Noname manuscript No.
(will be inserted by the editor)

Tabular Data Synthesis with Generative Adversarial Networks: Design Space and Optimizations

Tongyu Liu · Ju Fan · Guoliang Li · Nan Tang · Xiaoyong Du

Received: date / Accepted: date

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. To address this problem, the database community and machine learning community have recently studied a new problem of tabular data synthesis using generative adversarial networks (GAN) and proposed various algorithms. However, a comprehensive comparison between GAN-based methods and conventional approaches is still lacking, making it unclear why and how GANs can outperform conventional approaches in synthesizing tabular data. Moreover, it is difficult for practitioners to understand which components are necessary when building a GAN model for tabular data synthesis. To bridge this gap, we conduct a comprehensive experimental study that investigates applying GAN to tabular 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 provide optimization techniques to handle difficulties in training GAN in practice. We conduct extensive experiments to explore the design space, comparing with traditional data synthesis approaches. Through extensive experiments, we find that GAN is very promising for tabular data synthesis, and provide guidance for selecting appropriate design choices. We also point out limitations of GAN and identify future research directions. We make all code and datasets public for future research.

Tongyu Liu
Renmin University of China, Beijing 100872, China
E-mail: ltyzzz@ruc.edu.cn

Ju Fan
Renmin University of China, Beijing 100872, China
E-mail: fanj@ruc.edu.cn

Guoliang Li
Tsinghua University, Beijing 100084, China
E-mail: liguoliang@tsinghua.edu.cn

Nan Tang
Qatar Computing Research Institute,
E-mail: ntang@hbku.edu.qa

Xiaoyong Du
Renmin University of China, Beijing 100872, China
E-mail: duyong@ruc.edu.cn

1 Introduction

The unprecedented scale of big data has become an indispensable driving-force of the remarkable development of data science and machine learning in the past decades. However, the tremendous amount of big data does not automatically lead to easy access. The difficulty in data access is still one of the top barriers of many data scientists [37]. 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, e.g., predicting unexpected readmission to reduce operation cost [50]. However, the data sharing must be carefully reviewed by its legal department and institutional review boards to avoid disclosure of patient privacy, which usually takes months without guarantee of approval [36].

To address the difficulties, *privacy-preserving data publishing* has been extensively studied to provide a safer way for data sharing [17,48,24,10,79,80]. However, existing solutions suffer from limitations on effectively balancing privacy and utility of the released data [59]. Therefore, efforts have been made in the database and machine learning communities to apply *generative adversarial networks (GAN)* to tabular data synthesis [19,72,73,59,13,21,53]. The main advantages of GAN are as follows. First, different from the conventional methods [17,48,10,79] 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 [59]. Moreover, the adversarial learning mechanism of GAN enables the synthetic data to effectively preserve utility of the original data for supporting downstream applications, such as classification and clustering.

However, compared with the success of using GAN for image generation [54], GAN-based tabular data synthesis is still in its infancy stage. Despite some very recent attempts [19,72,73,59,13,21,53], as far as we know, a comprehensive comparison between GAN-based methods and conventional approaches is still lacking, making it unclear why and how GANs can outperform conventional approaches in synthesizing tabular data. Moreover, it is difficult for practitioners to understand which components are necessary when building a GAN model for tabular data synthesis.

To address this problem, in this paper, we conduct a benchmarking study that aims to provide guidance on constructing GAN models for tabular data synthesis and evaluating their effectiveness compared to conventional approaches. To this end, we first introduce a general framework that unifies existing solutions for GAN-based tabular data synthesis to help practitioners understand which components are necessary. We categorize existing design solutions for different components within this framework and compare solutions proposed by different works under the same framework. Based on this, we conduct extensive experiments to systemically investigate the following key questions.

First, it remains an unresolved question on how to effectively design GAN for tabular data synthesis. It is worth noting that tabular data has its own characteristics that make the design very challenging. (*i*) Tabular data has mixed data types, including categorical and numerical attributes. (*ii*) Different attributes have correlations. Thus, the state-of-the-art GAN design for image synthesis (e.g., DCGAN [63]) may not perform well for tabular data. We review existing solutions that realize GAN, including neural network design and data transformation, and define a *design space* by providing a categorization of existing solutions. Through exploring the design space, we systemically evaluate the solutions on datasets with various types and provide insightful experimental findings.

The second question is how to develop effective GAN training algorithms. Real-world tabular data synthesis scenarios are very complex that pose many difficulties in training a GAN model. First, a well-recognized obstacle in GAN training is *mode collapse* such that the generator produces records with limited varieties. Second, the performance of GAN heavily relies on the quality and diversity of the real data $\mathcal{T}$, and it is challenging to train GAN with limited data. Third, many real-world datasets have highly imbalanced data distribution, which increases the difficulty of data synthesis, especially for records with minority labels. To tackle these difficulties, we discuss optimization techniques, which are particularly designed for tabular data synthesis, e.g., avoiding mode collapse, data augmentation for limited data, and conditional GAN for tabular data synthesis with imbalanced distributions.

The third question is whether GAN is more helpful than other existing approaches for tabular data synthesis. To answer this question, we consider various baselines, including a representative deep generative model, variational auto-encoder (VAE) [43,65], and the state-of-the-art data synthesis approach using statistical models [79,80]. 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 [26]. Based on the comparison, we analyze the benefits and limitations of applying GAN to tabular data synthesis.

To summarize, we make the following contributions.

(1) We conduct an experimental study for applying GAN to tabular data synthesis. We formally define the problem and review existing approaches (Section 3). We introduce a unified framework and define a design space that summarizes the solutions for realizing GAN (Sections 4), which can help practitioners to easily understand how to design GAN for tabular data synthesis. We develop optimization techniques to handle difficulties in training GAN for real-world tabular data (Section 5).

(2) We empirically conduct a thorough evaluation to explore the design space and compare with the baseline approaches (Section 6). We make all code 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 tabular data synthesis, as it empirically provides better tradeoff between synthetic data utility and privacy. We

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

point out the limitations of GAN-based data synthesis and identify future research directions (Section 8).

This article extends our conference version [29], where the main extensions are summarized as follows.

– We introduce new optimization methods to address two common challenges faced while training GAN models on tabular data: (1) mode collapse that results in nearly duplicated records and (2) limited training data that leads to overfitting of the models. Section 5 presents the methods, and Section 7.2 reports the experimental results.
– We add additional experiments to explore the design space of GAN-based tabular data synthesis, such as adding deep learning classifiers to make results of data utility more comprehensive, evaluating data transformation and model efficiency to consider more evaluation aspects, and comparing Vanilla GAN and DPGAN to understand GAN training (Section 7.1). Moreover, we provide more detailed information of the real datasets used in the paper (Section 6).
– We improve the presentation by re-organizing the paper for readability and adding detailed design of different neural networks and pseudo-code of various GAN training algorithms (Section 4).

2 Related Work

2.1 Synthetic Data Generation

Tabular 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 varieties, existing works utilize copulas [46,61], Bayesian networks [79,80], Gibbs sampling [60] and Fourier decomposition [14]. Synopses-based approaches, such as wavelets and multi-dimensional sketches, build compact data summary for massive data [22,70], 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 tabular 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 [30] and variational autoencoders (VAE) [69], , while more attentions are paid on generative adversarial networks (GAN) [19,72,73,59,13,21,53,74,52,45, 39]. However, despite the aforementioned attempts on GAN-based tabular 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 tabular data synthesis.

Recently, some studies introduce *diffusion-based* generative models for tabular data synthesis [41,40,44], and these models also show good performance. Like GANs, diffusion models aim to transform random noise into data that follows the real data distribution. On the other hand, diffusion models differ in that they first define a Markov process to construct a mapping from the real data distribution to the noise distribution and then learn a model to approximate the reverse process.

2.2 Generative Adversarial Networks (GAN)

Generative adversarial networks (GAN) [31,54], are a kind of deep generative models, which have achieved breakthroughs in many areas, such as image generation [63,57,20], and sequence generation [77,75]. 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 minmax game with value function $V(G,D)$, i.e.,

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

where p_{data} is the distribution of the real samples transformed from our tabular table $\mathcal{T}$ and p_z is the distribution of the input noise $\mathbf{z}$. Minibatch stochastic gradient descent algorithms can be applied to optimizing parameters θ_g and θ_d in G and D respectively. Over the past few years, many variants of GAN architectures have been proposed. Radford et al. propose deep convolutional generative adversarial networks [63], introducing convolutional networks (CNN) to the framework. Mirza and Osindero add extra information to GANs by feeding conditions to both generator and the discriminator, which improves GAN's ability to model complex data distribution [57]. Chen et al. introduce interpretable

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

Fig. 1: An example of tabular data, with a numerical attribute `age`, three categorical attributes `gender`, `education` and `occupation` and a label `income`.

representation learning by proposing information maximizing GANs [20]. Lee et al. combine flow-based models with GANs to create an invertible generator [45]. Kim et al. enhance data utility by designing the generator and discriminator based on neural ordinary differential equations [39]. While a full survey of the approaches to realizing GAN is beyond the scope of this paper, we refer the reader to a very recent survey [54].

3 Tabular Data Synthesis with GAN

3.1 Tabular Data Synthesis

This paper focuses on a tabular data $\mathcal{T}$ 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 highly depends on the specific need of the synthetic data for downstream applications. This paper focuses on the specific need on using fake tables to train machine learning (ML) models, which is commonly considered by recent works [19, 72, 73, 59, 13, 21, 53, 27]. 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, inspired by the *TSTR* [27] method, we can train a classifier f' on the synthetic table $\mathcal{T}'$ and evaluate the classifier on the same $\mathcal{T}_{\texttt{test}}$ to obtain its performance $\texttt{Eval}(f'|\mathcal{T}_{\texttt{test}})$. The utility of $\mathcal{T}'$ is measured by the difference between these two classifiers' performance metrics as:

$$\texttt{Diff}(\mathcal{T}, \mathcal{T}') = \mid \texttt{Eval}(f|\mathcal{T}_{\texttt{test}}) - \texttt{Eval}(f'|\mathcal{T}_{\texttt{test}}) \mid . \quad (2)$$

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. In this paper, we employ two commonly used attacks: membership inference attack [67] and re-identification attack [60]. To perform membership inference attack, we train an attack model to classify whether a given instance is used in the GAN model training. We use the F1-score of the attack model as a metric to evaluate the attack performance. For re-identification attack, we evaluate the attack performance by measuring the likelihood that the original data records can be re-identified by the attacker from the synthetic data, here we use *hitting rate* and *distance to the closest record* (DCR) that have been widely used in the existing works [60, 53, 55] as the metrics.

Remarks. Real-world practice in tabular data synthesis may have various needs on preserving data utility. Besides classification, this paper also considers data utility for the following two applications.

(1) Data utility for clustering. We consider the need on using synthetic data to evaluate a *clustering* algorithm. 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.

(2) Data utility for answering queries. We examine the need on using synthetic data to support approximate query processing (AQP) [18,69]. For example, suppose that a user wants to perform data exploration or visualization on a large dataset. To reduce latency, some work [69] 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* [69], 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 $\texttt{Diff}_{\texttt{AQP}}(\mathcal{T},\mathcal{T}') = \sum_{q\in Q} \texttt{Diff}_{\texttt{AQP}}(\mathcal{T},\mathcal{T}'|q)/|Q|$.

3.2 **TDGAN**: A GAN-Based Synthesis Framework

We introduce a framework **TDGAN** that unifies the existing solutions for applying GAN to tabular data synthesis, as shown in Figure 2. **TDGAN** takes a relational table $\mathcal{T}$ as input and generates a table $\mathcal{T}'$ of synthetic data in three phases.

Phase I - Data Transformation. Tabular data has *mixed* data types, including numerical and categorical attributes. However, GAN models each data tuple as a sample $\boldsymbol{t} \in \mathbb{R}^d$ of numerical values. Thus, this phase to transform a record into such a sample[2]. To address this problem, 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. Attributes in tabular data usually have correlations. It remains challenging to enable the generator to capture such correlations. To this end, 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

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

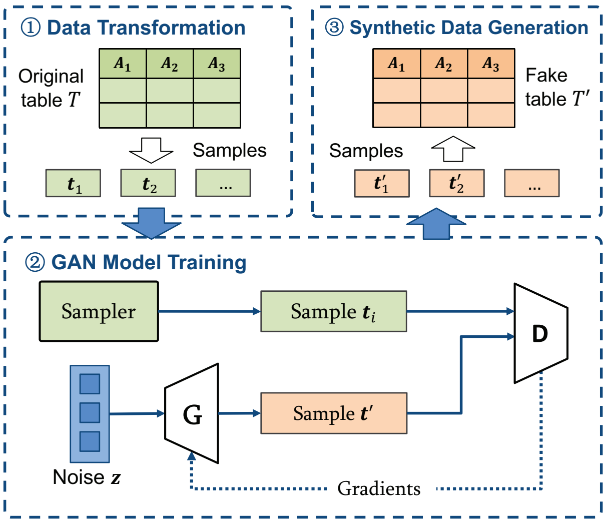


Fig. 2: Overview of data synthesis using GAN. (1) It transforms each record in a tabular data into a sample $\boldsymbol{t} \in \mathbb{R}^d$. (2) It takes the samples as input to train a deep generative model G using the adversarial training framework in GAN. (3) It utilizes the trained G to generate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

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. The key technical issue here is to design effective neural networks for G that can capture correlations among attributes.

Phase III - Synthetic Data Generation. This phase utilizes G, which is well trained previously, 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`, $> 50K$).

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	
GAN Training		(1) Vanilla GAN (2) DPGAN		

Fig. 3: Categorization of design solutions for components in TDGAN: Data Transformation, Neural Networks and GAN Training.

4 Design Space of TDGAN

We provide a categorization of design solutions for each component in **TDGAN**, which forms a design space as summarized in Figure 3.

– For *data transformation*, we examine how to encode categorical attributes to numerical values, and normalize numerical attributes to appropriate ranges that fit neural networks in Section 4.1.
– For *neural networks*, we consider three representative neural network architectures in Section 4.2.
– For *GAN training*, we present a vanilla training algorithm and a differential privacy (DP) preserving algorithm in Section 4.3.

4.1 Data Transformation

Data transformation converts a record t in $\mathcal{T}$ 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. Moreover, as different neural networks have different requirements for the input, sample $\boldsymbol{t}$ can be in the form of either *matrix* or *vector*. Next, we first describe the schemes for both categorical and numerical attributes, and then discuss how to combine multiple attributes. Note that a recent survey summarizes more sophisticated transformation schemes for tabular data [16]. We will explore these schemes as a future work.

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

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

(2) One-hot encoding assigns each category of categorical attribute $\mathcal{T}[j]$ with an integer, starting from 0 to $|\mathcal{T}[j]| - 1$. Then, it represents each category as a *binary vector* with all zero values, except that the index of the integer corresponding to the category is set as one. For example, for `gender`, ordinal encoding and one-hot encoding respectively transform its two values into 0 and 1, and $(1, 0)$ and $(0, 1)$.

Numerical attribute transformation. We normalize values in a numerical attribute to $[-1, 1]$, to enable neural networks in G to generate values in the attribute using `tanh` as an activation function. In particular, we investigate the following two normalization methods.

(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:

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

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

(2) GMM-based normalization. Some studies [72, 73] propose to consider the *multi-modal* distribution of a numerical attribute $\mathcal{T}[j]$, to avoid 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:

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

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

Combining multiple attributes. Once all attributes in t are transformed by the above schemes, we need to combine them together to generate sample $\boldsymbol{t}$.

(1) Matrix-formed samples. For CNN-based neural networks, we follow the method in [59] to convert attributes into a square matrix. For example, a record with 8 attributes is converted into a 3×3 square matrix after

padding one zero. Note that 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.

(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 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 $((0.2, 1, 2), (4, 1, 0), (0, 0, 0))$. When transforming it into a vector-formed sample, we may choose to use one-hot encoding and GMM-based normalization, and obtain $(\underline{-0.7, 0, 1}, \underline{0, 1, 0, 0, 1, 0}, \underline{0, 0, 0, 0, 1}, \underline{0, 1})$, where the underlines indicate different attributes.

4.2 Neural Networks

We present representative neural networks according to the form of input sample $\boldsymbol{t}$. Given matrix-formed samples, we examine Convolutional Neural Networks (CNN) for both generator G and discriminator D, in which G is a deconvolution process and D is a convolution process. Such solution has been proven to be effective for synthetic image generation [63]. For vector-formed samples, we can use fully-connected neural networks (MLP) [32] to realize G that uses multiple layers to transform random noise $\boldsymbol{z}$ to sample $\boldsymbol{t}'$. Alternatively, we can model record synthesis as a *sequence generation* process that generates attributes separately in sequential time-steps. In this case, we may choose some variant of recurrent neural networks, such as long short-term memory (LSTM) networks [33] to realize G.

CNN: convolutional neural networks. Inspired by the success of image synthesis, some apply DCGAN [63], and use Convolutional Neural Networks (CNN) for G and D, in which G is a deconvolution process and D is a convolution process [19,59].

Specifically, generator G takes as input a prior noise $\boldsymbol{z}$, which is also denoted by $\boldsymbol{h}_g^0$. It then uses L deconvolution 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.

Discriminator D takes as input a real/fake sample $\boldsymbol{t}$ in matrix form, 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.

MLP: fully connected neural networks. Following the original GAN [31], some studies [21,72] use multilayer perceptron (MLP) consisting of multiple fully-connected layers. G takes as input noise $\boldsymbol{z}$, which is also denoted by $\boldsymbol{h}^{(0)}$, and utilizes L fully-connected layers. 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 [34]. 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 if $\boldsymbol{t}$ 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 [72]. We first use $\texttt{tanh}(\texttt{FC}_{|\boldsymbol{h}^L| \to 1}(\boldsymbol{h}^L)$ to generate $v_{\texttt{gmm}}$ and then use $\texttt{softmax}(\texttt{FC}_{|\boldsymbol{h}^L| \to s}(\boldsymbol{h}^L))$ 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. A *sequence generation* mechanism is also utilized to generate attributes separately in sequential time-steps [73]. It uses recurrent neural networks, such as long short-term memory (LSTM) networks [33] for G. Specifically, it models a record $\boldsymbol{t}$ as a sequence and each element of the sequence is an attribute $\boldsymbol{t}_j$. It uses LSTM to generate $\boldsymbol{t}$ at multiple timesteps, where the j-th timestep is used to generate $\boldsymbol{t}_j$. Let $\boldsymbol{h}^j$ and $\boldsymbol{f}^j$ respectively denote the hidden state and output of the LSTM at the j-th timestep. Then, we have:

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

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

4.3 GAN Training

This paper investigates two alternatives to train GAN: (1) the vanilla training algorithm with an improved loss function, and (2) the differential privacy preserving (DP) GAN training.

Vanilla GAN Training. We use the vanilla GAN training algorithm [31] (VTrain) to iteratively optimize parameters θ_d in D and θ_g in G.

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

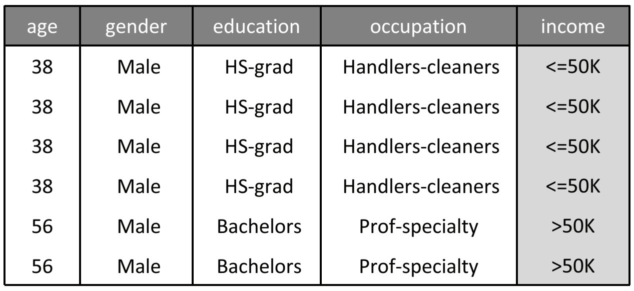
$$\theta_g \leftarrow \theta_g - \alpha_g \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(\boldsymbol{z}^{(i)}))),$$

where m is the minibatch size and α_d (α_g) is learning rate of D (G). One limitation of the algorithm is that it may not provide sufficient gradient to train G in the early iterations [31]. Existing work [73] introduces the KL divergence between real and synthetic data to warm up model training. Let $\mathtt{KL}(\mathcal{T}[j], \mathcal{T}'[j])$ denote the KL divergence regarding attribute $\mathcal{T}[j]$ between the sampled real examples $\{\boldsymbol{t}^{(i)}\}_{i=1}^{m}$ and synthetic samples $\{G(\boldsymbol{z}^{(i)})\}_{i=1}^{m}$. Specifically, we optimize G by considering the original loss and KL divergences regarding all attributes, i.e.,

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

Differential Privacy Preserving GAN Training. We consider differential privacy [26], 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 [71] in the GAN training process.

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 enable 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). 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.



age	gender	education	occupation	income
38	Male	HS-grad	Handlers-cleaners	<=50K
38	Male	HS-grad	Handlers-cleaners	<=50K
38	Male	HS-grad	Handlers-cleaners	<=50K
38	Male	HS-grad	Handlers-cleaners	<=50K
56	Male	Bachelors	Prof-specialty	>50K
56	Male	Bachelors	Prof-specialty	>50K

Fig. 4: An example of mode collapse in GAN training, which results in similar, or even nearly duplicated records in the synthetic data.

5 Optimizations in GAN Training

This section presents three optimization techniques to handle difficulties in training a GAN model on real-world tabular data. First, a well-recognized obstacle in GAN training is *mode collapse* such that the generator produces records with limited varieties. We discuss how to avoid *mode collapse* in Section 5.1. Second, the performance of GAN heavily relies on the quality and diversity of the real data $\mathcal{T}$, and it is challenging to train GAN with limited data. To address this, we introduce a *data augmentation* strategy in Section 5.2. Third, most real-world data has *imbalanced* label distribution, which increases the difficulty of data synthesis, especially for records with minority labels. We examine how to adopt conditional GAN for synthesizing data with imbalanced label distribution in Section 5.3.

5.1 Avoiding Mode Collapse

We investigate *mode collapse* [66,56], a well-recognized challenge in GAN training. Mode collapse would result in similar, or even nearly duplicated records in synthetic table $\mathcal{T}'$, as shown in Figure 4. 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, many records will have the same values for most of the attributes as well as the labels. Consequently, the synthetic table $\mathcal{T}'$ would fail to preserve the data utility of original table

$\mathcal{T}$. For example, a classifier trained on synthetic table $\mathcal{T}'$ may perform badly on the test set.

Our experimental study shows that, when mode collapse happens, the loss of D will easily decrease to a low value. In contrast to D, the loss of G can hardly decrease, it can even increase in the training process. This phenomenon means that D can be trained very well while the G fails to converge. This is because that, when D is trained too well, 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 first analyze why the gradient of generator G will disappear when discriminator D is perfectly trained. As analyzed in [31], given any generator G, the optimal discriminator D, denoted as $D^*(\boldsymbol{t})$, is

$$D^*(\boldsymbol{t}) = \frac{p_{data}(\boldsymbol{t})}{p_{data}(\boldsymbol{t}) + p_g(\boldsymbol{t})}, \tag{9}$$

where $p_{data}(\boldsymbol{t})$ and $p_g(\boldsymbol{t})$ respectively denote distributions of the real data and the synthetic data. Then, by substituting $D^*(\boldsymbol{t})$ into Equation (1), we have:

$$\begin{aligned} V(G, D) &= \mathbb{E}_{\boldsymbol{t} \in p_{data}(\boldsymbol{t})}\big[\log D^*(\boldsymbol{t})\big] + \mathbb{E}_{\boldsymbol{t} \in p_g(\boldsymbol{t})}\big[\log(1 - D^*(\boldsymbol{t}))\big] \\ &= \mathbb{E}_{\boldsymbol{t} \in p_{data}(\boldsymbol{t})}\big[\log \frac{p_{data}(\boldsymbol{t})}{p_{data}(\boldsymbol{t}) + p_g(\boldsymbol{t})}\big] \\ &+ \mathbb{E}_{\boldsymbol{t} \in p_g(\boldsymbol{t})}\big[\log \frac{p_g(\boldsymbol{t})}{p_{data}(\boldsymbol{t}) + p_g(\boldsymbol{t})}\big] \\ &= 2\texttt{JSD}(p_{data}||p_g) - 2\log 2, \end{aligned} \tag{10}$$

where $\texttt{JSD}(p_{data}||p_g)$ is the *Jensen-Shannon divergence* between the two distributions p_{data} and p_g [31].

The existing study [11] has shown that $\texttt{JSD}(p_{data}||p_g)$ tends to be a constant (i.e., $\log 2$) in most of the cases. The main reason is that the support sets of p_{data} and p_g are low dimensional manifolds in a high dimensional space, and thus the measurement of the overlap between p_{data} and p_g is 0. Please refer to [11] for more detailed analysis. Then, based on Equation (10), we obtain a constant value function $V(G, D)$. Thus, the gradients of $V(G, D)$ will become 0, leading to the fact that generator G fails to converge. To address this problem, we investigate the following two strategies.

Wasserstein GAN. The first strategy is to utilize the training algorithm of Wasserstein GAN (WTrain) [12], which is commonly used for addressing mode collapse in image synthesis. The idea is to use the *Wasserstein Distance* to replace the *Jensen-Shannon divergence.*

Specifically, it removes the `sigmoid` function of D and modifies the loss functions into

$$\begin{aligned} \mathcal{L}_D &= -\mathbb{E}_{\boldsymbol{t} \sim p_{data}(\boldsymbol{t})}[D(\boldsymbol{t})] + \mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))] \\ L_G &= -\mathbb{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))]. \end{aligned} \tag{11}$$

Moreover, it clips the parameters of D in each iteration to make sure that $D(\boldsymbol{t})$ can satisfy the *Lipschitz continuity*, i.e., $|D(\boldsymbol{t})|_L \leq K$. In this case, given an optimal discriminator $D^*(\boldsymbol{t})$, the value function becomes

$$\begin{aligned} V_w(G, D) =& \mathbb{E}_{\boldsymbol{t} \in p_{data}(\boldsymbol{t})}\big[D^*(\boldsymbol{t})\big] - \mathbb{E}_{\mathbf{t} \in p_g(\mathbf{t})}\big[D^*(\boldsymbol{t})\big] \\ =& \max_{|D(\boldsymbol{t})|_L \leq K} \mathbb{E}_{\boldsymbol{t} \in p_{data}(\boldsymbol{t})}\big[D(\boldsymbol{t})\big] - \mathbb{E}_{\mathbf{t} \in p_g(\mathbf{t})}\big[D(\boldsymbol{t})\big] \\ \approx& \texttt{WD}(p_{data}, p_g), \end{aligned} \tag{12}$$

where $\texttt{WD}(p_{data}, p_g)$ is the *Wasserstein Distance* between p_{data} and p_g. Different from $\texttt{JSD}(p_{data}||p_g)$, even when the measurement of the overlap between p_{data} and p_g is 0, $\texttt{WD}(p_{data}, p_g)$ is not a constant [12]. Thus, gradient of G will not disappear even when D is well trained. Specifically, It changes the gradient optimizer from `Adam` to `RMSProp` and uses T_g iterations to optimize G. In each G's training iteration, it uses T_d iterations to train D, and then trains G. In particular, it clips parameters θ_d of D into an interval $[-c_p, c_p]$ after each training iteration of D.

Simplified Discriminator. To avoid mode collapse, we can also use a very simple strategy: we still use the vanilla GAN training algorithm VTrain, while simplifying the neural network architectures of discriminator D. For example, we can reduce the numbers of layers or neurons in the neural network of the discriminator D. The idea is to limit the capacity of D, which can make it hard to obtain a well trained D, then avoids the chance of gradient disappearance of generator G.

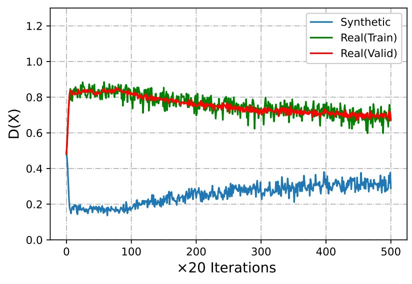
We will empirically evaluate the performance of these two strategies in alleviating mode collapse of GAN training. Please refer to Section 7.2.1 for the results.

5.2 Training GAN with Limited Data

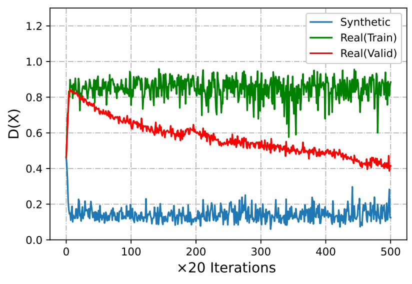
The size of our original table $\mathcal{T}$ (i.e., the number of tuples in $\mathcal{T}$) will significantly affect the performance of **TDGAN**. Typically, an important premise of obtaining a well-performed generator G that synthesizes "near-real" data is having sufficient original data as the training set. However, this may not be practical due to the high cost of data collection and labeling. To address the problem, this section investigates an optimization approach to training a GAN model with limited data. We first analyze the effect of limited data on GAN training in Section 5.2.1, and then introduce a novel solution, called MaskGAN, in Section 5.2.2.

5.2.1 Effect of Limited Data on GAN Training

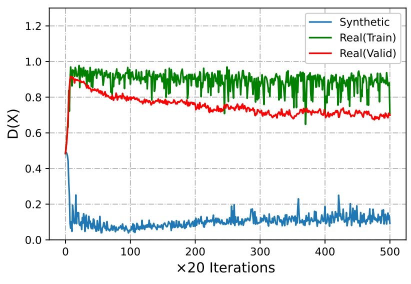
We conduct an empirical investigation on the effect of limited data on the `Adult` dataset (details of the `Adult`



(a) Vanilla GAN with sufficient data.



(b) Vanilla GAN with limited data.



(c) MaskGAN with limited data.

Fig. 5: Per-iteration performance of discriminator D on training and validation sets under different settings. (a) D is trained using Vanilla GAN with sufficient data. (b) D is trained using Vanilla GAN with limited data. (c) D is trained using our proposed MaskGAN with limited data.

dataset can be found in Section 6). Specifically, we report the output probability $D(\boldsymbol{t})$ of discriminator D for each sample $\boldsymbol{t}$ fed to D, where $\boldsymbol{t}$ could either come from the *training* set and the *validation* set of original data $\mathcal{T}$, or be synthesized by generator G. Recall that an optimized discriminator D will produce higher probability $D(\boldsymbol{t})$ if $\boldsymbol{t}$ come from $\mathcal{T}$, while giving lower $D(\boldsymbol{t})$ for a synthetic sample.

Figure 5(a) shows the per-iteration performance of D trained under Vanilla GAN with *sufficient* data. As expected, values of $D(\boldsymbol{t})$ for samples from both training and validation sets are higher than 0.5, which means that D tends to classify the samples as "real data". On the other hand, values of $D(\boldsymbol{t})$ for synthetic samples are lower than 0.5, and thus D is likely to identify them as fake samples. Also, with the increase of training iterations, the gap between real and fake samples becomes smaller, which means that generator G is improved to generate indistinguishable samples that fool D.

However, the behavior of discriminator D is significantly changed given limited training data, e.g., 10% of the original training set, as shown in Figure 5(b). We have an interesting observation that the values of $D(\boldsymbol{t})$ for samples from validation set sharply decrease when increasing the training iteration. The results indicate that discriminator D gradually tends to classify samples from the validation set as "fake data", although these samples are actually real. The main reason is that D *overfits* to the limited training data, that is, it simply memorizes the distribution of the training data rather than understanding how to distinguish real samples from synthetic ones. Consequently, D will not provide sufficient gradients to update G [78,38,82], leading to poor performance of synthetic data generation.

5.2.2 MaskGAN*: A Data Augmentation Approach*

To mitigate the overfitting issue of discriminator D, we propose an approach called MaskGAN. The ba-

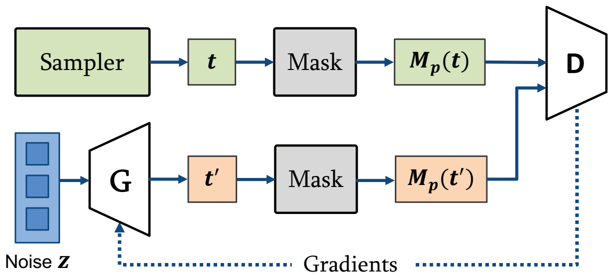


Fig. 6: Overview of MaskGAN. To mitigate overfitting of discriminator D, MaskGAN applies a masking function to real/synthetic samples before feeding these samples to D.

sic idea is to introduce *data augmentation* techniques that enhance the size and quality of the limited training data for D. Existing studies on GAN-based image synthesis have shown that data augmentation is very effective to mitigate overfitting and thus improve the performance of GAN training with limited data [82, 38]. However, supporting data augmentation for tabular data is quite challenging, because, different from images, tabular datasets cannot be augmented by a predefined set of transformation operations, such as pixel blurring, color transforms, image-space filtering, etc.

To address the challenge, MaskGAN utilizes a simple yet effective data augmentation operation. As illustrated in Figure 6, MaskGAN applies a masking function M_p to the training data sampled from the real distribution and the synthetic data generated by the generator G before feeding them into the discriminator D in the training process. Note that, although the masking function has achieved superior performance in our experiments, there could be more sophisticated data augmentation operations for tabular data . We will take a more comprehensive study as the future work.

Intuitively, the masking function aims to only allow discriminator D to observe a random subset of values in the original tuples. Given a transformed tuple $\boldsymbol{t}$, we formally define the masking function over the tuple as

$$M_p(\boldsymbol{t}) = \boldsymbol{t} \odot \boldsymbol{m}_p, \tag{13}$$

where $\odot$ is the element-wise multiplication and $\boldsymbol{m}_p \in \{0,1\}^N$ is a binary mask vector that each element $m_i \in \boldsymbol{m}_p$ indicates whether the corresponding $\boldsymbol{t}[j]$ is visible ($m_i = 1$) or masked ($m_i = 0$).

In particular, the mask vector $\boldsymbol{m}_p$ is randomly sampled from a binomial distribution with a masking probability $p(m = 0) = p$. For example, given a tuple $\boldsymbol{t} = \{38, 1, 9, 23, 1\}$ and $p = 0.4$, $M_{0.4}$ will first randomly sample a mask vector $\boldsymbol{m}_{0.4} = \{1, 1, 0, 0, 1\}$, then obtain the masked tuple $M_{0.4}(\boldsymbol{t}) = \boldsymbol{t} \odot \boldsymbol{m}_{0.4} = \{38, 1, 0, 0, 1\}$ for the discriminator D.

Based on the aforementioned masking function M_p, we introduce new loss functions for MaskGAN as

$$\begin{aligned}\mathcal{L}_D = &- \mathbb{E}_{\boldsymbol{t}\sim p_{data}(\boldsymbol{t})}[\log D(M_p(\boldsymbol{t}))] \\ &- \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(M_p(G(\boldsymbol{z}))))] \\ L_G = &\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(M_p(G(\boldsymbol{z}))))].\end{aligned} \tag{14}$$

Our experimental results show that the masking function can achieve satisfied results to mitigate overfitting of D, given a well-tuned masking probability p. For example, the result shown in Figure 5(c) has revealed a fact that, after applying $M_{0.4}$ to the limited training set mentioned above, the scores of validation set can be closer to the training data compared with the results shown in Figure 5(b). This indicates that MaskGAN can effectively alleviate the overfitting of the discriminator D. This is because it is more difficult for D to memorize the distribution of the training data after the masking function is applied.

Remarks. We also examine a question about whether MaskGAN can capture the distribution of the real data given that it can only access masked tuples. Some existing studies, such as Cheng et al. [49] and Liu et al. [51], consider a very similar problem for training a GAN model with data with missing values. They also apply masking functions to samples before feeding the samples to D. The studies find that that GAN can implicitly undoes the masking function and learn the original distribution as long as we (1) control the magnitude of the masking function (i.e., the mask probability p), and (2) apply the same masking function to both real and synthetic samples. Our empirical study in this paper also validate the above result: when we select an appropriate masking probability p for the masking function M_p and apply it to all samples, MaskGAN performs well for effective data augmentation.

5.3 Generating with Imbalanced Data

The imbalanced label distribution in real-world data may result in *insufficient training* for records with minority labels [72], so there could be a deviation between the distribution of the data with minority label which can also affect the utility of the synthetic data. For instance, most records in our example table in Figure 1 have `income` $\leq 50K$, while only a few with `income` $> 50K$.

To handle this issue, some studies [72] apply conditional GAN [57] to data synthesis, which encodes a label as a condition vector $\boldsymbol{c}$ to guide G (D) to generate (discriminate) samples with the label. We first introduce the design of the conditional GAN. Moreover, we also propose a different sampling strategies: label-aware sampling that gives fair opportunity for samples with different labels.

Design of 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. Specifically, given a domain Ω of labels, we use one-hot encoding (see Section 4.1) to encode the labels. Then, we concatenate the prior noise $\boldsymbol{z}$ and the condition vector $\boldsymbol{c}$ as a new vector $\boldsymbol{z}'$ and feed $\boldsymbol{z}'$ to G. Moreover, we also consider condition $\boldsymbol{c}$ in discriminator $D(\boldsymbol{t}|\boldsymbol{c})$ in the similar way. 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.,

$$\begin{aligned}\min_G \max_D V(G,D) = &\mathbb{E}_{\boldsymbol{t}\in p_{data}(\boldsymbol{t})}\big[\log D(\boldsymbol{t}|\boldsymbol{c})\big] \\ &+ \mathbb{E}_{\mathbf{z}\in p_z(\mathbf{z})}\big[\log(1 - D(G(\mathbf{z}|\boldsymbol{c})))\big].\end{aligned} \tag{15}$$

After injecting the conditional input $\boldsymbol{c}$, the target of G is changed to learn the conditional distribution $p_{data}(\boldsymbol{t}|\boldsymbol{c})$, which can then be used to generate the corresponding synthetic tuple $\boldsymbol{t}'$ according to the given condition $\boldsymbol{c}$.

Label-aware sampling in training. To make the data with minority label have sufficient training opportunities, we introduce *label-aware* data sampling in 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 1: Real datasets for our evaluation: #Rec, #C, #N, and #L are respectively numbers of records, numerical attributes, categorical attributes, and unique labels.

Dataset	Domain	#Rec	#N	#C	#L	Skewness
low-dimensional (#Attr≤ 20)						
HTRU2 [6]	Physical	17,898	8	0	2	skew
Digits [8]	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
mid-dimensional (20 <#Attr≤ 50)						
SAT [9]	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 [47]	Web	500,000	7	23	-	-
high-dimensional (#Attr> 50)						
Internet [7]	Web	10,108	1	70	-	-
Diabete [5]	Medicine	91,786	11	44	3	skew

We will evaluate the performance of the conditional GAN and training algorithm CTrain on the dataset with skew label distribution in Section 5.3.

6 Evaluation Methodology

This section presents the methodology of our experimental study that explores the design space and compares with baseline approaches. We make all codes and datasets public at the Github repo[3].

6.1 Datasets

To consider various characteristics of tabular data, e.g., mixed data types, attribute correlation and label skewness, we use 10 real datasets from diverse domains, such as Physical and Social. The datasets are representative that capture different data characteristics, as summarized in Table 1. We describe the detailed information of these datasets as below:
(1) HTRU2 dataset is a physical dataset that contains 17,898 pulsar candidates collected during the High Time Resolution Universe Survey [6]. 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 [8]. 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 of this dataset 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 [9]. 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 is 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 [47]. 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.
(9) Internet dataset contains general demographic information on internet users that come from a survey conducted by the Graphic and Visualization Unit in 1997 [7]. It has 1 numerical attribute and 70 categorical attributes. Since there is no explicit label in the original

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

dataset, we use it for evaluating performance of data synthesis on AQP.

(10) `Diabete` dataset collects 10-year data (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks [5]. It includes 11 numerical attributes and 44 categorical attributes to represent the features of diabetics and diagnoses. We use attribute `Readmitted` as label to predict days to inpatient readmission. The label distribution is also skewed: only 11% of the patients are readmitted in less than 30 days.

To provide an in-depth analysis on synthesis performance by varying degrees of attribute correlation and label skewness, we also use two simulated datasets, `SDataNum` and `SDataCat`, to evaluate data synthesis for records with purely *numerical* and *categorical* attributes respectively. Due to the space limit, we leave the details of how two create these two datasets in appendix B.2.

6.2 Evaluation Framework

We implement our GAN-based tabular data synthesis framework (see Figure 2) and optimization techniques (see Section 5 using PyTorch [62].

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 tabular 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 as follows. We first perform *hyper-parameter search*, as 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 tabular data $\mathcal{T}'$.

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 a classifier f on the training set $\mathcal{T}_{\texttt{train}}$. In our experiments, we consider the following four types of traditional machine learning classifiers and one deep learning based classifier for evaluation.

- *Decision Tree (DT)* is a classifier for classification by training a decision tree model. We adopt 2 decision trees with max depth 10 and 30 respectively.
- *Random Forest (RF)* is a classifier that uses multiple decision trees for training and predicting. We adopt two random forests with max depth 10 and 20 respectively.
- *AdaBoost (AB)* devises an iterative algorithm to train multiple classifiers (weak classifiers), and uses them to form a stronger final classifier.
- *Logical Regression (LR)* is a generalized linear regression model that uses gradient descent method to optimize the classifier for classification.
- *Multilayer Perceptron (MLP)* is a deep learning based classifier. The MLP we implemented in the experiment contains three hidden-layers and an output layer, where the numbers of neurons in the hidden layers are 512, 256 and 128 respectively. We use this classifier to evaluate the performance of synthetic data for deep-learning models.

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. Specifically, let `TP` denote true positives, which are positive tuples correctly outputted by the classifier. Let `FP` denote false positives, which are negative tuples incorrectly outputted by the classifier. Let `FN` denote false negatives, which are positive tuples omitted by the classifier. Then, we can respectively compute precision and recall as $\texttt{P} = \texttt{TP}/(\texttt{TP}+\texttt{FP})$ and $\texttt{R} = \texttt{TP}/(\texttt{TP}+\texttt{FN})$. Based on this, we can compute F1 score as $\texttt{F1} = 2 \cdot \texttt{P} \cdot \texttt{R}/(\texttt{P}+\texttt{R})$.

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 3. The smaller the difference is, the better $\mathcal{T}'$ is for 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 K-Means algorithm 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 $\texttt{Diff}_{\texttt{CST}}$, as defined in Section 3, 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 [47] 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 $\mathcal{T}_{\mathtt{train}}$. For each query, we measure the relative error e' of the result obtained from $\mathcal{T}'$ by comparing with that from $\mathcal{T}_{\mathtt{train}}$. Meanwhile, following the method in [69], 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 3, we compute the relative error difference $\mathtt{Diff}_{\mathtt{AQP}}$ 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 attacks: membership inference attacks and re-identification attack, which are widely used in the existing works [60,53,55,59,21] for privacy evaluation.
(1) Membership Inference Attack for a target GAN model aims to identify if a given record is used in training the GAN model. To this end, we follow [59] to train a classifier as *the attack model*. Based on this, we can use the F1-score of the attack model to measure the attack performance.

We use the method in [59] to train the attack model, which is described as follows. First, we use the synthetic table $\mathcal{T}'$ generated by the target GAN to train another GAN model, named *shadow GAN*. Then, we input each record $\boldsymbol{t}' \in \mathcal{T}'$ to the discriminator D of the shadow GAN to create a positive attack training example, denoted as $(\boldsymbol{t}', D(\boldsymbol{t}'), \mathtt{In})$, where $D(\boldsymbol{t}')$ is the probability produced by D and $\mathtt{In}$ is a label indicating $\boldsymbol{t}'$ is used for training the shadow GAN. Next, we use a *shadow test dataset* $\{(\boldsymbol{t}, D(\boldsymbol{t}), \mathtt{Out})\}$ as negative examples, where `Out` indicates that these examples are not used for training the shadow GAN. Finally, we combine these positive and negative examples as the attack training data for the attack model.
(2) Re-identification Attack aims to directly re-identify the information of the records used in the GAN model training. Here we utilize two metrics to measure the attack performance:

– *Hitting Rate*: It measures how many records in the original table $\mathcal{T}_{\mathtt{train}}$ can be hit by a synthetic record in $\mathcal{T}'$. To measure hitting rate, we first randomly sample 5000 synthetic records from $\mathcal{T}'$. For each sampled record, we measure the proportion of records in $\mathcal{T}_{\mathtt{train}}$ 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.
– *Distance to the closest record (DCR)*: This measures whether the synthetic data is weak from re-identification attacks [60,53]. Given a record t in the original table $\mathcal{T}_{\mathtt{train}}$, 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 each attribute contributes to the distance equally. We sample 3000 records from the original table $\mathcal{T}_{\mathtt{train}}$, 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

This section presents implementation details the methods evaluated in our experiments. All the implementations as well as their experimental settings can be found in our github repository mentioned above.

GAN-based methods. We have implemented the design choices shown in Figure 3.

– We use the code provided by [59] to implement the CNN-based model[4]. We use the hyper parameters 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.
– 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.
– We also implement the three practical optimization techniques in **TDGAN** introduced in Section 5.

Statistical methods. We compare our GAN-based approach with the state-of-the-art statistical synthesis method PrivBayes (or `PB` for simplicity) [79,80] for tabular data, using the source code downloaded here[5]. As `PB` has theoretical guarantee on differential privacy [26], we vary the privacy parameter ϵ to examine the tradeoff between privacy protection and data utility. According

[4] https://github.com/mahmoodm2/tableGAN
[5] https://sourceforge.net/projects/privbayes/

to the original papers [79,80], 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 [43,65] for tabular data synthesis. A typical VAE is composed by an encoder that encodes the input data to a low dimensional latent variable, and a decoder that receives the latent variable and reconstructs the input data. We adopt the loss function that consists of both the reconstruction loss and the KL divergence [23]. We use binary cross-entropy (BCE) loss for categorical attributes and 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 [54] 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. We run 5 times for each experiment and report the average result.

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

This section presents the experimental results. To be more specific, Section 7.1 evaluates different design choices of the modules in **TDGAN** as summarized in Figure 3 by investigating the following questions.

– **Q1:** Which neural network architectures are adequate for implementing GAN? And How does data transformation affect the performance?
– **Q2:** Can differential privacy (DP) preserving GAN model produce satisfactory synthetic data utility?

Section 7.2 evaluates our proposed optimization techniques (see Section 5) by answering the questions.

– **Q3:** How effective are different strategies for avoiding *mode collapse*, which is a well-recognized difficulty in training a GAN model?
– **Q4:** Is data augmentation strategy MaskGAN helpful for GAN training with limited data?
– **Q5:** How effective is conditional GAN with label-aware data sampling to address imbalanced label distribution on real and simulated datasets?

Section 7.3 compares **TDGAN** with existing approaches to tabular data synthesis, i.e., `VAE` and `PB` (see Section 6.3), by answering the following questions.

– **Q6:** Is **TDGAN** promising to preserving the data utility compared with the other synthesis methods?
– **Q7:** Can **TDGAN** achieve better trade-off between synthetic data utility and protecting privacy against the risk of re-identification attacks?

7.1 Evaluating the Design of **TDGAN**

7.1.1 Evaluation on Neural Network Architectures and Data Transformation Methods

We evaluate the neural network architectures, 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 appendix B.5).

Evaluation on data utility. We first evaluate synthetic data utility for classification, and will report data utility for clustering and AQP later. Due to space limit, we consider two low-dimensional (#`Attr` ≤ 20) datasets `Adult` and `CovType`, two mid-dimensional ones `SAT` and `Census`, and one high-dimensional dataset `Diabete`. We find similar results on other datasets. Tables 2(a), 2(b), 2(c), 2(d), and 2(e) report the experimental results on data utility for classification, 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`, `SAT` and `Diabete`, as the original code in [59] is not designed for multi-class classification.

The results show that, on the datasets `Adult` and `CovType` with less attributes, LSTM achieves the best performance in most of the cases, i.e., achieving 3% – 94% 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 tabular data synthesis. First, each attribute is generated from a separated noise $\boldsymbol{z}$, which avoids the disturbance among different attributes. Second, 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`, `SAT`, and `Diabete` with more attributes, the

Table 2: Evaluating different neural networks of generator G on synthetic data utility for classification. 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).

Classifier	CNN	MLP				LSTM			
		sn/od	sn/ht	gn/od	gn/ht	sn/od	sn/ht	gn/od	gn/ht
DT10	0.495	0.420	0.108	0.430	0.072	0.544	0.099	0.107	**0.030**
DT30	0.475	0.433	0.134	0.372	0.090	0.519	0.092	0.107	**0.065**
RF10	0.481	0.593	0.107	0.596	0.080	0.528	0.014	0.136	**0.003**
RF20	0.459	0.596	0.143	0.621	0.066	0.561	0.071	0.133	**0.032**
AB	0.225	0.537	0.112	0.556	0.055	0.575	0.099	0.099	**0.023**
LR	0.121	0.599	0.085	0.488	0.042	0.577	0.021	0.084	**0.006**
MLP	0.605	0.310	0.049	0.455	0.016	0.549	0.028	0.076	**0.001**

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

Classifier	MLP				LSTM			
	sn/od	sn/ht	gn/od	gn/ht	sn/od	sn/ht	gn/od	gn/ht
DT10	0.264	0.106	0.363	0.122	0.354	0.121	0.265	**0.091**
DT30	0.421	0.285	0.492	0.302	0.474	0.277	0.427	**0.277**
RF10	0.237	0.072	0.393	0.089	0.326	0.198	0.290	**0.070**
RF20	0.356	0.186	0.473	0.204	0.439	0.207	0.379	**0.173**
AB	0.105	0.019	0.207	0.033	0.200	**0.014**	0.116	0.024
LR	0.191	0.039	0.322	0.036	0.302	0.071	0.261	**0.027**
MLP	0.174	0.047	0.293	0.041	0.273	0.069	0.206	**0.038**

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

Classifier	CNN	MLP				LSTM			
		sn/od	sn/ht	gn/od	gn/ht	sn/od	sn/ht	gn/od	gn/ht
DT10	0.484	0.264	0.221	0.275	0.206	0.488	0.158	0.340	**0.104**
DT30	0.462	0.254	0.199	0.233	0.191	0.472	0.176	0.329	**0.098**
RF10	0.214	0.178	0.054	0.111	0.101	0.184	**0.019**	0.184	0.029
RF20	0.410	0.378	0.192	0.345	0.230	0.416	0.119	0.411	**0.031**
AB	0.506	0.291	0.212	0.127	0.308	0.487	0.118	0.459	**0.004**
LR	0.494	0.157	0.128	0.085	0.248	0.474	0.028	0.409	**0.021**
MLP	0.556	0.265	0.137	0.242	0.231	0.532	0.055	0.436	**0.004**

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

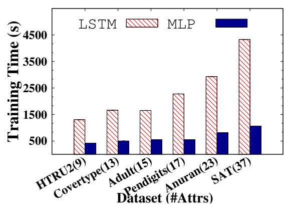
Classifier	MLP		LSTM	
	sn	gn	sn	gn
DT10	0.065	0.061	0.166	**0.055**
DT30	**0.058**	0.059	0.169	0.063
RF10	**0.034**	0.048	0.148	0.048
RF20	0.039	0.039	0.160	**0.037**
AB	**0.041**	0.072	0.234	0.119
LR	**0.039**	0.047	0.138	0.061
MLP	0.064	0.068	0.230	**0.054**

(e) `Diabete` dataset (high-dimensional).

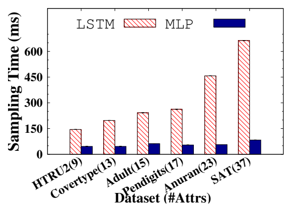
Classifier	MLP		LSTM	
	sn	gn	sn	gn
DT10	0.029	0.034	0.032	**0.027**
DT30	0.025	0.027	0.019	**0.015**
RF10	0.013	**0.007**	0.014	0.018
RF20	**0.019**	0.028	0.026	0.020
AB	**0.021**	0.030	0.041	0.024
LR	0.029	0.025	0.024	**0.017**
MLP	0.047	0.068	0.043	**0.034**

performance advantage of LSTM is less significant. The reason is that, with more attributes, 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 the inferior performance in data synthesis, which is different from image synthesis [63]. This is because matrix input of CNN is only compatible with



(a) Training time (s).



(b) Synthesis time (ms).

Fig. 7: Evaluating different neural networks of generator G on efficiency on model training and data synthesis.

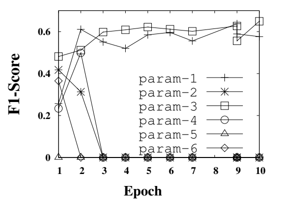
simple normalization and ordinal encoding, which is not effective for tabular data. Moreover, convolution/deconvolution operation in CNN is usually effective for data with *feature locality*. For example, features, which are locally close to each other in the matrix of an image, may also be semantically correlated. However, tabular data does not have such locality.

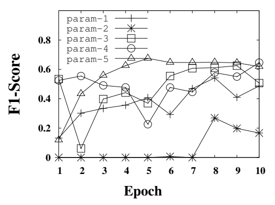
Finding 1: LSTM with appropriate transformation schemes generates the best synthetic data utility for classification. Nevertheless, with more attributes, the performance advantage achieved by LSTM becomes less significant.

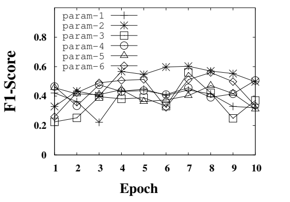
Evaluation on efficiency. We compare the training time and the synthesis time of the generator implemented by different neural networks. For fair comparison on different datasets, we fix input data in model training and output data in data synthesis as 4000 tuples. As shown in Figures 7(a) and 7(b), both training and synthesis time of the LSTM-based generator are longer than that of the MLP-based generator. Moreover, the training and synthesis time of the LSTM-based generator increases significantly with the increase of the number of the attributes, while the time of the MLP-based generator is relatively stable. This is because the LSTM-based generator generates each synthetic tuple attribute by attribute, and, naturally, the more attributes a dataset has, the more time the generator takes for training and synthesis. In contrast, the MLP-based generator generates all attributes in a synthetic tuple at one time. Therefore, the MLP-based generator is more efficient than the LSTM-based generator.

Finding 2: Compared with LSTM, MLP is more efficient in the model training and synthetic data generator processes of generator G.

Evaluation on robustness. We evaluate the robustness of MLP-based and LSTM-based generators wrt. hyper parameters. Given a setting 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 8 shows the results on datasets `Adult` and `CovType`. Note that we find


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


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


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

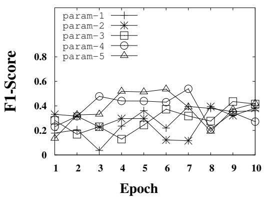

(d) MLP-based G (`CovType`).

Fig. 8: Evaluating GAN model training against different hyper-parameter settings. The MLP-based generator is more robust against various hyper parameters, while the LSTM-based generator is likely to result in mode collapse.

similar trends on other datasets. We have a surprising observation that the LSTM-based generator performs badly in some hyper parameter settings. For example, on the `Adult` dataset, the F1 score drops sharply to 0 after the few early epochs in 4 out of 6 hyper parameter settings. 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-based generator. We will provide an in-depth investigation on mode collapse in Section 7.2.

Finding 3: MLP is more robust against hyper parameters and achieves moderate results, while LSTM is more likely to result in mode collapse if its hyper parameters are not well tuned.

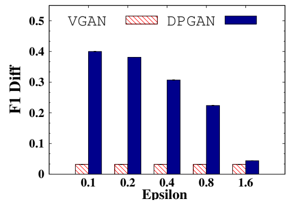
In the remainder of this section, for ease of presentation, we use LSTM with one-hot encoding and GMM-based normalization as default setting.

7.1.2 Evaluation on Vanilla GAN and DPGAN

We compare two GAN training algorithms, vanilla training (VGAN) and differential privacy preserving training (DPGAN), on the `Adult` and `CovType` datasets.

We first consider privacy protection against membership inference attack and re-identification attack. Table 3 shows the results of the membership inference attack. DPGAN outperforms VGAN on protecting privacy against the risk of membership inference attack, given smaller values of ϵ. Moreover, with the increase of ϵ, the privacy protection performance of DPGAN decreases, which is reflected by larger F1-scores of the attack model. This is because ϵ is used to control the privacy level in differential privacy: the larger the ϵ, the lower the privacy level. We also observe similar trends from the performance of re-identification attack shown in Table 4: the hitting rate and DCR are reduced with the decrease of the ϵ and DPGAN outperforms VGAN to protect the privacy against the re-identification attack. Based on the observations, we can conclude that DPGAN performs better than VGAN on privacy protection, and DPGAN can control the privacy level via parameter ϵ.

We also evaluate the DPGAN and VGAN on the data utility. Figure 9 reports the results on preserving the data utility varying privacy level ϵ. We can see that DPGAN can achieve better data utility as the increase of the ϵ, i.e., the decrease of privacy level. However, DPGAN cannot beat VGAN even at the lowest privacy level. 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


(a) `Adult` dataset.

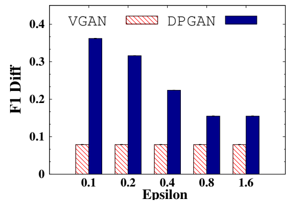

(b) `CovType` dataset.

Fig. 9: Comparing DPGAN and VGAN on varying privacy levels (using DT10 as classifier).

Table 3: Evaluation on privacy protection (F1-score of membership inference attack).

Dataset	VGAN	DPGAN				
		$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.4$	$\epsilon = 0.8$	$\epsilon = 1.6$
`Adult`	0.483	0.450	0.463	0.471	0.490	0.495
`CovType`	0.456	0.430	0.437	0.437	0.438	0.445

Table 4: Evaluation on privacy protection (re-identification attack).

Method	Hitting Rate (%)		DCR	
	`Adult`	`CovType`	`Adult`	`CovType`
DPGAN-0.1	0.25	0.275	0.201	0.093
DPGAN-0.2	0.29	0.320	0.189	0.088
DPGAN-0.4	0.31	0.376	0.176	0.082
DPGAN-0.8	0.36	0.380	0.156	0.082
DPGAN-1.6	0.43	0.425	0.145	0.077
VGAN	0.39	0.463	0.142	0.072

Table 5: Effect of size ratio between synthetic and original tables (using DT10 as classifier).

Dataset	Size ratio: $\vert\mathcal{T}'\vert/\vert\mathcal{T}_{\texttt{train}}\vert$			
	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

Table 6: 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`	0.3658	0.0043	**0.0013**	0.0181	0.0035
`Adult`	0.1141	0.0001	0.0002	0.0001	**0.0001**
`CovType`	-	0.0013	0.0009	0.0099	**0.0002**
`Digits`	-	0.0009	0.0144	0.0189	**0.0002**
`Anuran`	-	0.0032	0.0069	0.0136	**0.0013**
`Census`	0.0526	0.0018	0.0147	0.0006	**0.0006**
`SAT`	-	0.0031	0.0084	0.0005	**0.0002**
`Diabete`	-	0.0043	0.0003	0.0005	**0.0002**

Table 7: 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
`Internet`	-	0.113	0.077	**0.067**	0.097

adversarial training ineffective, as D now has limited ability to differentiate real/fake samples.

Finding 4: Although the current differential privacy (DP) preserving training algorithm for GAN can protect the privacy well, it sacrifices the utility of the synthetic data, which implies that better solutions for DP preserving GAN need to be invented.

7.1.3 Effect of Sample Size for Synthetic Data

We also evaluate whether the sample size $|\mathcal{T}'|$ of synthetic table would affect the utility. Table 5 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.

7.1.4 Evaluation on Clustering and AQP

This sections evaluates if GAN can preserve data utility for clustering and AQP. The results are reported in Table 6 (for clustering) and Table 7 (for AQP). For evaluating AQP, we select the datasets `CovType` and `Census` with more than 100,000 records, and the dataset `Internet` with more than 50 attributes. 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 distribution for the original table, which is also beneficial for clustering and AQP.

7.2 Evaluating the Optimizations in **TDGAN**

This section examines whether the optimization techniques, which are introduced in Section 5, are effective to improve the GAN training for tabular data.

7.2.1 Evaluation on Avoiding Mode Collapse

We evaluate the following training strategies to mitigate the mode collapse issue in **TDGAN**: (i) VTrain (with KL divergence), (ii) Wasserstein GAN training (WTrain) and (iii) VTrain with simplified discriminator D (Simplified). More details of WTrain and Simplified can be referred to Section 5.1

Figure 10 reports the overall results on the four representative datasets. We have an clear observation that Wasserstein GAN does not have advantage over vanilla GAN training, which is different from the image synthesis scenarios. On the other hand, VTrain with simplified discriminator (Simplified) achieves better performance than VTrain and WTrain. For example, on the `Adult` dataset, it reduces F1 difference compared with VTrain on most classifiers. To analyze the reasons, we report more detailed per-epoch result of Simplified against various hyper-parameters given the LSTM-based generator, as shown in Figure 11. We can find that, on the two datasets `Adult` and `CovType`, compared with the "normal" discriminator, a simplified discriminator D is effective to reduce the chances of mode collapse. The results reveal a fact that Simplified makes D not trained too well, and thus avoids the chance of gradient disappearance of generator G.

We also evaluate the effect of mode collapse on privacy by using our re-identification attack. We choose the synthetic data that contains many duplicate records as the `Collapse` results. Table 8 shows the results on `Adult` and `CovType`. We find that mode collapse, which generates synthetic data with many duplicate records, can reduce the risk of privacy disclosure, as the number of records that are disclosed becomes limited. However, mode collapse significantly degrades data utility, outweighing its benefits on privacy protection. For example, `No-Collapse` outperforms `Collapse` by 2 orders of

Table 8: Effect of mode collapse on data utility and privacy.

Datasets	Results	Privacy		Utility
		Hitting Rate (%)	DCR	F1 Diff
`Adult`	`Collapse`	**0.369**	**0.205**	0.248
	`No-collapse`	0.389	0.142	**0.030**
`CovType`	`Collapse`	**0.154**	**0.134**	0.556
	`No-collapse`	0.463	0.072	**0.091**

magnitude on data utility, while `Collapse` only reduces the hitting rate by 66% or increases the DCR by 100%.

Finding 5: Vanilla GAN training with simplified discriminator is shown effective to alleviate mode collapse, and outperforms Wasserstein GAN training in preserving data utility.

7.2.2 Evaluation on Training with Limited Data

We evaluate the performance of our data augmentation method MaskGAN to tackle GAN training with limited training data. Specifically, we randomly sample 20%, 10% and 5% tuples from the `Adult` and `Census` datasets to prepare training sets with limited data. We compare our GAN-based framework (denoted as GAN) with MaskGAN on the synthetic data utility for classification. In particular, we tune the mask probability p of each dataset for MaskGAN based on the corresponding validation sets. Figure 12 shows the experimental results. We observe a significant increase on the F1 difference of our GAN-based framework, e.g., 0.08 (5% of the `Adult` dataset) vs. 0.004 (100% of the `Adult` dataset). As analyzed in Section 5.2.1, this is attributed to the overfitting problem of discriminator D. We also find that our data augmentation method MaskGAN can reduce the F1 difference, while significantly outperforming the ordinary GAN-based framework, given all sample ratios on both `Adult` and `Census` datasets. The results indicate that the masking-based augmentation strategy applied in MaskGAN can enhance the size and quality of training data of discriminator D, and thus mitigates the overfitting issue of D and enables the generator to preserve the utility of the synthetic data.

Furthermore, we evaluate the effect of limited training data on privacy using our re-identification attack. We use 10% of the training data to train the GAN, denoted as `Limited`, to compare with the GAN trained on all training data, denoted as `Sufficient`. Table 9 reports the results on `Adult` and `CovType`. We can see that the limited training data can reduce the risk of re-identification. This is because the synthetic data generated by GANs trained on limited data fails to follow the same distribution as the original data, making it difficult to disclose information about the original data.

Table 9: Effect of limited training data on data utility and privacy.

Datasets	Training	Privacy		Utility
		Hitting Rate (%)	DCR	F1 Diff
`Adult`	`Limited`	**0.284**	**0.200**	0.076
	`Sufficient`	0.389	0.142	**0.030**
`CovType`	`Limited`	**0.325**	**0.105**	0.301
	`Sufficient`	0.463	0.072	**0.091**

However, similar to the results in Section 7.2.1, it sacrifices too much data utility.

Finding 6: The simple data augmentation strategy MaskGAN with random masking has been shown to be effective to mitigate the overfitting problem encountered by discriminator D, consequently improving synthetic data utility.

7.2.3 Evaluation on Synthesis with Imbalanced Data

We compare the traditional 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`. We also find similar results on the `HTRU2` dataset, and do not include the results due to the space limit. As shown in Figure 13, `CGAN-V` gains very limited improvements over `VGAN`, and sometimes it performs worse than `VGAN`. 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.

To evaluate the effect of label skewness, we also evaluate the conditional GAN on the simulated data. We set the correlation degree as 0.5 for both `SDataNum` and `SDataCat`, and consider their `balanced` and `skew` settings. As shown in Figure 14, 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.

Furthermore, we also evaluate the effect of absolute sizes (i.e., number of instances) of the minority class in the training data. To keep the skewness of the original training data while controlling the absolute size of the minority class, we follow [81] to sample at different ratios from the original training data to train the GAN model. Figure 15 shows the results of data utility on

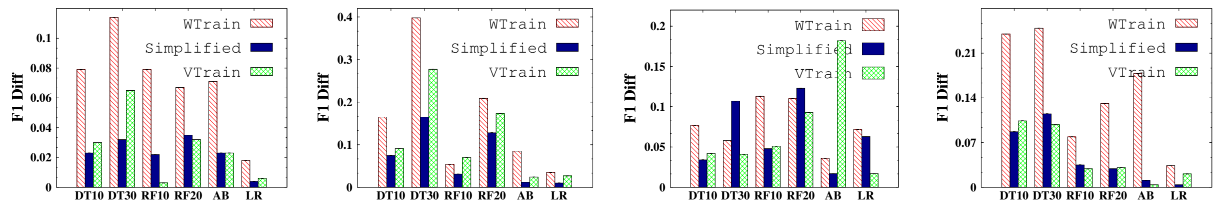
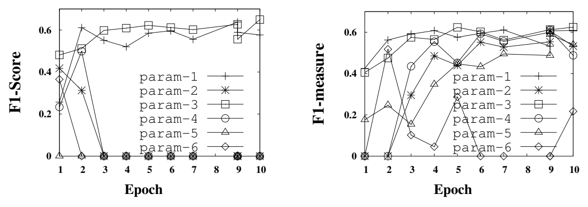

(a) `Adult` dataset. (b) `CovType` dataset. (c) `SAT` dataset. (d) `Census` dataset.

Fig. 10: Comparison of strategies for mitigating mode collapse. Vanilla GAN training with simplified discriminator is effective, and outperforms Wasserstein GAN training in preserving data utility.


(a) Normal D (`Adult`). (b) Simplified D (`Adult`).

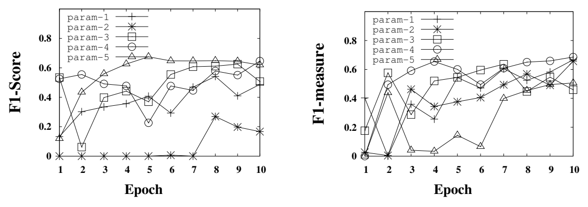

(c) Normal D (`CovType`). (d) Simplified D (`CovType`).

Fig. 11: Evaluating the performance of the Simplified strategy on various hyper-parameter settings for LSTM-based generator.

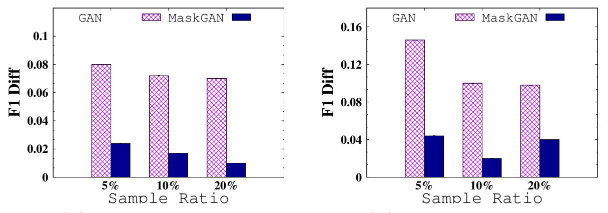

(a) `Adult` dataset. (b) `Census` dataset.

Fig. 12: Comparing the GAN framework with MaskGAN on training with limited data.

datasets `Adult` and `Anuran`. We find that even the absolute size of the minority class is limited (using 10% of the training data), `CGAN-C` and `CGAN-V` can still outperform `VGAN`, while the improvement is less significant. For example, when we reduce the number of records with minor class to 10%, `CGAN-C` can only reduce the F1 difference by 19%-22%, while it can reduce the F1 difference by 56%-65% when the records with minority class are sufficient. As analyzed in Section 5.2.1, the performance of GAN would degrade when the size of the training data is limited, so the performance of conditional GAN would also be affected when the data with minority class is insufficient.

Table 10: **Effect of the conditional training on data utility and privacy.**

Datasets	Methods	Privacy		Utility
		Hitting Rate (%)	DCR	F1 Diff
`Adult`	CGAN	0.421	0.122	**0.005**
	VGAN	**0.389**	**0.142**	0.030
`CovType`	CGAN	0.501	0.089	**0.071**
	VGAN	**0.463**	**0.072**	0.091

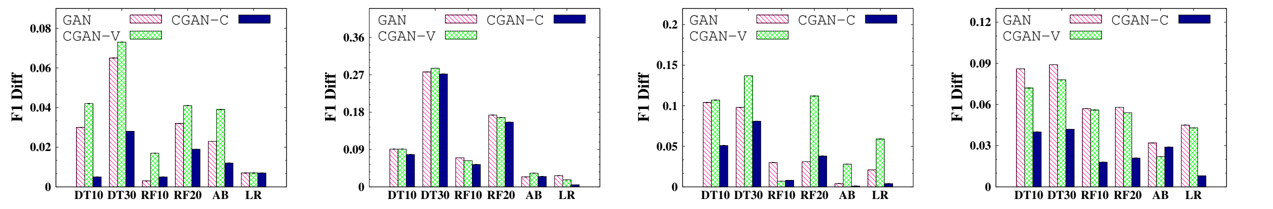
Additionally, we evaluate the effect of the conditional training on utility and privacy. The results reported in Table 10 show that conditional GAN may have more potential risk of privacy disclosure as it can achieve better performance on preserving the data utility, which is a similar trend as we find in Section 7.2.1 and Section 7.2.2.

Finding 7: Conditional GAN with label-aware data sampling is helpful to address imbalanced label distribution on real and simulated datasets, and improves the utility of synthetic data, while the improvement would be less significant when the absolute size of the data with minority class is limited.

7.2.4 Comparing Different Optimization Algorithms

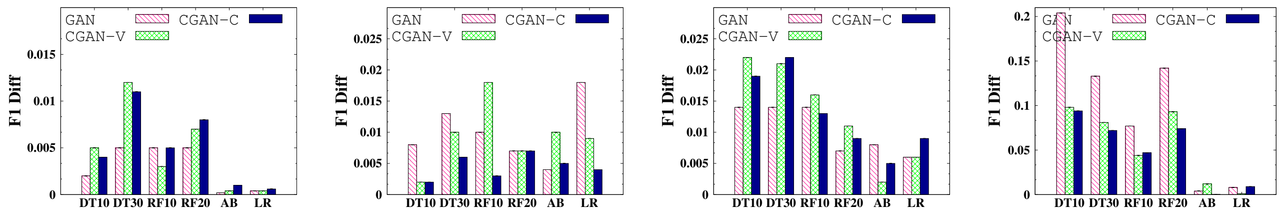
In this section, we compare the three optimizations in **TDGAN** introduced above: wasserstein GAN (WGAN) for avoiding mode collapse, MaskGAN for training with limited data, and conditional GAN (CGAN) with label-aware training strategy for synthesizing imbalanced data, and vanilla GAN (VGAN). To this end, we utilize *skill rating* [58] to evaluate the generator G of these GAN models. Skill rating is a method for assigning a numerical skill to players in a player-vs-player game. Higher skill ratings indicate higher player skills, i.e., stronger ability of data synthesis of G.

Figure 16 shows the skill rating scores of the generator G trained by VGAN, CGAN, MaskGAN and



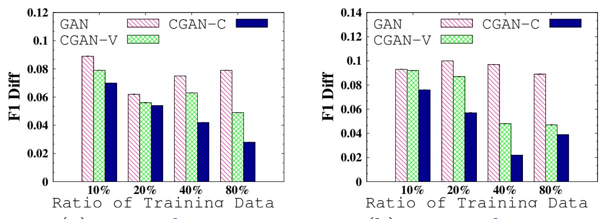
(a) `Adult` dataset. (b) `CovType` dataset. (c) `Census` dataset. (d) `Anuran` dataset.

Fig. 13: Evaluating conditional GAN on synthetic data utility for classification.



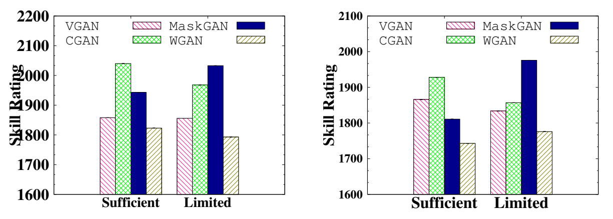
(a) `SDataNum`-balance dataset. (b) `SDataNum`-skew dataset. (c) `SDataCat`-balance dataset. (d) `SDataCat`-skew dataset.

Fig. 14: Evaluating conditional GAN on synthetic data utility for classification (Simulated data).



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

Fig. 15: Evaluating conditional GAN on different absolute sizes of the minority class for training.



(a) `Adult` dataset (Skew). (b) `Digits` dataset(Balance).

Fig. 16: Comparing different optimization algorithms via skill rating of the generator.

WGAN on the `Adult` and `Digits` datasets. In particular, we consider both all training data and limited training data (10% of the original training data). We find that the results reported in Figure 16 are consistent with the findings we concluded before. First, we find that WGAN performs worse than other training algorithms when synthesizing tabular data (Section 7.2.1). Our experiments also validate that MaskGAN outperforms other methods when the training data is limited, as it can alleviate the overfitting of the discriminator (Section 7.2.2). We have also observed that the improvement brought by CGAN is affected by the skewness of the training data and the absolute size of the records with minority classes (Section 7.2.3).

7.3 Comparing with Other Data Synthesis Methods

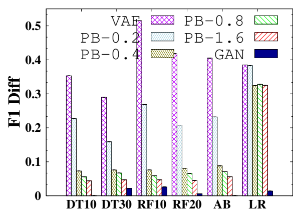
This section compares **TDGAN** with existing approaches to tabular data synthesis, i.e., `VAE` and `PB` on both synthetic data utility and privacy.

Table 11: Comparison of GAN and `PB` on privacy.

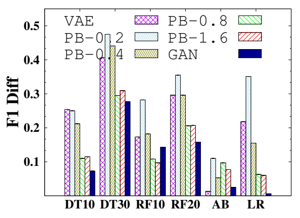
Method	Hitting Rate (%)		DCR	
	`Adult`	`CovType`	`Adult`	`CovType`
PB-0.1	0.49	0.002	0.164	0.106
PB-0.2	0.88	0.006	0.147	0.094
PB-0.4	2.16	0.022	0.123	0.082
PB-0.8	4.40	0.056	0.112	0.073
PB-1.6	4.64	0.070	0.110	0.069
GAN	0.42	0.501	0.122	0.089

7.3.1 Evaluation on Synthetic Data Utility

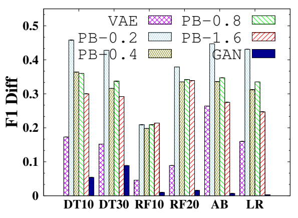
Figure 17 shows the experimental results on synthetic data utility on our real datasets. First, with the increase of privacy parameter ϵ, the result of `PB` becomes better as ϵ is used to control the privacy level. Second, `VAE` achieves moderate results, but the generated synthetic data is still worse than that synthesized by GAN. This is similar to the case in image synthesis [25]: 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 complex tabular data.



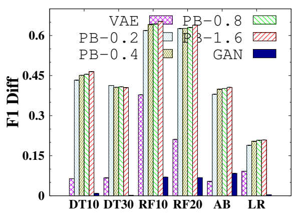
(a) `Adult` dataset.



(b) `CovType` dataset.



(c) `Census` dataset.



(d) `SAT` dataset.

Fig. 17: Comparison of approaches to tabular data synthesis on data utility for classification.

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 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 tabular data.

Finding 8: GAN significantly outperforms `VAE` and `PB` on synthetic data utility for classification. 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.

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 12. 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 13, 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 statistical 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. 0.411 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 [69] studies more sophisticated techniques to optimize `VAE`, such as partitioning the data and using multiple `VAE` models, adding rejection criteria for data sampling, etc.

Table 12: Comparison of approaches to tabular 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	**0.0008**
`CovType`	0.0089	0.0227	0.0121	0.0071	0.0031	**0.0002**
`Adult`	0.0891	0.0892	0.0959	0.0729	0.0494	**0.0001**
`Digits`	0.0425	0.2025	0.1839	0.1749	0.1545	**0.0002**
`Anuran`	0.2184	0.2989	0.2170	0.1505	0.1617	**0.0013**
`Census`	0.0010	0.0189	0.0101	0.0011	0.0112	**0.0005**
`SAT`	0.4891	0.2451	0.2277	0.2289	0.2279	**0.0002**
`Diabete`	0.0005	0.0010	0.0004	0.0004	0.0003	**0.0002**

Table 13: Comparison of approaches to tabular data synthesis on data utility $\texttt{Diff}_{\texttt{AQP}}$ for 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	**0.227**
`Bing`	0.632	0.830	0.805	0.783	0.761	**0.411**
`Internet`	0.291	0.696	0.590	0.481	0.449	**0.069**

We will leave a more thorough comparison with such new techniques in the future work.

Finding 9: GAN is also very promising for preserving the utility of original data for supporting the applications of clustering and AQP.

7.3.2 Evaluation on Privacy

Table 11 compares GAN with `PB` on protecting privacy against the risk of re-identification, measured by Hitting Rate and DCR introduced in Section 6.2. First, 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 14%. 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 [79,80], and thus a synthetic numerical value is seldom similar to the original one. Second, 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 10: 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.

8 Conclusion & Future Direction

In this paper, we have conducted a comprehensive experimental study for applying GAN to tabular data synthesis. We introduced a unified framework and defined a design space of the solutions that realize GAN. We developed three optimization techniques to handle difficulties in GAN training. 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 tabular data synthesizer.

Overall Evaluation for GAN. GAN is very promising for tabular data synthesis. It generates synthetic data with very good utility on classification, clustering and AQP (*Findings 8 and 9*). It can also achieves competitive performance on protecting privacy against the risk of re-identification (*Finding 10*). However, GAN has limitations on providing provable privacy protection: the current solution cannot produce superior data utility when preserving differential privacy (*Finding 4*).

Neural Network Selection & Data Transformation. For ordinary users with limited knowledge on deep learning, we suggest to use MLP to realize GAN, as MLP is more efficient and robust to achieve moderate results without parameter tuning (*Findings 2 and 3*). 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 5*). We introduce a data augmentation strategy with random masking to improve GAN training with limited data (*Finding 6*). We leverage conditional GAN for datasets with imbalanced data distribution (*Finding 7*).

Tabular Data Representation. Data transformation that converts original records to recognized input of GAN does affect the overall performance, which shows that representation of tabular data is important (*Finding 1*). This may imply an interesting future work that co-trains GAN and record representation through a hybrid optimization framework.

We identify some future directions in GAN-based tabular data synthesis that are worthy of exploration.

(1) Providing provable privacy protection. We have shown that GAN has limitations on providing provable privacy protection, i.e., differential privacy. Although enabling GAN to support differential privacy is a hot research topic in ML [71,35], 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 [64,15]. Despite some preliminary attempt [19], 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, tabular data synthesis should support a variety of applications, including ML tasks over time-series data and data synthesis for supporting AQP with theoretical bounds.

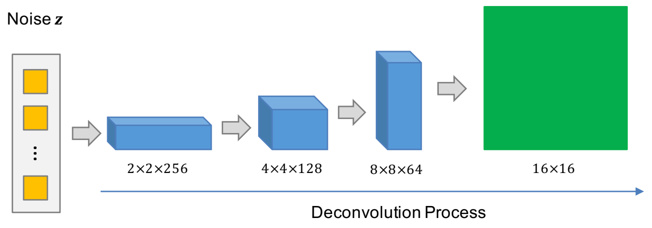
(4) Constructing GAN model automatically. Although the main goal of this paper is comparing different design choices under a unified framework, we can borrow the main idea of AutoML [76] to explore how to train a GAN model automatically for synthesizing tabular data for a given dataset, and this can be a potential avenue for future research.

References

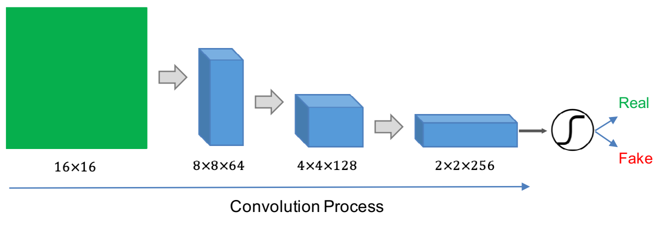
1. Adult data set. `https://archive.ics.uci.edu/ml/datasets/Adult`.
2. Anuran calls (mfccs) data set. `http://archive.ics.uci.edu/ml/datasets/Anuran+Calls+%28MFCCs%29`.

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. Diabete data set. https://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+1999-2008.
6. Htru2 data set. http://archive.ics.uci.edu/ml/datasets/HTRU2.
7. Internet data set. https://openml.org/search?type=data&status=active&id=372.
8. Pen-based recognition of handwritten digits data set. https://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits.
9. Statlog (landsat satellite) data set. https://archive.ics.uci.edu/ml/datasets/Statlog+%28Landsat+Satellite%29.
10. D. Agrawal and C. C. Aggarwal. On the design and quantification of privacy preserving data mining algorithms. In *PODS*, 2001.
11. M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.
12. M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *CoRR*, abs/1701.07875, 2017.
13. 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.
14. 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.
15. P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.
16. V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci. Deep neural networks and tabular data: A survey. *CoRR*, abs/2110.01889, 2021.
17. J. Brickell and V. Shmatikov. The cost of privacy: destruction of data-mining utility in anonymized data publishing. In *SIGKDD*, pages 70–78, 2008.
18. S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: No silver bullet. In *SIGMOD*, pages 511–519, 2017.
19. 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.
20. 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.
21. 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.
22. 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.
23. C. Doersch. Tutorial on variational autoencoders. *CoRR*, abs/1606.05908, 2016.
24. 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.
25. V. Dumoulin, I. Belghazi, B. Poole, A. Lamb, M. Arjovsky, O. Mastropietro, and A. C. Courville. Adversarially learned inference. In *ICLR*, 2017.
26. C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.
27. C. Esteban, S. L. Hyland, and G. Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *CoRR*, abs/1706.02633, 2017.
28. 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.
29. J. Fan, T. Liu, G. Li, J. Chen, Y. Shen, and X. Du. Relational data synthesis using generative adversarial networks: A design space exploration. *Proc. VLDB Endow.*, 13(11):1962–1975, 2020.
30. L. Gondara and K. Wang. MIDA: multiple imputation using denoising autoencoders. In *PAKDD*, pages 260–272, 2018.
31. 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.
32. T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics. Springer, 2009.
33. S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
34. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.
35. J. Jordon, J. Yoon, and M. van der Schaar. PATE-GAN: generating synthetic data with differential privacy guarantees. In *ICLR*, 2019.
36. J. H. Jr, L. O. Gostin, and P. Jacobson. Legal issues concerning electronic health information: privacy, quality, and liability. *Jama*, 282:1466–1471, 1999.
37. Kaggle. The state of data science and machine learning, 2017. https://www.kaggle.com/surveys/2017.
38. T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila. Training generative adversarial networks with limited data. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
39. J. Kim, J. Jeon, J. Lee, J. Hyeong, and N. Park. OCT-GAN: neural ode-based conditional tabular gans. In J. Leskovec, M. Grobelnik, M. Najork, J. Tang, and L. Zia, editors, *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 1506–1515. ACM / IW3C2, 2021.
40. J. Kim, C. Lee, and N. Park. Stasy: Score-based tabular data synthesis. *CoRR*, abs/2210.04018, 2022.
41. J. Kim, C. Lee, Y. Shin, S. Park, M. Kim, N. Park, and J. Cho. SOS: score-based oversampling for tabular data. In A. Zhang and H. Rangwala, editors, *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pages 762–772. ACM, 2022.
42. D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
43. D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

44. A. Kotelnikov, D. Baranchuk, I. Rubachev, and A. Babenko. Tabddpm: Modelling tabular data with diffusion models. *CoRR*, abs/2209.15421, 2022.
45. J. Lee, J. Hyeong, J. Jeon, N. Park, and J. Cho. Invertible tabular gans: Killing two birds with one stone for tabular data synthesis. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 4263–4273, 2021.
46. 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.
47. 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.
48. N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115, 2007.
49. S. C. Li, B. Jiang, and B. M. Marlin. Misgan: Learning from incomplete data with generative adversarial networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.
50. Z. J. Ling, Q. T. Tran, J. Fan, G. C. H. Koh, T. Nguyen, C. S. Tan, J. W. L. Yip, and M. Zhang. GEMINI: an integrative healthcare analytics system. *PVLDB*, 7(13):1766–1771, 2014.
51. T. Liu, J. Fan, Y. Luo, N. Tang, G. Li, and X. Du. Adaptive data augmentation for supervised learning over missing data. *Proc. VLDB Endow.*, 14(7):1202–1214, 2021.
52. T. Liu, J. Yang, J. Fan, Z. Wei, G. Li, and X. Du. Crowdgame: A game-based crowdsourcing system for cost-effective data labeling. In *SIGMOD*, pages 1957–1960, 2019.
53. 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.
54. 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.
55. 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.
56. L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. *CoRR*, abs/1611.02163, 2016.
57. M. Mirza and S. Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.
58. C. Olsson, S. Bhupatiraju, T. B. Brown, A. Odena, and I. J. Goodfellow. Skill rating for generative models. *CoRR*, abs/1808.04888, 2018.
59. 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.
60. Y. Park and J. Ghosh. Pegs: Perturbed gibbs samplers that generate privacy-compliant synthetic data. *Trans. Data Privacy*, 7(3):253–282, 2014.
61. N. Patki, R. Wedge, and K. Veeramachaneni. The synthetic data vault. In *DSAA*, pages 399–410, 2016.
62. PyTorch Developers. Tensors and dynamic neural networks in python with strong gpu acceleration. `https://pytorch.org`.
63. A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.
64. R. Ramakrishnan and J. Gehrke. *Database management systems (3. ed.)*. McGraw-Hill, 2003.
65. D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, pages 1278–1286, 2014.
66. 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.
67. R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 3–18. IEEE Computer Society, 2017.
68. I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.
69. S. Thirumuruganathan, S. Hasan, N. Koudas, and G. Das. Approximate query processing using deep generative models. *CoRR*, abs/1903.10000, 2019.
70. X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Trans. Knowl. Data Eng.*, 23(8):1200–1214, 2011.
71. L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou. Differentially private generative adversarial network. *CoRR*, abs/1802.06739, 2018.
72. L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. Modeling tabular data using conditional GAN. *CoRR*, abs/1907.00503, 2019.
73. L. Xu and K. Veeramachaneni. Synthesizing tabular data using generative adversarial networks. *CoRR*, abs/1811.11264, 2018.
74. J. Yang, J. Fan, Z. Wei, G. Li, T. Liu, and X. Du. Cost-effective data annotation using game-based crowdsourcing. *PVLDB*, 12(1):57–70, 2018.
75. L. Yang, S. Chou, and Y. Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. In *ISMIR*, pages 324–331, 2017.
76. Q. Yao, M. Wang, Y. Chen, W. Dai, Y.-F. Li, W.-W. Tu, Q. Yang, and Y. Yu. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.
77. L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017.
78. D. Zhang and A. Khoreva. PA-GAN: improving GAN training by progressive augmentation. *CoRR*, abs/1901.10422, 2019.
79. 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.
80. 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.
81. Z. Zhang, C. Yan, D. A. Mesa, J. Sun, and B. A. Malin. Ensuring electronic medical record simulation through better training, modeling, and evaluation. *J. Am. Medical Informatics Assoc.*, 27(1):99–108, 2020.
82. S. Zhao, Z. Liu, J. Lin, J. Zhu, and S. Han. Differentiable augmentation for data-efficient GAN training. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.



(a) Generator.



(b) Discriminator.

Fig. 18: GAN module implemented by CNN.

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 [19, 59], which is inspired by the well-known DCGAN [63], as illustrated in Figure 18. 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 deconvolution 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.,

$$\begin{aligned} \boldsymbol{h}_g^{l+1} &= \texttt{ReLU}(\texttt{BN}(\texttt{DeConv}(\boldsymbol{h}_g^l))), \\ \boldsymbol{t} &= \texttt{tanh}(\texttt{DeConv}(\boldsymbol{h}_g^L)), \end{aligned} \tag{16}$$

where DeConv is de-convolution function. Figure 18(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 18(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.,

$$\begin{aligned} \boldsymbol{h}_d^{l+1} &= \texttt{LeakyReLU}(\texttt{BN}(\texttt{Conv}(\boldsymbol{h}_d^l))), \\ f &= \texttt{sigmoid}(\texttt{BN}(\texttt{Conv}(\boldsymbol{h}_d^L))), \end{aligned} \tag{17}$$

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 [21, 72]. Figure 19 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

$$\boldsymbol{h}^{l+1} = \phi\big(\texttt{BN}(\texttt{FC}_{|\boldsymbol{h}^l| \to |\boldsymbol{h}^{l+1}|}(\boldsymbol{h}^l))\big), \tag{18}$$

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 [34].

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 4.1). 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.

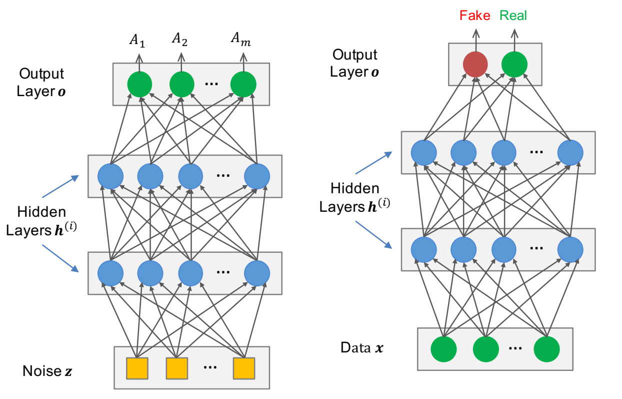
Figure 19(b) shows the NN structure of our discriminator D. Discriminator D is an MLP that takes a sample $\boldsymbol{t}$ as input, and utilizes multiple fully-connected layers and a sigmoid output layer to classify whether $\boldsymbol{t}$ is real or fake.

A.1.3 LSTM: recurrent neural networks.

Existing work also utilizes LSTM, a representative variant of RNN, to realize G [73]. 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 20. Let $\boldsymbol{h}^j$ and $\boldsymbol{f}^j$ respectively denote the hidden state and output of the LSTM at the j-th timestep. Then, we have

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

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



(a) Generator. (b) Discriminator.

Fig. 19: GAN module implemented by MLP.

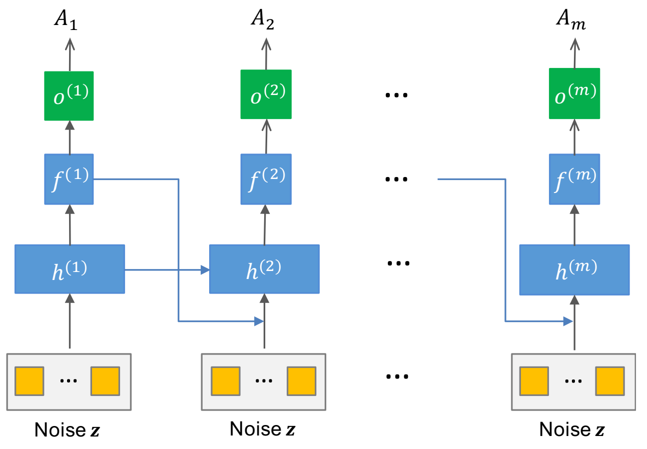


Fig. 20: Generator implemented by LSTM.

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 [68] to realize the discriminator D.

A.2 Training Algorithms

This section presents the pseudo-code of the training algorithms introduced in Sections 4.3, 5.3 and 5.1.

```
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 $G$, $D$
```

A.2.1 Vanilla GAN Training

We apply the vanilla GAN training algorithm [31] (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 [31]. 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 [42]. 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 [12] for training our data synthesizer (WTrain). Different from the original GAN, Wasserstein GAN removes the `sigmoid` function of D and changes the gradient optimizer from `Adam` to `RMSProp`. It uses the loss functions of D and G as

$$\mathcal{L}_D = -\mathbb{E}_{\boldsymbol{t}\sim p_{data}(\boldsymbol{t})}[D(\boldsymbol{t})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))]$$
$$L_G = -\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))]. \quad (19)$$

Algorithm 2 presents the training algorithm in Wasserstein GAN [12]. 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

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 $G$, $D$
```

in Equation (19). 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 [71] for details of c_g).

Algorithm 3: CTrain $(m, \alpha_d, \alpha_g, T, \Omega)$

```
Input: $m$: batch size; $\alpha_d$: learning rate of $D$; $\alpha_g$:
       learning rate of $G$; $T$: number of training
       iterations; $\Omega$: label domain in real data
Output: $G$: Generator; $D$: Discriminator
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 $(m, \alpha_d, \alpha_g, T_d, T_g, c_p, c_g, \sigma_n)$

```
Input: $m$: batch size; $\alpha_d$: learning rate of $D$; $\alpha_g$:
       learning rate of $G$; $T_d$: number of iterations for
       $D$; $T_g$: number of iterations for $G$; $c_p$: clipping
       parameter; $c_g$: bound on the gradient; $\sigma_n$:
       noise scale
Output: $G$: Generator; $D$: Discriminator
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 $G$, $D$ ;
```

B Additional Experiments

B.1 Detailed Dataset Information

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

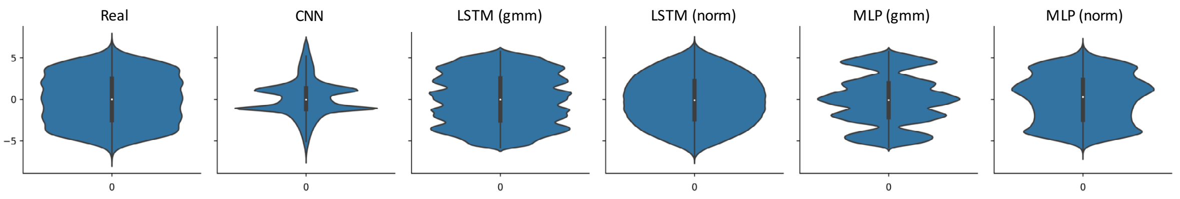


Fig. 21: Evaluating value distribution of synthetic numerical attributes (SDataNum).

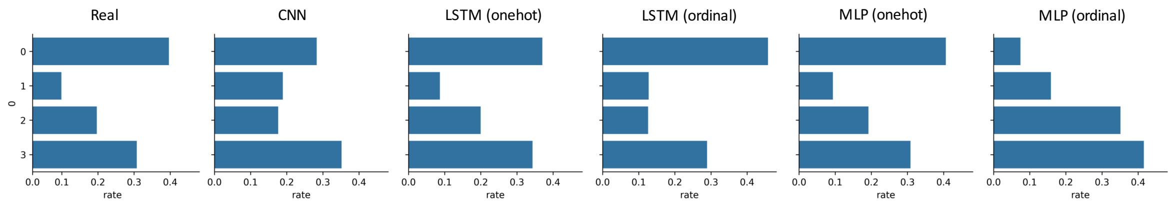
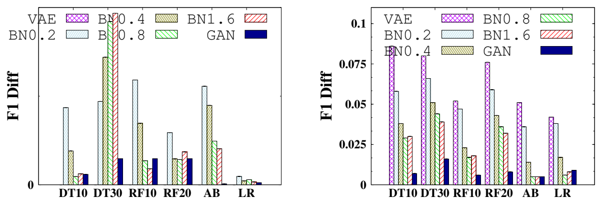


Fig. 22: Evaluating value distribution of synthetic categorical attributes (SDataCat).



(a) SDataNum dataset. (b) SDataCat dataset.

Fig. 23: Comparison of different approaches to relational data synthesis on data utility for classification.

(1) `HTRU2` dataset is a physical dataset that contains 17, 898 pulsar candidates collected during the High Time Resolution Universe Survey [6]. 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 [8]. 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 [9]. 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 [47]. 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.
(9) `Internet` dataset contains general demographic information on internet users that come from a survey conducted by the Graphic and Visualization Unit in 1997 [7]. It has 1 numerical attribute and 70 categorical attributes. Since there is no explicit label in the original dataset, we use it for evaluating performance of data synthesis on AQP.
(10) `Diabete` dataset collects 10-year data (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks [5]. It includes 11 numerical attributes and 44 categorical attributes to represent the features of diabetics and diagnoses. We use attribute `Readmitted` as label to predict days to inpatient readmission. The label distribution is also skewed: only 11% of the patients are readmitted in less than 30 days.

B.2 How to Create Simulated Datasets

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 [72] 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 $(\sigma_y) \sim \texttt{uniform}(0.5, 1)$. 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: `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.

B.3 Additional Evaluation on Mode Collapse

This section shows the results of GAN model training on various hyper-parameter settings on other datasets and the performance of SIMPLIFIED strategy to avoid mode collapse.

Figure 24 shows the results of GAN model training on various hyper-parameter settings on datasets `Census` and `SAT`, which is similar with the results in figure 8. Figure 25 and 26 show the performance of the SIMPLIFIED strategy on various hyper-parameter settings, we find that mode collapse can be effectively alleviated by replacing the simplified D, for example, on the `Adult` dataset, VTRAIN enables the LSTM-based generator to be more robust to hyper parameters and the chances of mode collapse are largely reduced. Moreover, compared with the MLP-based generator, it achieve much higher scores on F-measure.

B.4 Additional Evaluation on Synthetic Data Utility

This section shows the results of evaluation on synthetic data utility on the other datasets. Figure 27 shows the results of synthetic data utility on datasets `Anuran`, `Digits` and `HTRU2`, which has a similar trend to Figure 17, and we find that GAN-based framework still work well on the simulated datasets `SDataNum` and `SDataCat` from the results in figure 23.

Table 14: 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

B.5 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 [68] to realize D. Table 14 reports the experimental results. We can see that, compared with MLP-based discriminator D reported in Table 2(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.

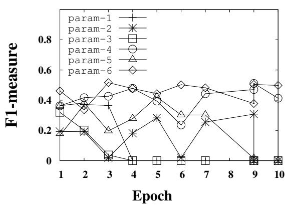
B.6 Synthetic Data Distribution

B.6.1 Evaluation on Data Transformation

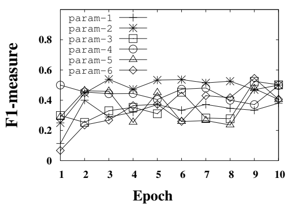
We also find that data transformation in preprocessing does affect the overall utility of synthetic data. GMM-based normalization and one-hot encoding achieve the best performance in most of the cases. To provide in-depth analysis, we further examine whether the value distribution of a synthetic attribute 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.

Figure 21 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 22 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.

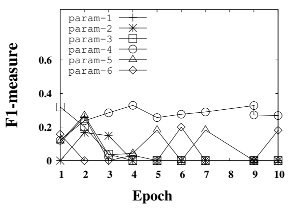
Finding: Data transformation does affect overall utility of synthetic data: GMM-based normalization performs better than simple normalization, especially for numerical attributes with multi-modal distribution; One-hot encoding is better than ordinal encoding for categorical attributes.



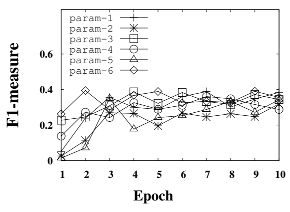
(a) LSTM-based G (`SAT`).



(b) MLP-based G (`SAT`).

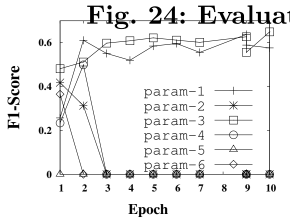


(c) LSTM-based G (`Census`).

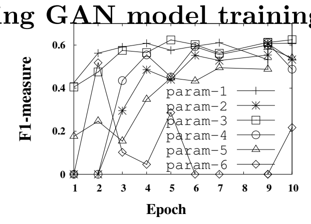


(d) MLP-based G (`Census`).

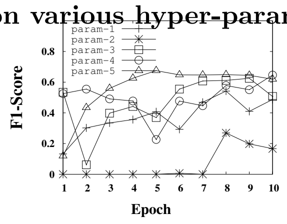
Fig. 24: Evaluating GAN model training on various hyper-parameter settings (1).



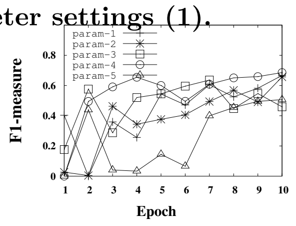
(a) Normal D (`Adult`).



(b) Simplified D (`Adult`).

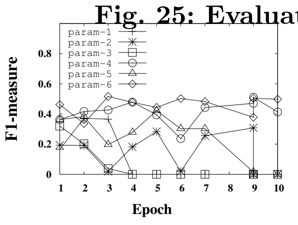


(c) Normal D (`CovType`).

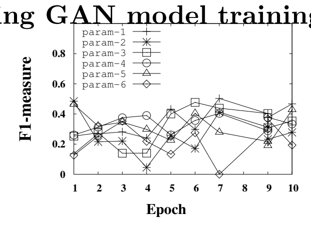


(d) Simplified D (`CovType`).

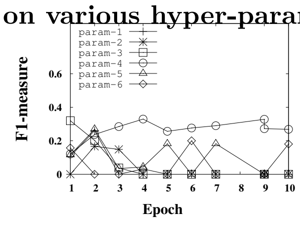
Fig. 25: Evaluating GAN model training on various hyper-parameter settings (2).



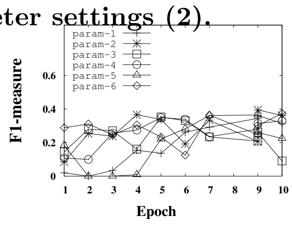
(a) Normal D (`SAT`).



(b) Simplified D (`SAT`).

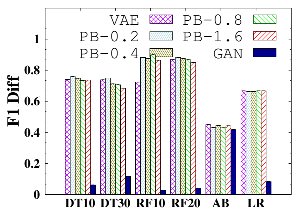


(c) Normal D (`Census`).

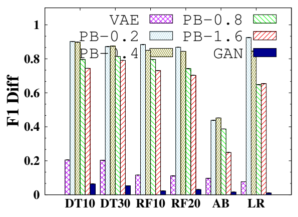


(d) Simplified D (`Census`).

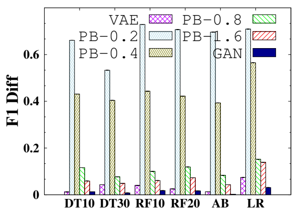
Fig. 26: Evaluating GAN model training on various hyper-parameter settings (3).



(a) `Anuran` dataset.



(b) `Digits` dataset.



(c) `HTRU2` dataset.

Fig. 27: Comparison of different approaches to relational data synthesis on data utility for classification.