ 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 [44, 38, 40], 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.

Synthetic data generation for Clustering. For example, suppose that a hospital wants to ask a CS team to develop a clustering algorithm that discovers groups of similar patients. It can first share the synthetic data to the team for ease of algorithm development. Then, it deploys the developed algorithm in the hospital to discover groups on the original data. In this case, the data utility is that a clustering algorithm should achieve similar performance on both original table $\mathcal{T}$ and fake table $\mathcal{T}'$. Let $\mathcal{C} = \{c_1, c_2, \dots, c_k\}$ and $\mathcal{C}' = \{c_1', c_2', \dots, c_k'\}$ respectively denote the sets of clusters discovered by a clustering algorithm on $\mathcal{T}$ and $\mathcal{T}'$. We can use a standard evaluation metric for clustering, such as normalized mutual information (NMI), to examine the quality of $\mathcal{C}$ and $\mathcal{C}'$. Then, the utility of $\mathcal{T}'$ for clustering is measured by the difference between these two metrics, i.e., $\texttt{Diff}_{\texttt{CST}}(\mathcal{T}, \mathcal{T}') = |\ \texttt{Eval}(\mathcal{C}|\mathcal{T}) - \texttt{Eval}(\mathcal{C}'|\mathcal{T}')\ |$, where $\texttt{Eval}(\mathcal{C}|\mathcal{T})$ ($\texttt{Eval}(\mathcal{C}'|\mathcal{T}')$) is the evaluation metric for clusters $\mathcal{C}$ ($\mathcal{C}'$) from original table $\mathcal{T}$ (fake table $\mathcal{T}'$). Intuitively, we prefer a smaller $\texttt{Diff}_{\texttt{CST}}$ for preserving the utility.

Synthetic data generation for *approximate query processing (AQP)* [15, 52]. For example, suppose that a user wants to perform data exploration or visualization on a large dataset. To reduce latency, some work [52] introduces a *lightweight* approach that utilizes synthetic data in the client to quickly answer aggregate queries, without communicating with the server. To support this, the synthetic data should preserve the utility that answers aggregate queries as accurate as possible to the original data $\mathcal{T}$. To formally measure the data utility, we adopt the *relative error difference* [52], as defined as below. For each aggregate query q, we compute the relative error e' over the synthetic table $\mathcal{T}'$, and the relative error e over a fixed size sample obtained from $\mathcal{T}$. Then, we compute the relative error difference as the absolute difference between these two errors, $\texttt{Diff}_{\texttt{AQP}}(\mathcal{T}, \mathcal{T}'|q) = | e - e' |$. Given a workload with a set Q of queries, we compute the average $\texttt{Diff}_{\texttt{AQP}}(\mathcal{T}, \mathcal{T}') = \sum_{q \in Q} \texttt{Diff}_{\texttt{AQP}}(\mathcal{T}, \mathcal{T}'|q)/|Q|$.

2.2 Related Works for Data Synthesis

Data synthesis has been extensively studied in the last decades, and the existing approaches 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 [35, 45], Bayesian networks [59, 60], Gibbs sampling [44] and Fourier decompositions [12]. Synopses-based approaches, such as wavelets and multi-dimensional sketches, build compact data summary for massive data [19, 53], which can be then used for estimating joint distribution. As the statistical models may have limitations on effectively balancing privacy and data utility, 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 [25] and variational autoencoders (VAE) [52], while more attentions are paid on generative adversarial networks (GAN) [16, 55, 56, 43, 11, 18, 38].

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 [14, 37] and perturbation [21, 8]. However, the existing study [43] has shown that GAN-based data synthesis outperforms these techniques.

2.3 Generative Adversarial Networks (GAN)

Generative adversarial networks (GAN) [26, 39], are a kind of deep generative models, which have achieved breakthroughs in many areas, such as image generation [47, 42, 17], and sequence generation [58, 57]. 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., $\min_G \max_D V(G, D) = \mathbb{E}_{\boldsymbol{t} \in p_{data}(\boldsymbol{t})} \big[\log D(\boldsymbol{t})\big] + \mathbb{E}_{\mathbf{z} \in p_z(\mathbf{z})} \big[1 - \log D(G(\mathbf{z}))\big]$, 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 $\mathbf{z}$.

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.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. 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. 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.

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.

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 [42] 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 in Figure 1, the framework transforms each record into a sample. Suppose that we adopt ordinal encoding for categorical attributes. The first record is transformed to* $[38, 0, 0, 0, 0]$. *Then, it uses the transformed samples to train the GAN model for obtaining an optimized generator G.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 examine how to encode categorical attributes to numerical values, and normalize numerical values to appropriate ranges that fit neural networks. We consider widely-used encoding schemes for cat-

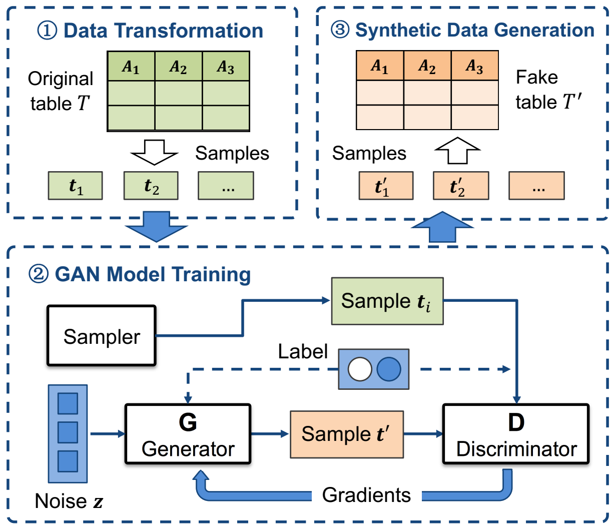


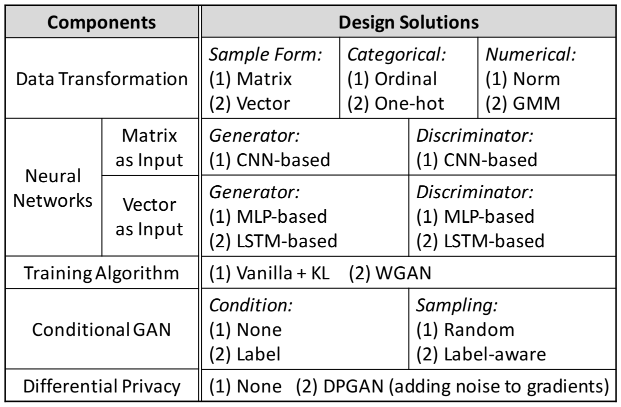
Figure 2: Overview of data synthesis using GAN. (1) It transforms each record in a relational table into a sample $t \in \mathbb{R}^d$. (2) It takes the samples as input to train a deep generative model G using the adversarial training framework in GAN. (3) It utilizes the trained G to generate a set of synthetic samples, which are then transformed back into fake records.

egorical 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). We will take exploration of more sophisticated transformation schemes as a future work. Moreover, as different neural networks have different requirements for the input, sample t can be in the form of either *matrix* or *vector*. More details of data transformation are in Section 4.

Neural networks. Existing works for relational data synthesis consider three representative neural networks. (1) Inspired by the success of image synthesis, some apply DC-GAN [47], and use Convolutional Neural Networks (CNN) for G and D, in which G is a deconvolution process and D is a convolution process [16, 43]. (2) Following the original GAN [26], some studies [18, 55] use multilayer perceptron (MLP) consisting of multiple fully-connected layers. (3) Some approaches utilize a *sequence generation* mechanism that generates attributes separately in sequential time-steps [56], and use recurrent neural networks, such as long short-term memory (LSTM) networks [28] for G. Note that different neural networks have different forms of input: CNN takes matrix-formed samples, while MLP and LSTM uses vector-formed samples. This paper focuses on comparing the aforementioned representative neural networks under the same framework. We will take an exploration of more sophisticated models, such as Bidirectional LSTM [27], as a future work. More details can be referred to Section 5.1.

Training algorithm. Minibatch-based stochastic gradient descent (SGD) strategy is applied for GAN training. This paper focuses on investigating *mode collapse* [50, 41], a well-recognized challenge in GAN training. To this end, we evaluate different training algorithms 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 [26] with an improved loss function and (2) Wasserstein GAN (WGAN) training [10]. See Section 5.2 for more details of these algorithms.

Conditional GAN. The imbalanced label distribution in real-world data may result in *insufficient training* for records



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.

with minority labels [55]. Thus, some studies [55] apply conditional GAN [42] to data synthesis. We examine the adoption of conditional GAN that encodes a label as a condition vector 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. We consider differential privacy [23], a well-adopted formalization of data privacy, to evaluate whether GAN can still be effective to preserve data utility while providing provable privacy protection. 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 [54] in the GAN training process, as elaborated in Section 5.4.

4. DATA TRANSFORMATION

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

Categorical attribute transformation. We consider two commonly-used label encoding schemes.

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

2) One-hot encoding 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.

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

1) Simple normalization uses $\mathcal{T}[j].\max$ and $\mathcal{T}[j].\min$ to respectively denote the maximum and minimum values of attribute $\mathcal{T}[j]$. Given an original value v in $\mathcal{T}[j]$, it normalizes the value as $v_{\texttt{norm}} = -1 + 2 \cdot \frac{v - \mathcal{T}[j].\min}{\mathcal{T}[j].\max - \mathcal{T}[j].\min}$.

2) GMM-based normalization. Some existing studies [55, 56] 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. 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 $v_{\texttt{gmm}} = \frac{v-\mu^{(k)}}{2\sigma^{(k)}}, where\ k = \arg\max_i \pi^{(i)}$. 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 $(-0.7, 0, 1)$ where $(0, 1)$ indicates the second 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 $\boldsymbol{t}$.

1) Matrix-formed samples. For CNN-based neural networks, we follow the method in [43] to convert attributes into a square matrix. Note that this method requires each attribute is transformed into one value instead of a vector (otherwise, the vector of an attribute may be split in the 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 shown in Figure 1. When transforming the record into a matrix-formed sample, we can only apply ordinal encoding and simple normalization and obtain a square matrix* $((0.2, 1, 2), (4, 1, 0), (0, 0, 0))$. *In contrast, when transforming the record into a vector-formed sample, we may choose to use one-hot encoding and GMM-based normalization, and obtain* $(\underline{-0.7, 0, 1}, \underline{0, 1}, \underline{0, 0, 1, 0, 0, 0, 0, 0, 1}, \underline{0, 1})$, *where the underlines indicate different attributes.*

5. GAN MODEL DESIGN

5.1 Neural Network Architectures

We describe the basic idea of neural networks evaluated in this paper, and leave more details in our report [24].

CNN: convolutional neural networks. CNN is utilized in the existing works for data synthesis [16, 43]. Generator G takes as input a prior noise $\boldsymbol{z}$, which is denoted by $\boldsymbol{h}_g^0$. It then uses L de-convolution layers $\{\boldsymbol{h}_g^l\}$ (i.e., fractionally strided convolution) to transform $\boldsymbol{z}$ to a synthetic sample in the form of matrix, where $\boldsymbol{h}_g^{l+1} = \texttt{ReLU}(\texttt{BN}(\texttt{DeConv}(\boldsymbol{h}_g^l)))$ and $\boldsymbol{t} = \texttt{tanh}(\texttt{DeConv}(\boldsymbol{h}_g^L))$. Discriminator D takes as input a real/fake sample $\boldsymbol{t}$ in matrix form, which is denoted by $\boldsymbol{h}_d^0$. It applies L convolution layers $\{\boldsymbol{h}_d^l\}$ where $\boldsymbol{h}_d^{l+1} = \texttt{LeakyReLU}(\texttt{BN}(\texttt{Conv}(\boldsymbol{h}_d^l)))$ and `Conv` is a convolution function. Finally, D outputs a probability indicating how likely $\boldsymbol{t}$ is real, i.e., $f = \texttt{sigmoid}(\texttt{BN}(\texttt{Conv}(\boldsymbol{h}_d^L)))$.

MLP: fully connected neural networks. MLP is used in the existing works [18, 55]. In this model, G takes as input noise $\boldsymbol{z}$, which is also denoted by $\boldsymbol{h}^{(0)}$, and utilizes with L fully-connected layers, where each layer is computed by $\boldsymbol{h}^{l+1} = \phi\big(\texttt{BN}(\texttt{FC}_{|\boldsymbol{h}^l| \to |\boldsymbol{h}^{l+1}|}(\boldsymbol{h}^l))\big)$, where $\texttt{FC}_{|\boldsymbol{h}^l| \to |\boldsymbol{h}^{l+1}|}(\boldsymbol{h}^l) = \boldsymbol{W}^l \boldsymbol{h}^l + \boldsymbol{b}^l$ 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 [29]. 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.

One issue 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 $\mathcal{T}[j]$, e.g., using `tanh` and `softmax` for simple normalization and one-hot encoding respectively. In particular, for GMM-based normalization, we adopt the following method in [55]. We first use $\texttt{tanh}(\texttt{FC}_{|\boldsymbol{h}^L| \to 1}(\boldsymbol{h}^L))$ to generate $v_{\texttt{gmm}}$ and then use $\texttt{softmax}(\texttt{FC}_{|\boldsymbol{h}^L| \to |\boldsymbol{t}_j|-1}(\boldsymbol{h}^L))$ 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.

LSTM: recurrent neural networks. The basic idea is to formalize record synthesis as a *sequence generation* process [56]: 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 $\boldsymbol{h}^{j+1} = \texttt{LSTMCell}(\boldsymbol{z}, \boldsymbol{f}^j, \boldsymbol{h}^j)$ and $\boldsymbol{f}^{j+1} = \texttt{tanh}(\texttt{FC}_{|\boldsymbol{h}^{j+1})| \to |\boldsymbol{f}^{j+1})|}(\boldsymbol{h}^{j+1}))$, where $\boldsymbol{h}^0$ and $\boldsymbol{f}^0$ are initialized with random values. to realize discriminator D, we use a typical *sequence-to-one* LSTM [51].

Similar to MLP, we make the output layer in G attribute aware by considering transformation method for each attribute. For example, for attribute $t[j]$ transformed by GMM-based normalization, we use two timesteps to generate its sample $\boldsymbol{t}_j$, and concatenate these two parts.

5.2 GAN Training and Mode Collapse

We apply the vanilla GAN training algorithm [26] (VTRAIN) to iteratively optimize parameters θ_d in D and θ_g in G. In each iteration, it trains D and D alternately. As the algorithm may not provide sufficient gradient to train G in the early iterations [26], existing work [56] introduces the KL divergence between real and synthetic data to warm up model training. 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 $\{G(\boldsymbol{z}^{(i)})\}_{i=1}^m$. Based on this, we optimize G by considering the original loss and KL divergences regarding all attributes, i.e.,

$$\mathcal{L}_G = \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] + \sum_{j=1}^{|\mathcal{T}|} \texttt{KL}(\mathcal{T}[j], \mathcal{T}'[j]), \quad (2)$$

Mode Collapse. This paper investigates *mode collapse* [50, 41], a well-recognized challenge in GAN training. To be more specific, mode collapse would result in very similar, or even nearly duplicated records in synthetic table $\mathcal{T}'$. The reason is that generator G would generate a limited diversity of samples, regardless of the input noise. Then, as synthetic records are transformed from the samples (see Section 4),

Table 1: Comparison of training algorithms.

Algorithm	Loss	Optimizer	Sampling	DP
VTrain	Eq.(2)	`Adam`	random	×
WTrain	Eq.(3)	`RMSProp`	random	×
CTrain	Eq.(4)	`Adam`	label-aware	×
DPTrain	Eq.(3)	`RMSProp`	random	√

many records will have the same values for most of the attributes as well as the labels. As a result, the synthetic table $\mathcal{T}'$ would fail to preserve the data utility of original table $\mathcal{T}$. For example, a classifier trained on $\mathcal{T}'$ may perform badly, and it sometimes achieves very low F1 scores on the test set. A deep investigation further shows that, when mode collapse happens, G cannot get sufficient gradient in training iterations and the training algorithm fails to decrease the loss of G. In this case, G won't converge and may overfit to a few training records.

We study how to avoid mode collapse by examining the following two strategies. The first one is to utilize the training algorithm of Wasserstein GAN (WTrain) [10], which is commonly used for addressing mode collapse in image synthesis. Different from VTrain, WTrain removes the `sigmoid` function of D and changes the gradient optimizer from `Adam` to `RMSProp`. It uses the loss functions as

$$\begin{aligned} \mathcal{L}_D &= -\mathbb{E}_{\boldsymbol{t}\sim p_{data}(\boldsymbol{t})}[D(\boldsymbol{t})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))] \\ L_G &= -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))]. \end{aligned} \tag{3}$$

The second strategy is to still use the vanilla GAN training algorithm VTrain, but simplify the neural network of discriminator D. The idea is to make D not trained too well, and thus avoid the chance of gradient disappearance of generator G. Based on some theoretical analysis [26, 9], if D is perfectly trained, the loss of G would become a constant and the gradient of G will be vanishing. In order to avoid such circumstance, we may choose to use a relatively simple neural network to realize D, e.g., reducing the numbers of layers or neurons in the neural network.

5.3 Conditional GAN

The basic idea of conditional GAN 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, i.e.,

$$\begin{aligned} \min_G \max_D V(G,D) =& \mathbb{E}_{\boldsymbol{t}\in p_{data}(\boldsymbol{t})}\big[\log D(\boldsymbol{t}|\boldsymbol{c})\big] \\ &+ \mathbb{E}_{\mathbf{z}\in p_z(\mathbf{z})}\big[1-\log D(G(\mathbf{z}|\mathbf{c}))\big]. \end{aligned} \tag{4}$$

One obstacle is that, 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 (CTrain). 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.

Table 2: Real datasets for our evaluation: #Rec is the number of records, #C and #N are the numbers of numerical and categorical attributes, and #L is the number of unique labels.

Dataset	Domain	#Rec	#N	#C	#L	Skewness
low-dimensional (#Attr≤ 20)						
`HTRU2` [5]	Physical	17,898	8	0	2	skew
`Digits` [6]	Computer	10,992	16	0	10	balanced
`Adult` [1]	Social	41,292	6	8	2	skew
`CovType` [4]	Life	116,204	10	2	7	skew
high-dimensional (#Attr> 20)						
`SAT` [7]	Physical	6,435	36	0	6	balanced
`Anuran` [2]	Life	7,195	22	0	10	skew
`Census` [3]	Social	142,522	9	30	2	skew
`Bing` [36]	Web	500,000	7	23	-	-

5.4 Differential Privacy Preserving GAN

We apply DPGAN [54] to enable our data synthesizer to support 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 minor modifications (DPTrain). See the original paper [54] for more details.

Algorithm comparison. All the algorithms in Sections 5.2 - 5.4 share the same optimization framework, i.e., minibatch stochastic gradient descent, but use different strategies. Table 1 compares the algorithms in loss function, gradient optimizer, sampling and differential privacy (DP) supporting. We also present the pseudo-codes of all training algorithms in our technical report [24] due to the space limit.

6. EVALUATION METHODOLOGY

6.1 Datasets

To consider various characteristics of relational data, e.g., mixed data types, attribute correlation and label skewness, we use 8 real datasets from diverse domains, such as Physical and Social. The datasets are representative that capture different data characteristics, as summarized in Table 2. First, they have different numbers of attributes (#`Attr`), which may affect the performance of data synthesis. For simplicity, we consider low-dimensional (#`Attr` ≤ 20) and high-dimensional (#`Attr` > 20). Second, they have different attribute types. For both high- and low-dimensional datasets, we differentiate them into *numerical* with only numerical attributes and *mixed* with both numerical and categorical attributes. Third, they have different label skewness. We consider a dataset is *skew* if the ratio between numbers of records with the most popular and the rarest labels is larger than 9. We use four low-dimensional datasets, `HTRU2`, `Digits`, `Adult` and `CovType`. Among them, `HTRU2` and `Digits` only contain numerical attributes, while `Adult` and `CovType` have both numerical and categorical attributes. Moreover, the datasets, except `Digits`, are skew in label distribution. For high-dimensional datasets, we use two numerical datasets, `SAT` and `Anuran` and two mixed datasets, `Census` and `Bing`, and both balanced and skew cases are considered. In particular, `Bing` is a Microsoft production workload dataset, which is used for evaluating AQP in the existing work [36]. Thus, we only use the `Bing` dataset for AQP in our experiments. Due to the space limit, we leave more details of the datasets in our technical report [24].

To provide in-depth analysis on synthesis performance by varying degrees of attribute correlation and label skewness, we also use two sets of simulated datasets.

(1) `SDataNum` datasets are used to evaluate data synthesis for records with purely *numerical* attributes. We follow the simulation method in [55] to first generate 25 two-dimensional variables, each of which follows Gaussian distribution $f(x, y) = \mathcal{N}(\mu_x, \mu_y; \sigma_x, \sigma_y)$ where the means μ_x, μ_y are randomly picked from the set of two-dimensional points, $(u, v)|u, v \in \{-4, -2, 0, 2, 4\}$ and standard deviation σ_x (σ_y) follows $\texttt{uniform}(0.5, 1)$. Then, we iteratively generate records, in which each record is randomly sampled from one of the 25 Gaussian variables, and assigned with a binary label.

In the simulation, we control *attribute correlation* by varying correlation coefficient $\rho_{xy} = cov(x, y)/\sqrt{\sigma_x \sigma_y}$ in each Gaussian distribution. We consider two degrees of attribute correlation by setting the coefficients to 0.5 and 0.9 respectively. We also control *label skewness* by varying the ratio between positive and negative labels assigned to the records. We consider two settings on skewness: `balanced` with ratio $1 : 1$ and `skew` with ratio $1 : 9$.

(2) `SDataCat` datasets are used to evaluate the synthesis of records with purely *categorical* attributes. We generate 5 categorical attributes as follows. We first construct a chain Bayesian network with 5 nodes linked in a sequence, each of which corresponds to a random variable. Then, we generate each record by sampling from the joint distribution modeled by the network, and assign it with a binary label.

We control *attribute correlation* by varying the conditional probability matrix associated with each edge in the Bayesian network. Specifically, we let the diagonal elements to be a specific value p and set the remaining ones uniformly. Intuitively, the larger the p is, the higher dependencies the attributes possess. For example, in an extreme case that $p = 1$, each attribute (except the first one) deterministically depends on its previous attribute in the network. In such a manner, we consider two degrees of attribute correlation by setting $p = 0.5$ and $p = 0.9$ respectively. Moreover, similar to `SDataNum` datasets, we also consider `balanced` and `skew` settings for label skewness on these datasets.

6.2 Evaluation Framework

We implement our GAN-based relational data synthesis framework, as shown in Figure 2, using PyTorch [46]

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, following the existing works for relational data synthesis. 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 of 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 for parameter optimization. We divide the training iterations in the algorithm evenly into 10 *epochs* and evaluate the performance of the model snapshot 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 data utility and privacy protection.

Evaluation on data utility for classification. 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): We adopt 2 decision trees with max depth 10 and 30 respectively. (2) Random Forest (RF): We adopt two random forests with max depth 10 and 20 respectively. (3) AdaBoost (AB): 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 $\mathcal{T}_{\texttt{test}}$. 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, we measure the F1 score of the rare label, which is more difficult to predict than others. We evaluate the performance of a data synthesizer by measuring the difference `Diff` of the F1 scores between f' and f, as defined in Section 2.1. The smaller the difference is, the better $\mathcal{T}'$ is for ML training. Note that we also consider area under the receiver operating characteristic curve (AUC) as evaluation, and obtain similar trends with that of F1 score.

Evaluation on data utility for clustering. We evaluate the performance of the well-known clustering algorithm K-Means on both $\mathcal{T}_{\texttt{train}}$ and $\mathcal{T}'$. Note that we exclude the label attribute from the features fed into K-Means and instead use it as the gold-standard. We use *normalized mutual information (NMI)* to evaluate the clustering performance. NMI measures the mutual information, i.e., reduction in the entropy of gold-standard labels that we get if we know the clusters, and a larger NMI indicates better clustering performance. After obtaining NMI scores from both the clustering results on $\mathcal{T}_{\texttt{train}}$ and $\mathcal{T}'$, we compute the absolute difference of the scores as $\texttt{Diff}_{\texttt{CST}}$, which is defined in Section 2.1, and use $\texttt{Diff}_{\texttt{CST}}$ to measure the utility of $\mathcal{T}'$ for clustering.

Evaluation on data utility for AQP. We use the fake table $\mathcal{T}'$ to answer a given workload of aggregation queries. We follow the query generation method in [36] to generate 1,000 queries with aggregate functions (i.e., `count`, `avg` and `sum`), selection conditions and groupings. We also run the same queries on the original table $\mathcal{T}_{\texttt{train}}$. For each query, we measure the relative error e' of the result obtained from $\mathcal{T}'$ by comparing with that from $\mathcal{T}_{\texttt{train}}$. Meanwhile, following the method in [52], we draw a fixed size random sample set (1% by default) from the original table, run the queries on this sample set, and obtain relative error e for each query. To eliminate randomness, we draw the random sample sets for 10 times and compute the averaged e for each query. Then, as mentioned in Section 2.1, we compute the relative error difference $\texttt{Diff}_{\texttt{AQP}}$ and average the difference for all queries in the workload, to measure the utility of $\mathcal{T}'$ for AQP.

Evaluation on privacy protection. We adopt the following two metrics, which are widely used in the existing works [44, 38, 40] 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 syn-

thetic records from $\mathcal{T}'$. For each sampled record, we measure 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 [44, 38]. 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 shown in Figure 3. We use the code provided by [43] to implement the CNN-based model[2]. 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.

Statistical methods. We compare GAN with a state-of-the-art statistical data synthesis method PrivBayes (or `PB` for simplicity) [59, 60], using the source code downloaded here[3]. As `PB` has theoretical guarantee on differential privacy [23], we vary the privacy parameter ϵ to examine the tradeoff between privacy protection and data utility. According to the original papers [59, 60], we run `PB` in multiple times and report the average result.

Variational Autoencoder (VAE). We implement variational autoencoder (VAE), which is another representative deep generative model [34, 49] for relational data synthesis. We adopt the loss function that consists of both the reconstruction loss and the KL divergence [20]. 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 [39] for hyper parameter search. Given a GAN model, we firstly generate a set of candidate hyper parameter settings. Then, we train the model for several times, and at each time, we randomly select a hyper parameter setting and evaluate the trained model on the validation set $\mathcal{T}_{\texttt{valid}}$. Based on this, we select the hyper parameter setting that results in a model with the best performance.

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

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

Table 3: Evaluating different neural networks of generator G on synthetic data utility for classification, where `CLF` stands for classifier. For low-dimensional datasets with less attributes, LSTM with appropriate transformation achieves much less F1 differences than MLP and CNN. For high-dimensional datasets with more attributes, the performance advantage of LSTM over MLP becomes less significant.

(a) `Adult` dataset (low-dimensional).

CLF	CNN	MLP				LSTM			
		sn/od	sn/ht	gn/od	gn/ht	sn/od	sn/ht	gn/od	gn/ht
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**
AB	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

(b) `CovType` dataset (low-dimensional).

CLF	MLP				LSTM			
	sn/od	sn/ht	gn/od	gn/ht	sn/od	sn/ht	gn/od	gn/ht
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**
AB	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**

(c) `Census` dataset (high-dimensional).

CLF	CNN	MLP				LSTM			
		sn/od	sn/ht	gn/od	gn/ht	sn/od	sn/ht	gn/od	gn/ht
DT10	0.484	0.188	0.119	0.119	**0.113**	0.211	0.332	0.162	0.180
DT30	0.462	0.172	**0.106**	0.116	0.114	0.288	0.288	0.185	0.157
RF10	0.214	**0.007**	0.038	0.050	0.028	0.132	0.035	0.058	0.012
RF20	0.410	0.166	**0.051**	0.053	0.095	0.189	0.244	0.109	0.089
AB	0.506	0.215	0.107	0.127	**0.063**	0.144	0.239	0.082	0.113
LR	0.494	0.133	**0.047**	0.085	0.069	0.358	0.085	0.250	0.053

(d) `SAT` dataset (high-dimensional).

CLF	MLP		LSTM	
	sn	gn	sn	gn
DT10	0.098	0.048	0.047	**0.042**
DT30	0.063	0.090	0.041	**0.041**
RF10	0.155	0.100	0.149	**0.051**
RF20	0.181	0.108	0.157	**0.093**
AB	**0.017**	0.065	0.040	0.182
LR	**0.009**	0.014	0.018	0.017

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.

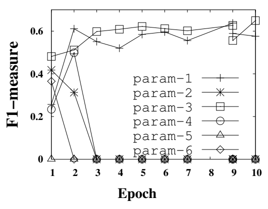
7. EVALUATION RESULTS

7.1 Evaluating GAN-based Framework

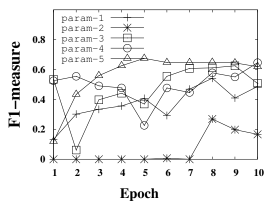
This section explores the design space of our GAN-based framework. We focus on synthetic data utility for classification, and report privacy results and data utility for clustering and AQP in next sections.

7.1.1 *Evaluation on Neural Networks*

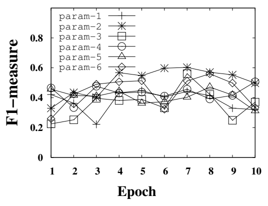
We evaluate the neural networks, CNN, MLP and LSTM that realize the generator G in our framework. For MLP and LSTM, we fix the discriminator D as MLP. We also evaluate the LSTM-based discriminator and obtain inferior result (the result is included in our technical report [24]). We first evaluate the data utility for classification model training. Due to the space limit, we report the results on two low-dimensional (#`Attr` ≤ 20) datasets `Adult` and `CovType`, and two high-dimensional ones `SAT` and `Census`, and we find



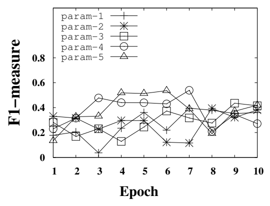
(a) LSTM-based G (`Adult`).



(b) LSTM-based G (`CovType`).



(c) MLP-based G (`Adult`).



(d) MLP-based G (`CovType`).

Figure 4: Evaluating GAN model training on various hyper-parameter settings. MLP-based generator is more robust against various hyper parameters, while LSTM is likely to result in mode collapse.

similar results on other datasets. Tables 3(a), 3(b), 3(c) and 3(d) report the experimental results on data utility for classification model training, where `sn`, `gn`, `od`, and `ht` respectively denote simple normalization, GMM-based normalization, ordinal encoding and one-hot encoding. Note that CNN is not evaluated on `CovType` and `SAT`, as the original code in [43] is not designed for multi-class classification.

On datasets `Adult` and `CovType` with less attributes, LSTM achieves the best performance in most of the cases, i.e., achieving 7% – 90% less F1 difference than the second best model MLP. This suggests the *sequence generation* mechanism in LSTM, which generates a record attribute by attribute, is more adequate for relational data synthesis. Firstly, 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 would be capable of capturing column correlation. Nevertheless, on datasets `Census` and `SAT` with more attributes, the performance advantage of LSTM is less significant. The reason is that, with the increase of attribute numbers, it becomes more difficult for LSTM to capture correlation among attributes, which implies that more effective models should be invented for data synthesis.

CNN achieves inferior performance in data synthesis, which is different from image synthesis [47]. This is because matrix input of CNN is only compatible with simple normalization and ordinal encoding, which is not effective for relational data. Moreover, 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. However, relational data does not have such locality.

Finding 1: LSTM with appropriate transformation schemes generates the best synthetic data utility for classification. Nevertheless, with the increase of the number of attributes, the performance advantage of LSTM becomes less significant.

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

7.1.2 *Evaluation on GAN Training*

We evaluate the *robustness* of MLP-based and LSTM-based generator wrt. hyper parameter settings. Given a group of parameters, 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 results on datasets `Adult` and `CovType`. Note that we find similar trends on other datasets, and include result in [24]. 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. After sampling records from inferior synthetic table snapshots, we find the reason is *mode collapse*: generator G only produces nearly duplicated samples, rather than outputting diverse synthetic records. MLP-based generator is robust against various hyper parameter settings, and 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.

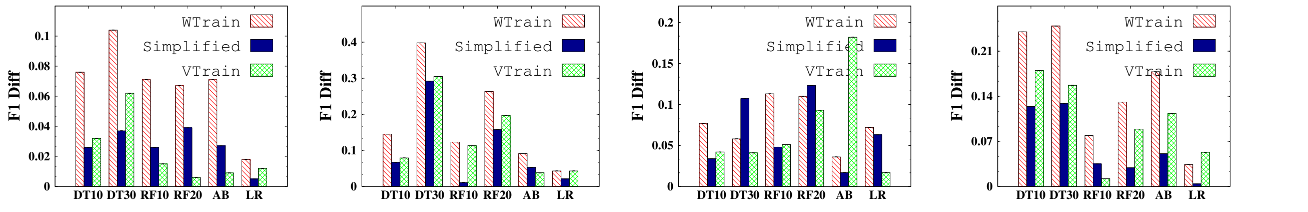
We also examine the following training strategies to alleviate mode collapse: (i) VTrain (with KL divergence), (ii) Wasserstein GAN training (WTrain) and (iii) VTrain with simplified discriminator D (Simplified). As shown in Figure 5, Wasserstein GAN does not have advantage over vanilla GAN training, which is different from the image synthesis scenarios. Simplified achieves better performance than VTrain. For example, on the `Adult` dataset, Simplified reduces the F1 difference compared with VTrain on most classifiers. We also report a result of Simplified against various hyper-parameters in [24] and find it more robust in avoiding mode collapse. The reason is that Simplified makes D not trained too well, and thus avoids the chance of gradient disappearance of generator G.

Finding 3: Vanilla GAN training with simplified discriminator is shown effective to alleviate mode collapse, and outperforms Wasserstein GAN training in preserving data utility.

7.1.3 *Evaluation on Conditional GAN*

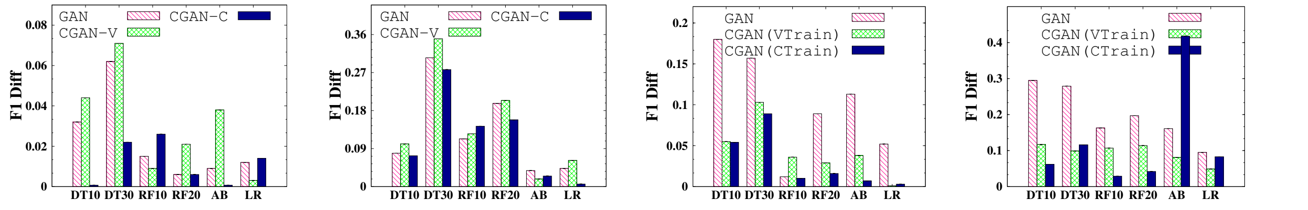
This section investigates if conditional GAN is helpful to address the challenge of imbalance label distribution. We compare the original GAN, conditional GAN trained by random data sampling and conditional GAN trained by label-aware data sampling, which are denoted by `VGAN`, `CGAN-V` and `CGAN-C` respectively, on the skew datasets `Adult`, `CovType`, `Census` and `Anuran`. As shown in Figure 6, `CGAN-V` gains very limited improvements over `VGAN`, and sometimes it performs worse than `GAN`. This is because that `VTrain` uses the random strategy to sample each minibatch of real records. Due to the label imbalance, records with minority labels 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 imbalance label distribution and improves the utility of synthetic data.



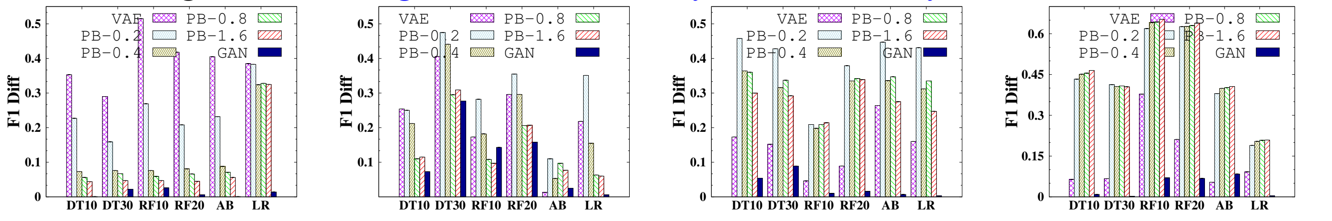
(a) `Adult` dataset. (b) `CovType` dataset. (c) `SAT` dataset. (d) `Census` dataset.

Figure 5: Comparison between strategies to avoid mode collapse. Vanilla GAN training with simplified discriminator is shown effective, and outperforms Wasserstein GAN training in preserving data utility.



(a) `Adult` dataset. (b) `CovType` dataset. (c) `Census` dataset. (d) `Anuran` dataset.

Figure 6: Evaluating conditional GAN on synthetic data utility for classification.



(a) `Adult` dataset. (b) `CovType` dataset. (c) `Census` dataset. (d) `SAT` dataset.

Figure 7: Comparison of different approaches to relational data synthesis on data utility for classification.

Table 4: Effect of size ratio between synthetic and original tables (using DT10 as classifier).

Dataset	Size ratio: $\|\mathcal{T}'\|/\|\mathcal{T}_{\texttt{train}}\|$			
	50%	100%	150%	200%
`Adult`	0.073	0.032	0.028	**0.024**
`CovType`	0.088	0.079	0.117	**0.064**
`SDataNum`	0.007	0.002	0.003	**0.001**
`SDataCat`	0.029	**0.013**	0.018	0.016

7.1.4 *Effect of Sample Size for Synthetic Data*

This section evaluates whether the sample size $|\mathcal{T}'|$ of synthetic table would affect the utility. Table 4 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 sample size, the performance of classifier 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.

7.2 Comparing Data Synthesis Methods

This section compares GAN with `VAE` and `PB`, which are described in Section 6.3. Note that we use the conditional GANas the default setting of GAN.

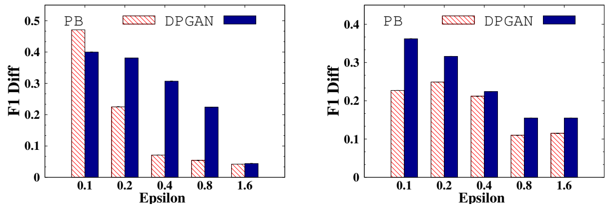
7.2.1 *Evaluation on Synthetic Data Utility*

Figure 7 shows the experimental results on synthetic data utility on our real datasets. First, with the increase of privacy parameter ϵ, the result of `PB` becomes better. This is because ϵ is used to control the privacy level: the larger the ϵ, the lower the privacy level. Second, `VAE` achieves moderate results, but the generated synthetic data is still worse than that synthesized by GAN. This is similar to the case in image synthesis [22]: the images synthesized by VAE is worse than that generated by GAN. This is because the low dimensional latent variable in VAE may not be sufficient to capture the complex relational data.

Our GAN-based framework significantly outperforms `PB` and `VAE` on preserving data utility for classification. For example, the F1 difference achieved by `GAN` is $45-98\%$ and $10-90\%$ smaller than that achieved by `PB` with the lowest

Table 5: Comparison between `GAN` and `PB` on privacy, i.e., Hitting Rate and DCR.

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



(a) `Adult` dataset. (b) `CovType` dataset.

Figure 8: Comparing DPGAN and `PB` on varying privacy levels (using DT10 as classifier).

privacy level ($\epsilon = 1.6$) on the `Adult` and `CovType` datasets respectively. This is mainly attributed to their different data synthesis mechanisms. `PB` aims at approximating a joint multivariate distribution of the original table, which may not perform well if the data distribution is complex. In contrast, GAN utilizes the adversarial training mechanism to optimize generator G. The result shows that the adversarial mechanism is useful for synthesizing relational data.

Finding 5: GAN significantly outperforms `VAE` and `PB` 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 `PB` by an order of magnitude.

7.2.2 *Evaluation on Privacy*

Table 5 compares `GAN` with `PB` on protecting privacy against the risk of re-identification, measured by Hitting Rate and DCR introduced in Section 6.2. Firstly, on the `Adult` dataset, `GAN` achieves lower hitting rate than `PB`. For example, even compared with `PB` with the highest privacy level $\epsilon = 0.1$, `GAN` reduces the hitting rate by 39%. On the `CovType` 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 `CovType` dataset, the hitting rate of `GAN` is higher than that of `PB`. This is because most of the attributes on `CovType` are *numerical* attributes and `PB` discretizes the domain of each numerical attribute into a fixed number of equi-width bins [59, 60], and thus a synthetic numerical value is seldom similar to the original one. Secondly, considering the metric DCR, `GAN` provides comparable overall performance to `PB`, and even outperforms `PB` with moderate privacy levels ($\epsilon = 0.8$ or 1.6). The 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 6: Empirically, the GAN-based data synthesis framework shows better tradeoff between synthetic data 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 8 reports the experimental results on varying privacy level ϵ. We can see that DPGAN cannot beat `PB` at almost all privacy levels on the `Adult` and `CovType` 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 7: The current solution for differential privacy (DP) preserving GAN cannot beat traditional data synthesis methods with DP guarantees.

7.3 Evaluation on Simulated Datasets

This section first evaluates the effect of attribute correlation by using the simulated datasets, and the results on performance and efficiency are reported in Table 6. We can see that LSTM-based generator achieves the best performance on datasets with various degrees of attribute correlation. This shows that the sequence generation mechanism in LSTM is effective and outperforms the other models. In contrast, LSTM is less efficient than MLP and CNN. This is because LSTM uses a more complicated neural network structure to generate each record attribute by attribute.

Table 6: Effect of attribute correlation on data synthesis performance (using DT30 as classifier).

Dataset	F1 Difference			Synthesis Time (Min)		
	CNN	MLP	LSTM	CNN	MLP	LSTM
`SDataNum-0.5`	0.385	0.010	0.005	10	31	67
`SDataNum-0.9`	0.486	0.047	0.020	10	35	60
`SDataCat-0.5`	0.200	0.023	0.014	6	27	60
`SDataCat-0.9`	0.752	0.019	0.012	6	27	50

Table 7: Evaluating neural networks of generator G on synthetic data utility for clustering.

Dataset	CNN	MLP		LSTM	
		sn/ht	gn/ht	sn/ht	gn/ht
`HTRU2`	0.3614	0.0009	0.0019	**0.0003**	0.0035
`Adult`	0.1157	0.0882	0.0124	0.3336	**0.0024**
`CovType`	-	0.0047	0.0015	0.0022	**0.0008**
`Digits`	-	0.0010	0.0021	0.0015	**0.0008**
`Anuran`	-	0.0021	**0.0015**	0.0142	0.0645
`Census`	0.0768	0.0217	0.0153	0.0412	**0.0004**
`SAT`	-	**0.0007**	0.0076	0.0007	0.0029

Table 8: Evaluating neural networks of G on synthetic data utility for AQP.

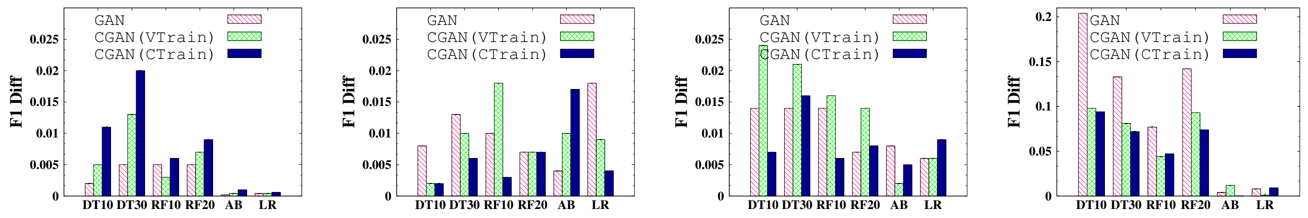
Dataset	CNN	MLP		LSTM	
		sn/ht	gn/ht	sn/ht	gn/ht
`CovType`	-	0.295	0.400	0.609	**0.053**
`Census`	3.499	0.170	**0.167**	0.271	0.204

We also evaluate the effect of label skewness. We set the correlation degree as 0.5 for both `SDataNum` and `SDataCat`, and consider their `balanced` and `skew` settings. As shown in Figure 9, conditional GAN does not improve the data utility, and it sometimes even achieves inferior performance (e.g., on the `SDataNum-balanced` dataset) if label distribution is balanced. In contrast, if label distribution is skew, conditional GAN is helpful for improving the performance.

7.4 Evaluation on Additional Applications

We first investigate the design choices of neural networks and report the results in Table 7 for clustering and Table 8 for AQP. For evaluating AQP, we select the datasets `CovType` and `Census` with more than 100,000 records. Observing from the tables, we find a similar result to that of data utility for classification. The results show that LSTM is effective on capturing the underlying data distribution for the original table, which is also beneficial for clustering and AQP.

We also compare GAN with `VAE` and `PB` on data utility for clustering and AQP. The result on data utility for clustering is reported in Table 9. We can see that GAN outperforms the baselines by 1-2 orders of magnitude. The results show that GAN is very promising in preserving the clustering structure of the original data, e.g., synthesizing similar attributes to the records within in the same group. For AQP, as observed from Table 10, GAN achieves less relative error difference than `VAE` and `PB` on preserving data utility. This is because that GAN, if effectively trained, is more capable of generating synthetic data that well preserves the statis-



(a) `SDataNum`-balance dataset. (b) `SDataNum`-skew dataset. (c) `SDataCat`-balance dataset. (d) `SDataCat`-skew dataset.

Figure 9: Evaluating conditional GAN on synthetic data utility for classification (Simulated data).

Table 9: Comparison of different approaches to relational data synthesis on data utility for clustering.

Dataset	Approaches					
	VAE	PB-0.2	PB-0.4	PB-0.8	PB-1.6	GAN
`HTRU2`	0.0160	0.1769	0.13904	0.0594	0.0331	**0.0007**
`CovType`	0.0089	0.0227	0.0121	0.0071	0.0031	**0.0018**
`Adult`	0.0891	0.0892	0.0959	0.0729	0.0494	**0.0015**
`Digits`	0.0425	0.2025	0.1839	0.1749	0.1545	**0.0008**
`Anuran`	0.2184	0.2989	0.2170	0.1505	0.1617	**0.0020**
`Census`	0.0010	0.0189	0.0101	0.0011	0.0112	**0.0004**
`SAT`	0.4891	0.2451	0.2277	0.2289	0.2279	**0.0007**

Table 10: Comparison of different approaches to relational data synthesis on data utility by AQP.

Dataset	Approaches					
	`VAE`	PB-0.2	PB-0.4	PB-0.8	PB-1.6	GAN
`CovType`	0.251	0.201	0.113	0.183	0.108	**0.015**
`Census`	0.469	2.348	1.262	0.786	0.767	**0.240**
`Bing`	0.632	0.830	0.805	0.783	0.761	**0.422**

tical properties of the original table. Thus, the synthetic data could answer the query workload with less errors. We also notice that, on the AQP benchmarking dataset `Bing`, `VAE` achieves comparable results with GAN, i.e., 0.632 vs. 0.422 on relative error difference. The results show that `VAE` may also be promising for supporting AQP, considering it may be more easy and efficient to train than GAN. Some existing work [52] studies more sophisticated techniques to optimize `VAE`, such as partitioning the data and using multiple VAE models, adding rejection criteria for data sampling, etc. We will leave a more thorough comparison with such new techniques in the future work.

Finding 8: GAN is also very promising for preserving the utility of original data for supporting the applications of clustering and AQP.

8. CONCLUSION & FUTURE DIRECTION

In this paper, we have conducted a 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. 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 is very promising for relational data synthesis. It generates synthetic data with very good utility on classification, clustering and AQP (*Findings 5 and 8*). Moreover, it also achieves competitive performance on protecting privacy against the risk of re-identification (*Finding 6*). However, GAN has limitations on providing provable privacy protection: the current solution cannot produce superior data utility when preserving differential privacy (*Finding 7*).

Neural Network Selection. For ordinary users with limited knowledge on deep learning, we suggest to use MLP to realize GAN, as MLP is more robust and can achieve moderate results without parameter tuning (*Finding 2*). For expert users who want to spend sufficient efforts to finetune parameters, we recommend LSTM that can achieve the best performance (*Finding 1*), given proper training strategies as discussed below, and data transformation schemes.

Model Training Strategy. We provide guidelines to users on how to train GAN models. To avoid mode collapse, we introduce solutions to boost model training, including adding KL divergence in the loss function for warm-up and using simplified discriminator to avoid gradient vanishing in generator (*Finding 3*). We leverage conditional GAN for datasets with imbalanced data distribution (*Finding 4*).

Relational Data Representation. Data transformation that converts original records to recognized input of GAN does affect the overall performance, which shows that representation of relational data is 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 [54, 30], 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 attribute correlations. LSTM achieves good performance as its sequence generation mechanism can *implicitly* capture attribute correlations. The DB community has long studied how to model attribute correlations *explicitly* by providing solutions like functional dependency [48, 13]. Despite some preliminary attempt [16], 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 studies synthetic data utility for training classifiers, evaluating clustering algorithms and supporting AQP. However, relational data synthesis should support a variety of applications, including ML tasks over time-series data and data synthesis for supporting AQP with theoretical bounds.

9. REFERENCES

[1] Adult data set. `https://archive.ics.uci.edu/ml/datasets/Adult`.

[2] Anuran calls (mfccs) data set. `http://archive.ics.uci.edu/ml/datasets/Anuran+Calls+%28MFCCs%29`.

[3] Census-income (kdd) data set. `http://archive.ics.uci.edu/ml/datasets/Census-Income+(KDD)`.

[4] Covertype data set. `http://archive.ics.uci.edu/ml/datasets/covertype`.

[5] Htru2 data set. `http://archive.ics.uci.edu/ml/datasets/HTRU2`.

[6] Pen-based recognition of handwritten digits data set. `https://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits`.

[7] Statlog (landsat satellite) data set. `https://archive.ics.uci.edu/ml/datasets/Statlog+%28Landsat+Satellite%29`.

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

[9] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In *ICLR*. OpenReview.net, 2017.

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

[11] 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.

[12] 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.

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

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

[15] S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: No silver bullet. In S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, and D. Suciu, editors, *SIGMOD*, pages 511–519. ACM, 2017.

[16] 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.

[17] 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.

[18] 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.

[19] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Found. Trends Databases*, 4(1-3):1–294, 2012.

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

[21] 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.

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

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

[24] J. Fan, T. Liu, G. Li, J. Chen, Y. Shen, and X. Du. Relation data synthesis using generative adversarial network: A design space exploration. In *Technical Report*, 2020. `https://github.com/ruclty/Daisy/blob/master/daisy.pdf`.

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

[26] 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.

[27] A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.

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

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

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

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

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

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

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

[35] 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.

[36] K. Li, Y. Zhang, G. Li, W. Tao, and Y. Yan. Bounded approximate query processing. *IEEE Trans. Knowl. Data Eng.*, 31(12):2262–2276, 2019.

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

[38] 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.

[39] 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.

[40] 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.

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

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

[43] 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.

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

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

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

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

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

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

[50] 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.

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

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

[53] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Trans. Knowl. Data Eng.*, 23(8):1200–1214, 2011.

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

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

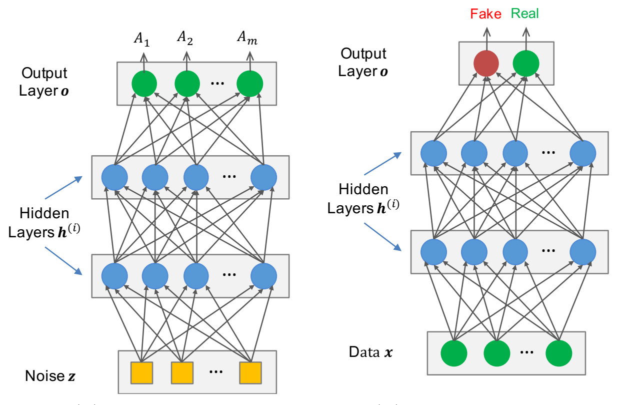
[56] L. Xu and K. Veeramachaneni. Synthesizing tabular data using generative adversarial networks. *CoRR*, abs/1811.11264, 2018.

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

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

[59] 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.

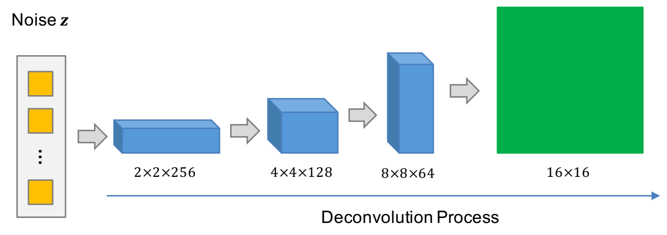
[60] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. *ACM Trans. Database Syst.*, 42(4):25:1–25:41, 2017.



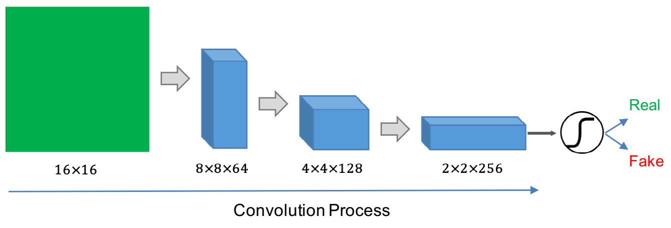
A_1
A_2
A_m
Output Layer $\boldsymbol{o}$
Hidden Layers $\boldsymbol{h}^{(i)}$
Noise $\boldsymbol{z}$
Fake
Real
Output Layer $\boldsymbol{o}$
Hidden Layers $\boldsymbol{h}^{(i)}$
Data $\boldsymbol{x}$

(a) Generator. (b) Discriminator.

Figure 11: GAN module implemented by MLP.



(a) Generator.



(b) Discriminator.

Figure 10: GAN module implemented by CNN.

APPENDIX

A. DETAILED GAN MODEL DESIGN

This section presents more details of our GAN model design. We first present the design of neural network architectures in Section A.1 and then provide the pseudo-codes of the training algorithms in Section A.2.

A.1 Neural Network Architectures

A.1.1 CNN: convolutional neural networks.

CNN is utilized in the existing works for relational data synthesis [16, 43], which is inspired by the well-known DC-GAN [47], as illustrated in Figure 10. One characteristic of these works is that they use a *matrix* instead of a vector to represent a real/fake sample $\boldsymbol{t}$. To this end, they use ordinal encoding and simple normalization to respectively preprocess categorical and numerical attributes. Then, they convert the preprocessed records into a square matrix padded with zeros. For example, consider the first record in our example table in Figure 1. The record is firstly preprocessed into $(-0.2, 0, 0, 0, 0)$ using ordinal encoding and simple normalization. Then, it is converted into a 3×3 matrix with four values are padded with zeros. Based on this, the GAN model can be trained using the converted square matrices. Generator G takes as input a prior noise $\boldsymbol{z}$, which is denoted by $\boldsymbol{h}_g^0$, and uses L de-convolution 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.,

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

where `DeConv` is de-convolution function. Figure 10(a) illustrates a de-convolution process that converts $\boldsymbol{z}$ to a 16×16 matrix that represents a synthetic sample.

Discriminator D, as shown in Figure 10(b), takes as input a real/fake sample $\boldsymbol{t}$ in matrix form, which is denoted by $\boldsymbol{h}_d^0$. It applies L convolution layers $\{\boldsymbol{h}_d^l\}$ to convert $\boldsymbol{t}$ to a probability indicating how likely $\boldsymbol{t}$ is real, i.e.,

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

where `Conv` is a convolution function.

A.1.2 MLP: fully connected neural networks.

MLP is used in the existing works for relational data synthesis [18, 55]. Figure 11 provides generator G and discriminator D realized by MLP. 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

$$\boldsymbol{h}^{l+1} = \phi\big(\texttt{BN}(\texttt{FC}_{|\boldsymbol{h}^l| \to |\boldsymbol{h}^{l+1}|}(\boldsymbol{h}^l))\big), \tag{7}$$

where $\texttt{FC}_{|\boldsymbol{h}^l| \to |\boldsymbol{h}^{l+1}|}(\boldsymbol{h}^l) = \boldsymbol{W}^l \boldsymbol{h}^l + \boldsymbol{b}^l$ 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 [29].

The challenge here is how to make the output layer in G *attribute-aware*. More formally, G needs to output a synthetic sample $\boldsymbol{t} = \boldsymbol{t}_1 \oplus \boldsymbol{t}_2 \oplus \ldots \oplus \boldsymbol{t}_m$, where $\boldsymbol{t}_j$ corresponds to the j-th attribute. Note that, for simplicity, we also use notation $\boldsymbol{t}$ to represent fake samples if the context is clear. We propose to generate each attribute vector $\boldsymbol{t}_j$ depending on the transformation method on the corresponding attribute $\mathcal{T}[j]$, i.e.,

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

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 GMM-based normalization, we first use $\texttt{tanh}(\texttt{FC}_{|\boldsymbol{h}^L| \to 1}(\boldsymbol{h}^L)$ to generate v_{gmm} and then use $\texttt{softmax}(\texttt{FC}_{|\boldsymbol{h}^L| \to |\boldsymbol{t}_j|-1}(\boldsymbol{h}^L))$ to generate a one-hot vector indicating which component v_{gmm} belongs to. After generating $\{\boldsymbol{t}_j\}$ for all attributes, we concatenate them to obtain $\boldsymbol{t}$ as a synthetic sample.

Figure 11(b) shows the NN structure of our discriminator D. 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.

```
Algorithm 1: VTrain ($m$, $\alpha_d$, $\alpha_g$, $T$)
  Input: $m$: batch size; $\alpha_d$: learning rate of $D$; $\alpha_g$: learning
         rate of $G$; $T$: number of training iterations
  Output: $G$: Generator; $D$: Discriminator
1 Initialize parameters $\theta_d^{(0)}$ for $D$ and $\theta_g^{(0)}$ for $G$
2 for training iteration $t = 1, 2, \ldots, T$ do
      /* Training discriminator $D$                          */
3     Sample $m$ noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ from noise prior $p_z(\boldsymbol{z})$
4     Sample $m$ samples $\{\boldsymbol{t}^{(i)}\}_{i=1}^{m}$ from real data $p_{data}(\boldsymbol{t})$
5     $\bar{g}_1 \leftarrow \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(\boldsymbol{t}^{(i)}) + \log(1 - D(G(\boldsymbol{z}^{(i)})))]$
6     $\theta_d^{(t)} \leftarrow \theta_d^{(t-1)} + \alpha_d \cdot \texttt{Adam}(\theta_d^{(t-1)}, \bar{g}_1)$
      /* Training generator $G$                              */
7     Sample $m$ noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ from noise prior $p_z(\boldsymbol{z})$
8     $\bar{g}_2 \leftarrow \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(\boldsymbol{z}^{(i)})))$
9     $\theta_g^{(t)} \leftarrow \theta_g^{(t-1)} - \alpha_g \cdot \texttt{Adam}(\theta_g^{(t-1)}, \bar{g}_2)$
10 return $G$, $D$
```

```
Algorithm 2: WTrain ($m$, $\alpha_d$, $\alpha_g$, $T_d$, $T_g$, $c_p$)
  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
1 Initialize parameters $\theta_d^{(0)}$ for $D$ and $\theta_g^{(0)}$ for $G$
2 for training iteration $t_1 = 1, 2, \ldots, T_g$ do
      /* Using $T_d$ iterations to train $D$                 */
3     for training iteration $t_2 = 1, 2, \ldots, T_d$ do
4         Sample noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ from noise prior
          $p_z(\boldsymbol{z})$
5         Sample samples $\{\boldsymbol{t}^{(i)}\}_{i=1}^{m}$ from real data $p_{data}(\boldsymbol{t})$
6         $\bar{g}_1 \leftarrow \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [D(\boldsymbol{t}^{(i)}) - D(G(\boldsymbol{z}^{(i)}))]$
7         $\theta_d^{(t_2)} \leftarrow \theta_d^{(t_2-1)} + \alpha_d \cdot \texttt{RMSProp}(\theta_d^{(t_2-1)}, \bar{g}_1)$
8         $\theta_d^{(t_2)} \leftarrow \texttt{clip}(\theta_d^{(t_2)}, -c_p, c_p)$
      /* Training generator $G$                              */
9     Sample noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m}$ from noise prior $p_z(\boldsymbol{z})$
10    $\bar{g}_2 \leftarrow -\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} D(G(\boldsymbol{z}^{(i)}))$
11    $\theta_g^{(t_1)} \leftarrow \theta_g^{(t_1-1)} - \alpha_g \cdot \texttt{RMSProp}(\theta_g^{(t_1-1)}, \bar{g}_2)$
12 return $G$, $D$
```

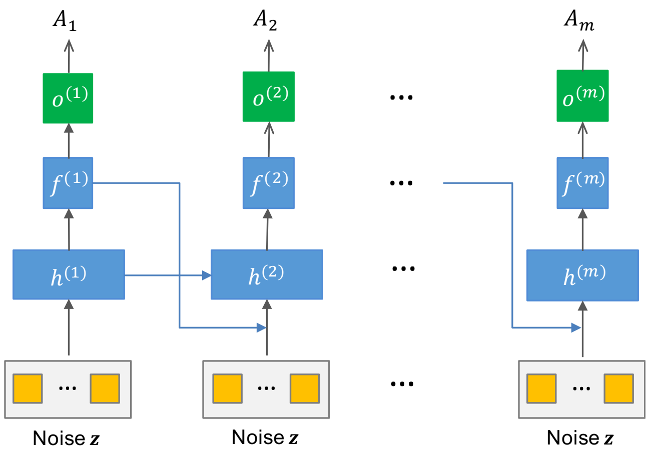


Figure 12: Generator implemented by LSTM.

A.1.3 LSTM: recurrent neural networks.

Existing work also utilizes LSTM, a representative variant of RNN, to realize G [56]. 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 $\boldsymbol{t}_j$, as illustrated in Figure 12. 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

$$\boldsymbol{h}^{j+1} = \texttt{LSTMCell}(\boldsymbol{z}, \boldsymbol{f}^j, \boldsymbol{h}^j),$$
$$\boldsymbol{f}^{j+1} = \texttt{tanh}(\texttt{FC}_{|\boldsymbol{h}^{j+1})| \to |\boldsymbol{f}^{j+1})|}(\boldsymbol{h}^{j+1})),$$

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

Next, we compute attribute $\boldsymbol{t}_j$ 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.

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

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 $v_{\text{gmm}} = \texttt{tanh}(\texttt{FC}_{|\boldsymbol{f}^{j_1}| \to 1}(\boldsymbol{f}^{j_1}))$, while the second timestep j_2 generates a vector that indicates which GMM component v_{gmm} comes from, i.e., $\texttt{softmax}(\texttt{FC}_{|\boldsymbol{f}^{j_2}| \to |\boldsymbol{t}_j - 1|}(\boldsymbol{f}^{j_2}))$. Then, we concatenate these two parts to compose $\boldsymbol{t}_j$.

Note that we can use a typical *sequence-to-one* LSTM [51] to realize the discriminator D.

A.2 Training Algorithms

This section presents the pseudo-code of the training algorithms introduced in Sections 5.2, 5.3 and 5.4.

A.2.1 Vanilla GAN Training

We apply the vanilla GAN training algorithm [26] (VTrain) to iteratively optimize parameters θ_d in D and θ_g in G.

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

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

where m is the minibatch size and α_d (α_g) is learning rate of D (G).

Algorithm 1 presents the pseudo-code of the vanilla training algorithm [26]. It takes as input size m of minibatch, learning rates α_d and α_g) of discriminator D and generator G, and number T of training iterations, and iteratively optimize parameters θ_d in D and θ_g in 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 [33]. Second, it fixes D and trains G by sampling another set of noise samples and updating parameters θ_g.

A.2.2 Wasserstein GAN Training

We also evaluate Wasserstein GAN [10] for training our data synthesizer (WTrain). Different from the original GAN, Wasserstein GAN removes the `sigmoid` function of D

Algorithm 3: CTrain (m, α_d, α_g, T, Ω)

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

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

```
Input: m: batch size; α_d: learning rate of D; α_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; σ_n:
       noise scale
Output: G: Generator; D: Discriminator
1  Initialize parameters $\theta_d^{(0)}$ for $D$ and $\theta_g^{(0)}$ for $G$
2  for training iteration $t_1 = 1, 2, \ldots, T_g$ do
       /* Using T_d iterations to train D */
3      for training iteration $t_2 = 1, 2, \ldots, T_d$ do
4          Sample noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^m$ from prior $p_z(\boldsymbol{z})$
5          Sample samples $\{\boldsymbol{t}^{(i)}\}_{i=1}^m$ from real data $p_{data}(\boldsymbol{t})$
6          for each $i$ do
7              $g_1(\boldsymbol{t}^{(i)}, \boldsymbol{z}^{(i)}) \leftarrow \nabla_{\theta_d}[D(\boldsymbol{t}^{(i)}) - D(G(\boldsymbol{z}^{(i)}))]$
8          $\bar{g}_1 \leftarrow \frac{1}{m}(\sum_{i=1}^m g_1(\boldsymbol{t}^{(i)}, \boldsymbol{z}^{(i)}) + N(0, \sigma_n^2 c_g^2 I))$
9          $\theta_d^{(t_2)} \leftarrow \theta_d^{(t_2-1)} + \alpha_d \cdot \texttt{RMSProp}(\theta_d^{(t_2-1)}, \bar{g}_1)$
10         $\theta_d^{(t_2)} \leftarrow \texttt{clip}(\theta_d^{(t_2)}, -c_p, c_p)$
       /* Training generator G */
11     Sample noise samples $\{\boldsymbol{z}^{(i)}\}_{i=1}^m$ from prior $p_z(\boldsymbol{z})$
12     $\bar{g}_2 \leftarrow -\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m D(G(\boldsymbol{z}^{(i)}))$
13     $\theta_g^{(t_1)} \leftarrow \theta_g^{(t_1-1)} - \alpha_g \cdot \texttt{RMSProp}(\theta_g^{(t_1-1)}, \bar{g}_2)$
14 return G, D ;
```

and changes the gradient optimizer from `Adam` to `RMSProp`. It uses the loss functions of D and G as

$$\mathcal{L}_D = -\mathbb{E}_{\boldsymbol{t}\sim p_{data}(\boldsymbol{t})}[D(\boldsymbol{t})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))]$$
$$L_G = -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))]. \quad (8)$$

Algorithm 2 presents the training algorithm in Wasserstein GAN [10]. 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 shown in Equation (8). Algorithm 2 takes as input size m of mini-batch, learning rates α_d and α_g) of discriminator D and generator G, and number training iterations T_d and T_g and a clipping parameter c_p. 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.

A.2.3 *Conditional GAN Training*

Algorithm 3 presents the algorithm for training conditional GAN. Basically, it follows the framework of the vanilla GAN training in Algorithm 1 with a minor modification of *label-aware* sampling. The idea is to avoid that the minority label has insufficient training opportunities due to the highly imbalanced label distribution. 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 records with different labels have "fair" opportunities for training.

A.2.4 *DPGAN Training*

Algorithm 4 presents the training algorithm for DPGAN. Basically, it follows the framework of Wasserstein GAN training in Algorithm 2 with minor modifications (DPTrain). 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 [54] for details of c_g).

B. ADDITIONAL EXPERIMENTS

B.1 Detailed Dataset Information

We describe the detailed information of the eight datasets we used in our experiments.

(1) `HTRU2` **dataset** is a physical dataset that contains 17,898 pulsar candidates collected during the High Time Resolution Universe Survey [5]. This dataset has 8 numerical attributes, which are statistics obtained from the integrated pulse profile and the DM-SNR curve, and a binary label (i.e., pulsar and non-pulsar). The label distribution is balanced.

(2) `Digits` **dataset** contains 10,992 pen-based handwritten digits [6]. Each digit has 16 numerical attributes collected by a pressure sensitive tablet and processed by normalization methods, and a label indicating the gold-standard number from $0-9$. The label distribution is balanced.

(3) `Adult` **dataset** contains 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` [1]. We use attribute `Income` as label and predict whether a person has income larger than $50K$ per year (positive) or not (negative), where the label distribution is skew, i.e., the ratio between positive and negative labels is 0.34.

(4) `CovType` **dataset** contains the information of 116,204 forest records obtained from US Geological Survey (USGS) and US Forest Service (USFS) data [4]. It includes 2 categorical attributes, `Wild-area` and `Soil-type`, and 10 numerical attributes, such as `Elavation` and `Slope`. We use attribute `Cover-type` with 7 distinct values as label and predict 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.

(5) `SAT` dataset consists of the multi-spectral values of pixels in 3x3 neighborhoods in a satellite image [7]. It has 36 numerical attributes that represent the values in the four spectral bands of the 9 pixels in a neighborhood, and uses a label with 7 unique values indicating the type of the central pixel. The label distribution is balanced in the dataset.
(6) `Anuran` dataset a dataset from the life domain for anuran species recognition through their calls [2]. It has 22 numerical attributes, which are derived from the audio records belonging to specimens (individual frogs), and associates a label with 10 unique values that indicates the corresponding species. The label distribution is very skew: there are $3,478$ records with label 2 and 68 with label 9.
(7) `Census` dataset contains weighted census data extracted from the 1994 and 1995 Current Population Surveys [3]. We use demographic and employment variables, i.e., 9 numerical and 30 categorical attributes, as features, and `total-person-income` as label. We remove the records containing null values and then obtain $142,522$ records with very skew label distribution, i.e., 5% records with income larger than $50K$ vs. 95% with income smaller than $50K$.
(8) `Bing` dataset is a Microsoft production workload dataset, which contains the statistics of Bing Search and is used for evaluating AQP [36]. We sample $500,000$ records with 23 categorical and 7 numerical attributes. As the dataset does not have any attribute used as label, we only use the dataset for evaluating performance of data synthesis on AQP.

B.2 Additional Evaluation on Mode Collapse

B.3 Evaluating LSTM Discriminator

This section evaluates LSTM-based discriminator D for GAN-based relational data synthesis on the `Adult` dataset. Note that we use a typical *sequence-to-one* LSTM [51] to realize D. Table 11 reports the experimental results. We can see that, compared with MLP-based discriminator D reported in Table 3(a), the F1 difference is significantly higher. Considering classifier DT10 as an example, the F1 difference increases by $18 - 416\%$ when changing MLP to LSTM for realizing discriminator. We also find similar results in other datasets. Therefore, we use MLP to implement discriminator in our experiments reported in Section 7.

B.4 Synthetic Data Distribution

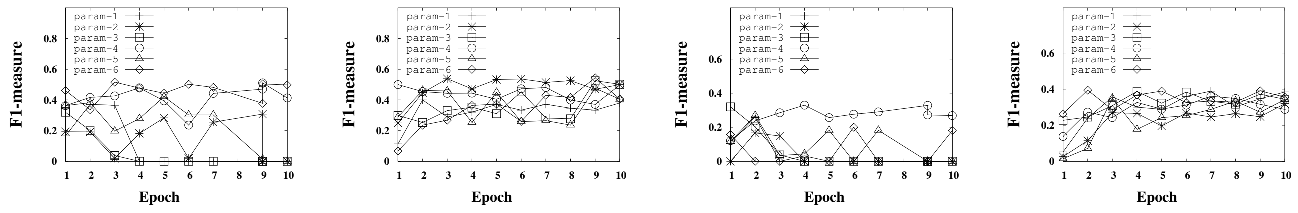
Figure 20 shows the distribution for numerical attributes using the violin plots on the two real datasets, where we fix categorical attribute encoding as one-hot and focus on comparing simple (`sn`) and GMM-based normalization (`gn`). As observed, LSTM equipped with `gn` can generate attributes having the most approximate distribution to their counterpart real attributes, and it is remarkably effective when the attribute has *multi-modal* distribution. We have analyzed the reason in Section 7. Figure 21 shows the distribution for categorical attributes. We can see that one-hot is significantly better than ordinal embedding.

B.4.1 Evaluation on Data Transformation

We also find that data transformation in preprocessing does affect the overall utility of synthetic data. Observed from Tables 3(a), 3(b), **??** and **??**, 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 `CovType`, and we include their results in [24].

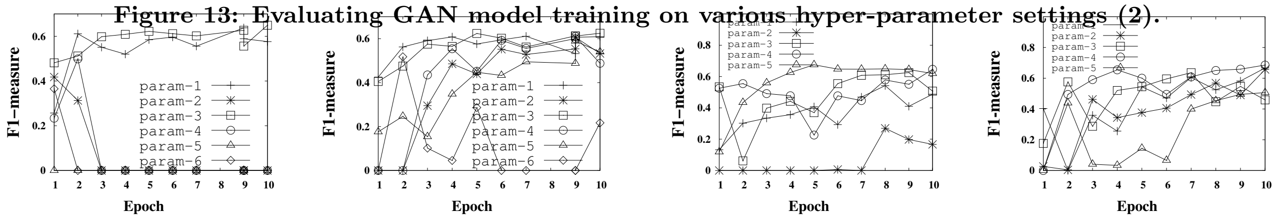
Figure 17 shows the distribution for numerical attributes using the violin plots. LSTM with `gn` can generate the attribute having the most approximate distribution to their counterpart real attribute, and it is remarkably effective for the attribute with *multi-modal* distribution. This is attributed to the Gaussian Mixture model used in this method, which is more powerful to represent multi-modal attributes. Moreover, it also outperforms MLP with `gn`. 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 18 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: Data transformation does affect overall 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.



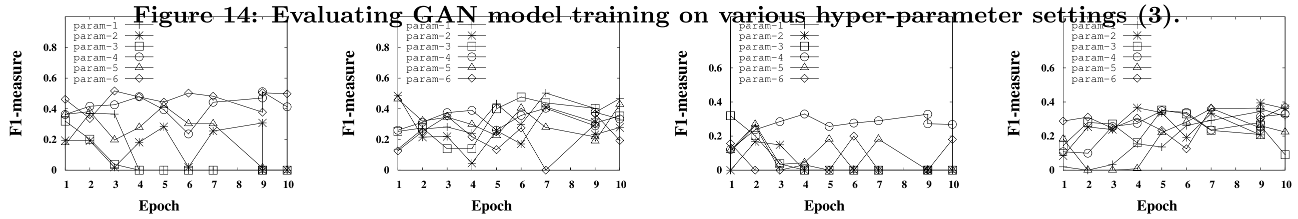
(a) LSTM-based G (SAT). (b) MLP-based G (SAT). (c) LSTM-based G (Census). (d) MLP-based G (Census).

Figure 13: Evaluating GAN model training on various hyper-parameter settings (2).



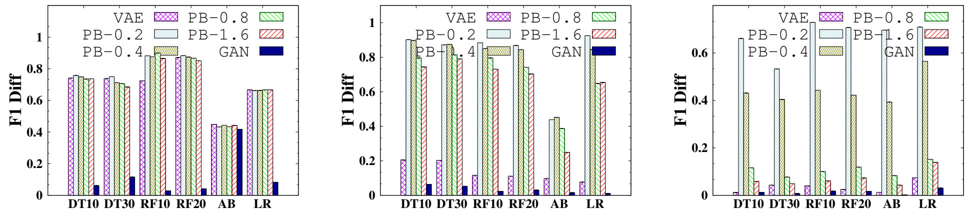
(a) Normal D (Adult). (b) Simplified D (Adult). (c) Normal D (CovType). (d) Simplified D (CovType).

Figure 14: Evaluating GAN model training on various hyper-parameter settings (3).



(a) Normal D (SAT). (b) Simplified D (SAT). (c) Normal D (Census). (d) Simplified D (Census).

Figure 15: Evaluating GAN model training on various hyper-parameter settings (4).



(a) Anuran dataset. (b) Digits dataset. (c) HTRU2 dataset.

Figure 16: Comparison of different approaches to relational data synthesis on data utility for classification.

Table 11: Evaluating LSTM-based discriminator on synthetic data utility (Adult dataset).

Classifier	MLP				LSTM			
	sn +od	sn +ht	gn +od	gn +ht	sn +od	sn +ht	gn +od	gn +ht
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

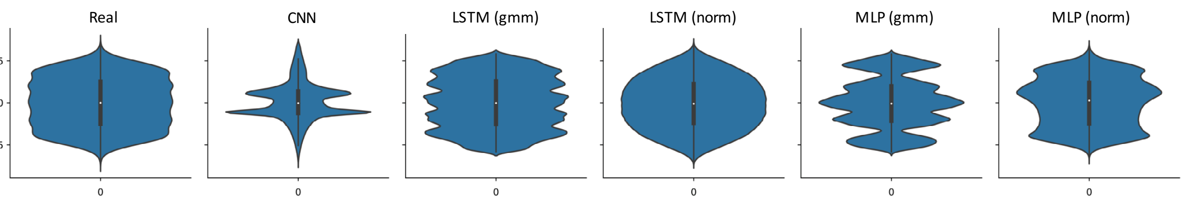


Figure 17: Evaluating value distribution of synthetic numerical attributes (SDataNum).

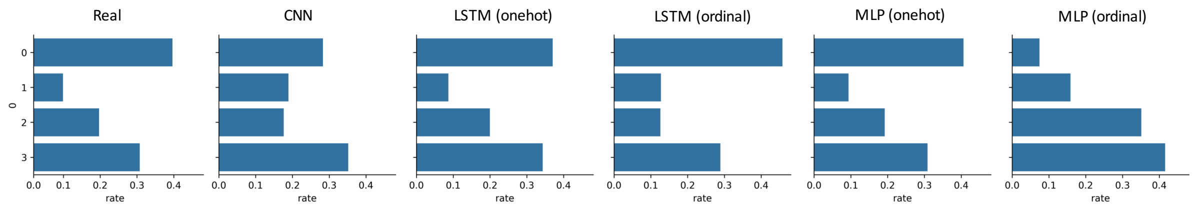
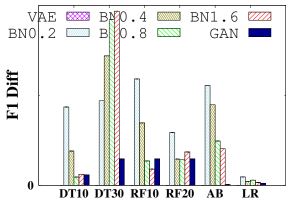
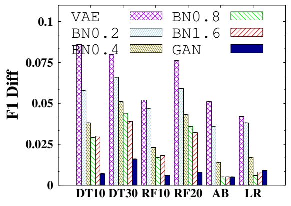


Figure 18: Evaluating value distribution of synthetic categorical attributes (`SDataCat`).



(a) `SDataNum` dataset.

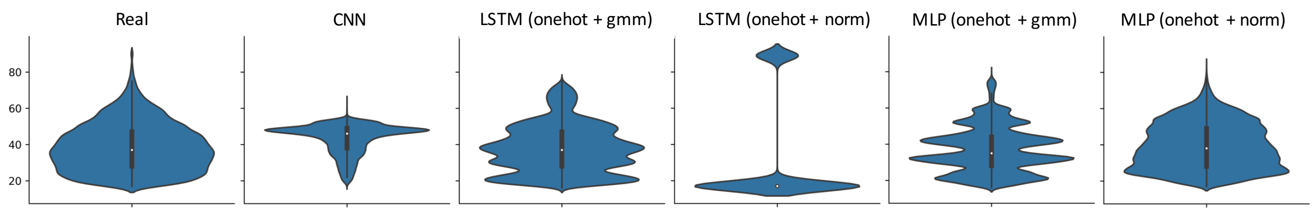


(b) `SDataCat` dataset.

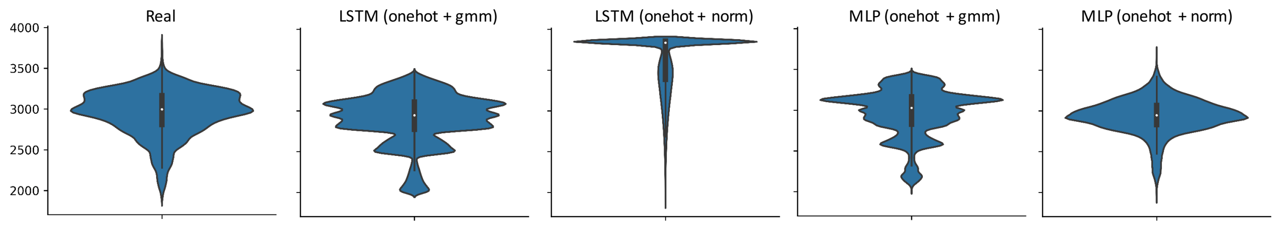
Figure 19: Comparison of different approaches to relational data synthesis on data utility by Classification (1).

Table 12: Comparison of different approaches to relational data synthesis on data utility for AQP (selectivity ≥ 0.1).

Dataset	**Approaches**					
	VAE	BN0.2	BN0.4	BN0.8	BN1.6	GAN
`CovType`	0.059	0.038	0.037	0.037	0.032	**0.012**
`Census`	0.078	0.160	0.151	0.148	0.142	**0.044**
`Bing`	0.558	2.069	1.218	0.596	0.586	**0.335**

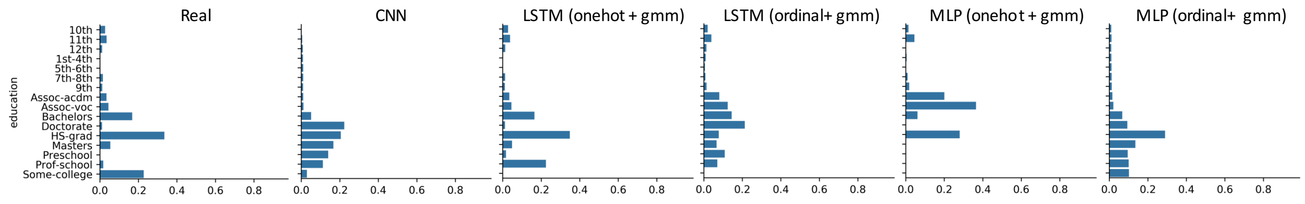


(a) Attribute `age` on the `Adult` dataset.

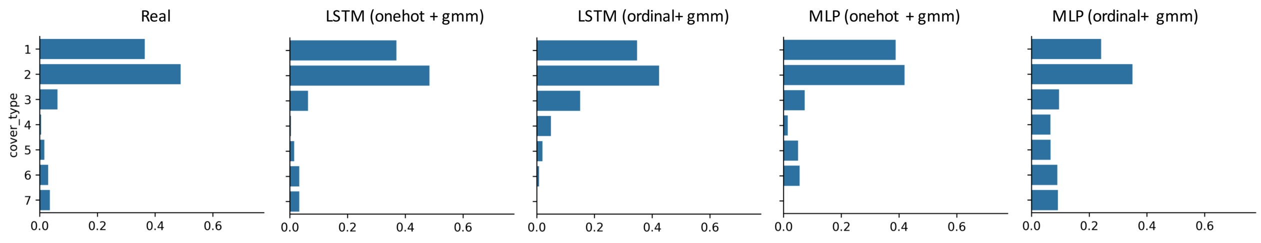


(b) Attribute `elevation` on the `CovType` dataset.

Figure 20: Evaluating value distribution of numerical attributes synthesized by different models.



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



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

Figure 21: Evaluating value distribution of categorical attributes synthesized by different models.