Permutation-Invariant Tabular Data Synthesis

Yujin Zhu
Computer Engineering
TU Delft
Delft, the Netherlands
Y.Zhu-17@student.tudelft.nl

Zilong Zhao*
Computer Science
TU Delft
Delft, the Netherlands
Z.Zhao-8@tudelft.nl

Robert Birke
Computer Science
University of Torino
Torino, Italy
birke@ieee.org

Lydia Y. Chen
Computer Science
TU Delft
Delft, the Netherlands
LydiaYChen@ieee.org

Abstract—Tabular data synthesis is an emerging approach to circumvent strict regulations on data privacy while discovering knowledge through big data. Although state-of-the-art AI-based tabular data synthesizers, e.g., table-GAN, CTGAN, TVAE, and CTAB-GAN, are effective at generating synthetic tabular data, their training is sensitive to column permutations of input data. In this paper, we first conduct an extensive empirical study to disclose such a property of permutation invariance and an in-depth analysis of the existing synthesizers. We show that changing the input column order worsens the statistical difference between real and synthetic data by up to 38.67% due to the encoding of tabular data and the network architectures. To fully unleash the potential of big synthetic tabular data, we propose two solutions: (i) AE-GAN, a synthesizer that uses an autoencoder network to represent the tabular data and GAN networks to synthesize the latent representation, and (ii) a feature sorting algorithm to find the suitable column order of input data for CNN-based synthesizers. We evaluate the proposed solutions on five datasets in terms of the sensitivity to the column permutation, the quality of synthetic data, and the utility in downstream analyses. Our results show that we enhance the property of permutation-invariance when training synthesizers and further improve the quality and utility of synthetic data, up to 22%, compared to the existing synthesizers.

Index Terms—GAN; Autoencoder; Tabular data synthesis; Column permutation invariance

I. INTRODUCTION

As one of the most common data types, tabular data are ubiquitous in the operation of banks, governments, hospitals, and manufacturers, which lay the foundations of modern society [1]. The synthesis of realistic tabular data, i.e., generating synthetic tabular data that are statistically similar to the original data, is crucial for many applications, such as data augmentation [2], imputation [3], [4], and re-balancing [5]–[7]. Another important application is to use generated data to overcome data sharing restrictions [8] caused by regulations on data protection and privacy, such as European General Data Protection Regulation (GDRP) [9].

Synthesizing realistic tabular data is a non-trivial task. Compared to image and language data, tabular data are heterogeneous - they contain dense continuous features and sparse categorical features. The former can have multiple
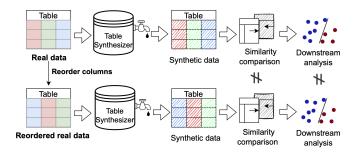
*Contact author.

Fig. 1. Illustration of lacking column permutation invariance of training tabular data synthesizer. This leads to dissimilarity between real and synthetic data and different downstream analysis results.

modes, whereas the latter often have highly-imbalanced distributions [10]. Furthermore, the correlation between features in tabular data is often more complex than the spatial or semantic correlation in image or language data. Related features can be far apart spatially, and multiple features can be inter-correlated.

Although the state-of-the-art AI-based tabular data synthesizers show promising results [8], [11]–[13], they suffer from a critical and undiscovered limitation: their training is sensitive to column permutations. Changing the input column order during training influences the quality of the synthetic data, e.g., statistical similarity with real data (see Figure 1). Theoretically, reordering the columns of the training input shall not change the capability of synthesizers because the position of columns does not imply any semantic information. We call this property *column permutation invariance*, i.e., the output of generative models is invariant to the column order of their training input. However, our extensive empirical analysis shows that the statistical difference between real and synthetic data increases by up to 38.67% after changing the input column order. The main reason is the sparsity issue caused by one-hot data encoding for categorical features and mode-specific normalization for numerical features.

In this paper, we address the limitation of being sensitive to input column permutations and striking a tradeoff between the quality of synthetic data and the training time by two approaches. Our first solution is to leverage an autoencoder [14] to improve the representation of tabular data and then design a Wasserstein GAN with gradient penalty (WGAN-GP) [15] based on fully connected networks to synthesize the latent representation. This solution is named AE-GAN. Our second

solution is to leverage a feature sorting algorithm that is capable of exploring the correlation among columns and provides the permutation that enhances the synthesizer training, especially for the ones based on convolutional neural networks.

We evaluate both solutions on five real-world machine learning datasets against four state-of-the-art tabular data synthesizers: table-GAN [8], CTGAN [11], CTAB-GAN [12], and TVAE [11]. The results show that, compared to the baselines, AE-GAN makes a better trade-off among more permutation-invariance, generating high-quality data, and leading to accurate downstream analyses. Moreover, its training time is significantly shorter than CTAB-GAN, the best performing model in synthesizing realistic tabular data. Besides, after applying the feature sorting algorithm to CTAB-GAN, the statistical difference between real and synthetic data is improved by 22% averaged across five datasets.

The contributions of this paper are as follows:

- The first empirical study to analyze sensitivity to column permutations for training tabular data synthesizer which reveals its root cause, i.e., data representation and sparsity.
- A novel tabular data synthesizer, AE-GAN, which effectively achieves high permutation invariance and good quality of synthetic data in terms of statistical similarity and machine learning utility.
- A feature sorting algorithm for tabular data synthesizers, which helps preserve the relation between highly-correlated features.

II. BACKGROUND AND RELATED WORK

In this section, we first provide the background on two methods, GAN and autoencoder, which are the cores of the prior art. Then, we compare the related studies in terms of their network structures and data encoding schemes.

A. Generative Adversarial Network

Generative adversarial networks (GAN) are a recently developed algorithm [16] for synthetic data generation. A GAN consists of two components: a generator (G) that learns to produce realistic synthetic data, and a discriminator (D) that tries to distinguish real data from synthetic (fake) data. Both G and D are neural networks, e.g., Fully-Connected Networks (FCNs) or Convolutional Neural Networks (CNNs). In the training process, G and D play an adversarial game described as follows:

$$\min_{G} \max_{D} V(G, D) = \mathbb{E}[log D(x)]_{x \sim p_{data}(x)} \\ + \mathbb{E}[log(1 - D(G(z)))]_{z \sim p(z)}, \quad (1)$$

where x is the real sample, z is the random input signal given to G, $G(z)$ is the synthetic sample, and $D(\cdot)$ is the probability of a sample being real from the perspective of D. The goal of G is to minimize the chance that its generated samples are identified as synthetic, whereas D maximizes the chance of correctly distinguishing real and synthetic samples.

B. Autoencoder

An autoencoder (AE) is an unsupervised learning algorithm that learns a mapping from high-dimensional inputs to low-dimensional representations [14], [17], namely latent vectors. It consists of two models, an encoder (Enc) and a decoder (Dec). Enc takes a high-dimensional input and compresses it to a latent vector, and Dec uses the latent vector to reconstruct the original input. Enc and Dec are trained as a whole and penalized for creating output that deviates from the input. The loss function is defined as follows:

$$\min_{\theta, \phi} L(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} ||x_i - Dec_\theta(Enc_\phi(x_i))||_2^2, \quad (2)$$

where θ and ϕ are the parameters of Dec and Enc, x_i is a high-dimensional input, $Enc_\phi(x_i)$ is the latent vector, $Dec_\theta(Enc_\phi(x_i))$ is the reconstructed input, and N is the total number of samples.

C. Tabular data synthesizers

We focus on deep-learning approaches for tabular data synthesis and skip the discussion of classical methods such as Copulas [18], [19] and Bayesian Networks [20]. Table I summarizes the recently developed deep learning methods for tabular data synthesis in terms of models, network architecture, and datasets. MedGAN [21] is designed for aggregated electronic health records (EHRs), which only have count and binary features. Since EHRs are high-dimensional and sparse [22], medGAN uses a pre-trained autoencoder to learn compact representations of the input data and thereby simplifies the GAN's task. MedGAN is improved by [22], where the standard GAN loss is replaced by Wasserstein loss with gradient penalty, and the new model is named medWGAN. However, different from AE-GAN, medGAN and medWGAN are limited in generalizing to real-world scenarios because they only consider count and binary features.

A few recent tabular data synthesizers are suitable for general data types, including table-GAN [8], CTGAN [11], TVAE [11], and CTAB-GAN [12]. CTGAN, TVAE, and CTAB-GAN use Variational Gaussian Mixture (VGM) to encode numerical features and one-hot encoding for categorical features. Moreover, CTAB-GAN defines the *mixed* datatype and proposes a new encoding method. In addition, CTGAN, TVAE, and CTAB-GAN adopt the training-by-sampling technique to handle highly-imbalanced distributions. Despite their effectiveness in tabular data synthesis, these models overlook and do not abide by the key property of column permutation invariance.

D. Column permutation invariance

In computer vision similar concepts have been brought up and investigated, including *permutation invariance* [23], [24], *translation invariance* [25]–[27], and *translation equivalence* [28]. Permutation invariance means that the output of a neural network stays the same despite permutations of its input. For example, the classification of an image should not change after adjusting the object location in the image.

Method	Model design	Network	Data
medGAN [21]	AE + GAN	FCN	Medical records
table-GAN [8]	DCGAN + Classifier	CNN	General
medWGAN [22]	AE + WGAN-GP	FCN	Medical records
CTGAN [11]	Conditional WGAN-GP	FCN	General
TVAE [11]	Conditional VAE	FCN	General
CTAB-GAN [12]	Conditional DCGAN + Classifier	CNN, FCN	General

TABLE II
TABLEGAN EXPERIMENT RESULTS: WASSERSTEIN-1 DISTANCE
BETWEEN REAL AND SYNTHETIC DATA

Dataset	Column order			Max diff. (%)
	Original order	Order by type	Order by corr.	
Loan	2.062	**2.047**	2.066	**0.93%**
Adult	12.153	12.563	**11.512**	9.13%
Credit	0.420	0.410	**0.403**	4.22%
Covtype	**1.282**	1.284	1.345	4.91%
Intrusion	6.486	5.896	**5.645**	14.90%
Avg.	4.481	4.440	**4.194**	6.82%

Motivated by that, we define column permutation invariance in tabular data synthesis as follows. The performance of a tabular data synthesizer should not be affected by permutations on the input column order. To the best of our knowledge, column permutation invariance has not been researched by the prior art on tabular data synthesis.

III. EMPIRICAL ANALYSIS

In this section, we analyze the column permutation invariance property of the state-of-the-art tabular data synthesizers, with a particular focus on its root causes.

A. Pitfall of CNNs for tabular data synthesis

Initially designed for images, CNNs use a set of convolution kernels to slide over the input feature space, abstract high-dimensional features, and then aggregate them into knowledge about the input. Due to the limited kernel size, CNNs only learn local relations between neighboring features and fall short to capture global dependencies.

The focus of CNNs on local relations hinders high quality tabular data synthesis. In contrast to image data, tabular data do not have necessarily strong local relations. Highly-correlated features can be very far apart, and their dependencies can be complex and irregular [29]. These characteristics make modeling tabular data extra challenging for CNNs [30], [31], despite their remarkable performance in many machine learning tasks [32].

Since CNNs capture mainly local relations, CNN-based tabular data synthesizers are sensitive to column permutations. We use table-GAN to verify this assumption. We test it with five datasets arranged using three column orders, namely the original order, order by type, and order by correlation. Order by type means putting all continuous columns on the left of the table, and all categorical columns on the right. Order by correlation means placing highly-correlated columns

on the left and weakly-correlated columns on the right. For each order, we train the model separately and calculate the Wasserstein-1 distance (WD) between real and synthetic data. Every experiment is repeated 5 times. Table II summarizes our results. The best, i.e. lowest, distance values are highlighted in bold. The last column shows the maximum WD change in percent across all three column permutations. The results show that table-GAN is most sensitive on the Intrusion dataset with a maximum difference in WD of 14.90%.

B. Sparsity v.s. sensitivity

Efficient representation of categorical features is one of the main challenges in tabular data synthesis. In the state-of-the-art, table-GAN uses label encoding to transform categorical features into numerical ones and normalizes them with min-max normalization. This method often leads to sub-optimal performance due to the artificial order in categorical features [33]. In contrast, CTGAN, CTAB-GAN, and TVAE use one-hot encoding to represent categorical features. Despite its simplicity and effectiveness, one-hot encoding introduces many zeros to the input data and thus increases sparsity.

Representing numerical features in tabular data is relatively straightforward. The most common method is mapping them into [-1, 1] with min-max normalization. However, the authors of CTGAN, TVAE, and CTAB-GAN adopt *mode-specific normalization*, which uses Variational Gaussian Mixture to represent multi-modal numerical features. Although this method improves the quality of the synthetic data, we find it leads to sparse input because one-hot encoding is used to represent multiple modes.

Our experiments show that sparse tabular data causes sensitivity to column permutations. We compare CTAB-GAN, a model using one-hot encoding and mode-specific normalization, with table-GAN, a model using label encoding and min-max normalization. Note that label encoding and min-max normalization do not change the dimensionality of the input data, whereas one-hot encoding and mode-specific normalization increase sparsity. Table III shows our experiment results of CTAB-GAN. Compared to table-GAN (see Table II), CTAB-GAN synthesizes more realistic data (shown by a lower WD which is about $1/4$ of table-GAN's WD), but with higher sensitivity to column permutations on all datasets. The average maximum change in WD across the five datasets is 38.67%, whereas in table-GAN it is 6.82%.

To further highlight the sparsity of the encoded data we visualize the input of table-GAN and CTAB-GAN in Figure 2. We note that the sparsity is determined by the sum of all levels of all variables, i.e., the number of modes per continuous variable and the discrete levels per discrete variable. We reshape each row of a table into a square matrix to make it compatible with CNNs. In table-GAN, one row of the Adult dataset is represented by a 4×4 matrix, but in CTAB-GAN a 24×24 matrix is needed due to one-hot encoding and mode-specific normalization which increases the number of zeros, i.e. purple matrix cells in the figure accounting for roughly 97% area. This increase in sparsity makes CTAB-GAN more

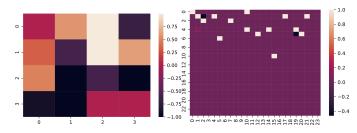| Dataset	Column order			Max diff. (%)
	Original order	Order by type	Order by corr.	
Loan	0.356	0.283	**0.216**	64.81%
Adult	1.517	**0.934**	1.203	62.42%
Credit	**0.115**	0.144	0.137	25.22%
Covtype	0.539	**0.514**	0.583	**13.42%**
Intrusion	**2.668**	3.401	2.831	27.47%
Avg.	1.039	1.055	**0.994**	38.67%

Fig. 2. Visualization of the encoded input to table-GAN (left) and to CTAB-GAN (right) using Adult dataset.

sensitive to column permutations than table-GAN since the average distance between related columns (pixels) is increased.

C. FCNs column permutation invariance

Theoretically, fully-connected networks (FCNs) should be robust to column permutations because all features are connected. However, we find FCNs are not fully permutation invariant. Tests with CTGAN and TVAE, two FCN-based tabular data synthesizers, on five datasets show an average WD of 1.87 with an average maximum change of 18.62% for CTGAN and 1.80 with 14.89% for TVAE (details skipped due to space constraints). Overall, the state-of-the-art tabular data synthesizers either provide the high quality synthetic data or are resilient to column order permutations, struggling to make a good trade-off.

IV. AE-GAN

We propose AE-GAN, a GAN-based tabular data synthesizer, which aims to improve the resilience to the input column permutations by using latent representations of tabular data via an autoencoder. Figure 3 shows the overall architecture and data flow of AE-GAN. It has five components: Encoder (Enc), Decoder (Dec), Generator (G), Discriminator (D), and Classifier (C). The main objective of the encoder and the decoder is to find a more compact latent representation of the input data, which follow the data encoding scheme proposed below. Once the autoencoder is trained, the encoded latent vector and the random noise vector are used as input to train the GAN. The GAN aims to generate a synthetic latent vector having high similarity to the original one. During training the classifier provides additional feedback to ensure the semantic integrity of synthetic data. We explain the design choice of each component in the following.
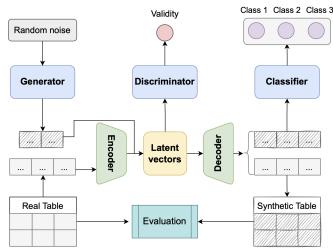

Fig. 3. The overall architecture and data flow of AE-GAN.

A. Data representation

Following [11], we use mode-specific normalization for numerical features and one-hot encoding for categorical features. Mode-specific normalization preserves the multi-modal distribution of numerical features and improves the performance of tabular data synthesizers [11], [12]. One-hot encoding is a simple yet effective way to convert categorical values to numerical ones without losing too much information. Certainly, one-hot encoding and mode-specific normalization causes sparse input, but the autoencoder in AE-GAN maps the sparse encoded input into compact latent vectors solving this issue.

B. Encoder and decoder

Since we identify sparsity as one of the main reasons for tabular data synthesizers' sensitivity to column permutations, one natural solution is to use an autoencoder to extract the features of tabular data and compress them into compact latent vectors. Such an advantage can apply to all kinds of tabular data synthesizers.

We use two three-layer fully connected networks as Enc and Dec. Based on our study of mainstream open-source implementations of autoencoders [34]–[36], fully-connected networks with 2-4 layers are common choices for autoencoders.

Another important design choice is the length of the latent vector, i.e., the output size of Enc and the input size of Dec. This determines the autoencoder's capacity to represent high-dimensional data. We choose this parameter based on the size of the input dataset. For datasets with a large number of columns, we increase the length of the latent vector to ensure that complex relations between columns can be well represented, therefore helping the generator to synthesize realistic data.

C. Generator and discriminator

The core of AE-GAN is a GAN, which has two competing networks, namely the generator, G, and the discriminator, D.
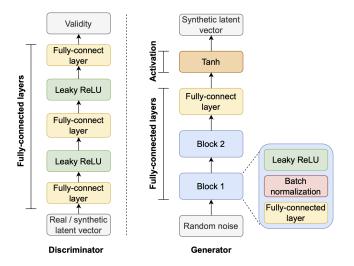
Fig. 4. Architectures of the discriminator and the generator in AE-GAN.

Figure 4 shows their architectures. Both G and D have three fully-connected layers which are more resilient to permutations of elements in the (more compact) latent vector. In the discriminator, the fully connected layers are followed by leaky ReLU activation and the final output is the validity of the input. In the generator, the fully-connected layers are followed by batch normalization and leaky ReLU activation, and the final output is the synthetic latent vector. The design is based on the architecture of WGAN-GP [37].

D. Auxiliary classifier

To enhance the training of G and thus the quality of the generated tabular data, we introduce an auxiliary classifier C to the GAN. This design is inspired by [8], where an auxiliary classifier is added to maintain the semantic consistency of synthetic data. Note that the input to C is the reconstructed data from Dec, rather than the latent data from Enc and G because we want C to learn the semantic relations between columns directly. Specifically, given a categorical target column with several categories, C learns to classify which category a sample belongs to according to the columns other than the target column. This is how C differs from the D: D determines the "realness" of a sample based on all columns in the latent space, whereas C learns the relationship between the target column and all other columns in the reconstructed space. By combining C with the GAN, we simultaneously leverage the flexibility of unsupervised training with the control provided by supervised training, thereby improving the quality of the synthetic data.

E. Loss functions

The training of AE-GAN requires four loss functions: autoencoder loss $\mathbb{L}_{AE}$, generator loss $\mathbb{L}_G$, discriminator loss $\mathbb{L}_D$, and classifier loss $\mathbb{L}_C$.

1) Autoencoder loss: Autoencoder loss is the reconstruction loss, i.e., the element-wise mean squared error between its input and reconstructed output. It is defined as follows:

$$\mathbb{L}_{AE} = \mathbb{E}||x - \tilde{x}||_2^2, \qquad (3)$$

where x and $\tilde{x}$ are the input and the reconstructed output.

2) Generator loss: The generator receives feedback from both the discriminator and the classifier. Therefore, its loss function is the sum of: discriminator feedback $\mathbb{L}_G^D$ and classifier feedback $\mathbb{L}_G^C$.

$$\mathbb{L}_G = \mathbb{L}_G^D + \mathbb{L}_G^C \qquad (4)$$

Discriminator feedback is the validity of synthetic samples:

$$\mathbb{L}_G^D = -\mathbb{E}[D(G(z))], \qquad (5)$$

where $G(z)$ is the generator output and $D(G(z))$ is the discriminator output.

Classifier feedback is the cross entropy between the predicted value and the actual value of the target column:

$$\mathbb{L}_G^C = H(m, m'), \qquad (6)$$

where m and m' are the actual and predicted values of the target column, and $H(\cdot)$ is the cross entropy operator.

3) Discriminator loss: The discriminator loss measures how well it differentiates the real samples and the synthetic samples. We use Wasserstein loss with gradient penalty to improve the training stability and alleviate the vanishing gradient problem of GANs [15]. It is calculated by:

$$\mathbb{L}_D = -\mathbb{E}[D(x) - D(G(z)) - \lambda \cdot (||\nabla D(\hat{x})||_2 - 1)^2], \quad (7)$$

where $D(x), D(G(z))$ and $D(\hat{x})$ are the discriminator output on real samples, synthetic samples, and the interpolates between real and synthetic samples. λ is the gradient penalty coefficient, $\nabla D(\hat{x})$ is the gradient of $D(\hat{x})$ on $\hat{x}$.

4) Classifier loss: The classifier loss also has two parts: loss on real samples $\mathbb{L}_C^R$ and loss on synthetic samples $\mathbb{L}_C^S$.

$$\mathbb{L}_C = \mathbb{L}_C^R + \mathbb{L}_C^S. \qquad (8)$$

The calculation of $\mathbb{L}_C^R$ and $\mathbb{L}_C^S$ are similar to $\mathbb{L}_G^C$.

F. Training algorithm

We test two training strategies: disjoint training and joint training. For disjoint training we first train the AE until convergence and then train the GAN with the classifier while utilizing the compression power of the AE. For joint training, inspired by TimeGAN [38] and the hypothesis of possible training synergies, we first pre-train the autoencoder for a certain number of epochs and then co-train it with the GAN and the classifier. Ablation tests, details in Section VI-D, show that disjoint training achieves lower training losses. Hence we use disjoint training for all results if not specified.
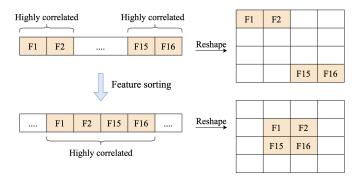
Fig. 5. **Top:** The highly correlated features, $F1, F2, F15$ and $F16$ are on the border of the input matrix, suffering from the boundary effects of CNNs. **Bottom:** After feature sorting, $F1, F2, F15$ and $F16$ are at the center of the input matrix.

V. FEATURE SORTING ALGORITHM

Although features in tabular data are often heterogeneous, some are highly correlated. However, as discussed in Section III-A, CNN-based tabular data synthesizers often fail to capture these correlations due to the distance between correlated columns.

A simple solution for this problem is to put highly-correlated features together, such that a convolution kernel can capture them simultaneously.

The location of features also matters. In a convolution process, features on the border of the input matrix are convoluted fewer times than features within the border, leading to potential loss of information. This is called *boundary effects* in image processing [39], [40] which can lead to statistical biases in finite-sampled data [25], [41], [42]. To better capture the correlation between features, one must also carefully choose the location of high-correlated features.

To address both issues our solution is to group highly-correlated features together in the middle of the table and mitigate the boundary effect. Figure 5 illustrates the base of our idea. Since each row must be reshaped into a square matrix to be fed into a CNN, we have to carefully sort the features such that they end up in the middle after reshaping.

Although this idea seems straightforward, one complication arises due to the encoding of features. Variational Gaussian Mixture [11] and One-hot encoding require multiple columns to represent one feature. Consequently, each feature may occupy a different number of columns. To put highly correlated features in the middle after encoding, we must consider the length of each encoded feature.

We developed a feature sorting algorithm that groups highly-correlated features and puts them in the middle, see Algorithm 1. We first pick the most correlated feature pairs and then add other features to their left or right side. We have two counters, c_{left} and c_{right}, for columns taken by features added to the left and the right side. Before adding a feature, we compare c_{left} and c_{right}, and then add it to the side with fewer columns. Thereby we ensure that the highly correlated features stay in the middle even after encoding.

Algorithm 1 Feature Sorting Algorithm
___
Input: Original Table $T_o = \{F_0, F_1, ..., F_n\}$
Output: Sorted Table T_{sorted}
___
1: $T_{sorted} \leftarrow \{\}$
2: $c_{left}, c_{right} \leftarrow 0$ ▷ No. columns added to the left / right of T_{sorted}
3: $corr \leftarrow [\ ]$ ▷ Pair-wise correlation / association
4: **for** all possible pairs of features in T_o **do**
5:     Calculate the absolute value of their correlation / association and save it in $corr$
6: **end for**
7: **while** $length(T_{sorted}) \neq length(T_o)$ **do**
8:     Find the largest value v in $corr$
9:     Find the corresponding pair of features $\{F_x, F_y\}$
10:     $F_{new} \leftarrow \{F_x, F_y\} - \{F_x, F_y\} \cap T_{sorted}$ ▷ Feature(s) not yet in T_{sorted}
11:     $c \leftarrow$ No. columns occupied by F_{new} after encoding
12:     **if** T_{sorted} is empty **then**
13:         $T_{sorted} \leftarrow \{F_x, F_y\}$ ▷ Add the first pair
14:     **else**
15:         **if** $c_{right} < c_{left}$ **then**
16:             $T_{sorted} \leftarrow T_{sorted} + F_{new}$ ▷ Add the new feature(s) to the right
17:             $c_{right} \leftarrow c_{right} + c$
18:         **else**
19:             $T_{sorted} \leftarrow F_{new} + T_{sorted}$ ▷ Add the new feature(s) to the left
20:             $c_{left} \leftarrow c_{left} + c$
21:         **end if**
22:     **end if**
23:     Remove v from $corr$
24: **end while**
25: **return** T_{sorted}

VI. EXPERIMENTAL EVALUATION

A. Experimental Setup

1) Datasets: We use five tabular datasets that are common in the machine learning community. Table IV summaries their main statistics. The Loan dataset[1] contains the demographic information about bank customers and their response to a personal loan campaign. The Adult dataset[2] has many census data and is used to predict whether the income of an adult exceeds $50k/year. The Credit dataset[3] consists of anonymized credit card transactions labeled as fraudulent or genuine. The Intrusion dataset[4] has encrypted WiFi traffic records and classifies whether a record is from an unmanned aerial vehicle. The Covtype dataset[5] contains the cover type of forests and the related geographical information. Every dataset has a

---

[1]https://www.kaggle.com/code/pritech/bank-personal-loan-modelling/data
[2]https://archive.ics.uci.edu/ml/datasets/Adult
[3]https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
[4]https://archive.ics.uci.edu/ml/datasets/Unmanned+Aerial+Vehicle+%28U AV%29+Intrusion+Detection
[5]https://archive.ics.uci.edu/ml/datasets/Covertype

TABLE IV
STATISTICS OF DATASETS

Datasets	# Continuous columns	# Categorical columns	# Columns after encoding	# Samples
Loan	5	8	55	5k
Adult	5	9	151	48k
Credit	30	1	301	50k
Covertype	10	45	205	50k
Intrusion	22	20	322	50k

TABLE V
AE-GAN EVALUATION RESULTS AGAINST THE STATE-OF-THE-ART. FOR ALL METRICS, A LOWER VALUE IS BETTER.

Model	Sensitivity to permutations	Stat. diff.		ML utility diff.	Training time (mins)
		WD	Dif. Corr.		
table-GAN	**6.82%**	4.481	3.651	21.14%	**1.31**
CTAB-GAN	38.67%	**1.039**	**1.905**	**9.11%**	63.70
CTGAN	18.62%	1.857	3.079	14.99%	10.47
TVAE	17.29%	1.723	2.848	12.84%	7.00
AE-GAN	11.71%	2.699	2.331	9.98%	9.89

target column for classification tasks. Due to the limitation of computational resources, we randomly select 50k samples from the Credit, Intrusion, and Covtype datasets.

2) Baselines: Four state-of-the-art tabular data synthesizers are selected as the baseline models, namely table-GAN [8], CTGAN [11], TVAE [11], and CTAB-GAN [12]. We use the same hyperparameters as the original papers, and every experiment is repeated three times to obtain reliable results.

3) Computational Environment: We implemented our proposed solutions using Pytorch on a server equipped with an Intel(R) Core(TM) i9-10900KF CPU @3.70GHz and a GeForce RTX 2080 Ti GPU.

B. Evaluation Metrics

Our evaluation of tabular data synthesizers focuses on the statistical difference and machine learning utility difference between real and synthetic data. Two metrics quantify the statistical difference.

Wasserstein-1 Distance (WD) measures the difference between two continuous/discrete 1-dimensional distributions. We use this metric to compare the per features difference between real and synthetic data.

Difference in Correlation Matrix (Dif. Corr.) measures how well the cross-column correlations[6] are captured by a tabular data synthesizer. We calculate the difference between the correlation matrices of the real and synthetic table as follows:

$$Dif.\ Corr. = \sqrt{\sum_{i,j}(Corr_{i,j}^R - Corr_{i,j}^F)^2}, \qquad (9)$$

where $Corr_{i,j}^R$ and $Corr_{i,j}^F$ are the correlation coefficients between features i and j in the real and synthetic correlation matrices.

We measure machine learning utility as the performance differences of machine learning models trained on the real and synthetic data. Specifically, we first train four machine learning models with real and synthetic data separately. Then we obtain their average prediction accuracy and compute the difference. The difference is small if the synthetic data has high machine learning utility.

[6]Note that we use "correlation" as a general term. For two numerical features, it refers to the Pearson correlation coefficient; for two categorical features, it is their Cramer's V; and for a categorical features and a numerical features, it means their correlation ratio.

C. AE-GAN

AE-GAN is evaluated on four aspects: column permutation invariance, statistical similarity and machine learning utility of the synthetic data, and training time. We aim to verify if AE-GAN is robust to column permutations and achieves good synthesis quality compared with the state-of-the-art. Additionally, we evaluate the scalability of AE-GAN by analyzing its training time. Table V summarises the results averaged on five datasets.

Column Permutation Invariance. Similar to our empirical analysis, we arange training data in three different orders, i.e., original order, order by type, and order by correlation, and test the performance of AE-GAN. The second column of Table V shows the sensitivity to column permutations of the baseline models and AE-GAN, averaged on five datasets. We find AE-GAN ranks second in permutation invariance among the five models. Table-GAN is the most permutation-invariant model because it does not have the sparsity issue caused by one-hot encoding and mode-specific normalization. In contrast, CTAB-GAN, TVAE, CTGAN and AE-GAN adopt one-hot encoding for categorical features and mode-specific normalization for numerical features and thus have sparse input, leading to higher sensitivity. However, since AE-GAN has an autoencoder to compress the input, it is more robust to column permutations than CTAB-GAN, TVAE, and CTGAN.

Synthesis Quality Comparison. We evaluate the quality of the synthesized data with two metrics: statistical difference and ML utility difference between real and synthetic data. Table V shows that CTAB-GAN is the best model in synthesis quality because its synthetic data have the lowest statistical difference and ML utility difference compared with real data. Table-GAN is the worst among all models. In the rest three models, AE-GAN is better than CTGAN and TVAE on Dif. Corr, but worse on WD. However, the ML utility of AE-GAN is better than CTGAN and TVAE. The results show that the autoencoder of AE-GAN helps preserve the correlation between different features, but for each feature, the statistical difference between real and synthetic data may increase due to the information loss caused by data compression. Future work may look into optimizing the trade-off between the performance boost for GAN and the compression loss caused by compact representations.

Training Time Analysis. A model with a short training time can scale up to large datasets. It also requires fewer

hardware resources than slow models given the same input. We compare the scalability of AE-GAN with the baseline models by analyzing their total training time. Table V shows that AE-GAN is faster than CTGAN and CTAB-GAN, but slower than table-GAN and TVAE. Table-GAN has the shortest training time because of its simple data representation, which leads to small input size. TVAE is faster than AE-GAN because AE-GAN has an auxilliary classifier. Nonetheless, AE-GAN is significantly faster than CTAB-GAN, speeding up as much as 6 times.

Summary. AE-GAN achieves the best tradeoff between permutation invariance, synthesis quality, and training time compared with state-of-the-art tabular data synthesizers. It is more permutation-invariant than CTAB-GAN, CTGAN, and TVAE, and it has better synthesis quality than table-GAN, CTGAN, and TVAE in terms of ML utility. Although table-GAN is less sensitive to column permutations and takes less time to train than AE-GAN, its synthesis quality is much worse. In a similar vein, although CTAB-GAN is better than AE-GAN in synthesis quality, it is much slower and more sensitive to column permutations. Compared with CTGAN and TVAE, AE-GAN is more permutation-invariant, has better ML utility, and takes a similar time to train.

D. Ablation Study

We conduct an ablation study to understand the influence of the design choices we made with AE-GAN. We change the data representations, model architecture, and training algorithm to test their effect. Table VI summarizes the results of the ablation study.

Without Mode-Specific Normalization (MSN). We use mode-specific normalization in AE-GAN to normalize numerical features. Although it preserves the multi-model distribution of numerical features, it increases the sparsity in training data. We replace it with min-max normalization to understand its effect. Table VI shows that after removing mode-specific normalization, the WD becomes worse on all datasets, meaning that mode-specific normalization improves the synthesis quality. However, it also makes AE-GAN more sensitive to column permutations. After removing it, the sensitivity to column permutations decreases on the Loan, Credit, and Covtype datasets. In conclusion, mode-specific normalization increases sensitivity to column permutations, but it improves synthesis quality.

Without One-hot and Mode-Specific Normalization. To further reduce sparsity in the input data, we remove one-hot encoding and mode-specific normalization together. Similar to table-GAN, we pre-process categorical and numerical features using min-max normalization. We found that the WD is worse than only removing mode-specific normalization, which proves that one-hot encoding can enhance synthesis quality. Moreover, the sensitivity to column permutations decreases on all datasets except the Adult dataset after removing one-hot encoding and mode-specific normalization, especially on datasets with a high proportion of categorical features such as

the Covtype and Intrusion datasets. The results verify again that reducing sparsity can enhance permutation invariance.

Without Auxiliary Classifier. We use an auxiliary classifier to improve the synthesis quality of AE-GAN. After removing the auxiliary classifier, the Wasserstein distance between real and synthetic data worsens on all datasets except the Loan dataset. Overall, the average WD on five datasets increases from 2.669 to 2.773 after removing auxiliary classifier, showing that the classifier improves synthesis quality.

Co-training AE and GAN. The AE and GAN in AE-GAN are trained separately. To study whether co-training AE and GAN can improve the synthesis quality, we first pre-train the AE for 300 epochs and then train it together with GAN. Surprisingly, the results show that co-training makes the synthesis quality worse. We find that the training loss of AE is already low after pre-training. However, during co-training, the feedback from GAN increases AE's loss and makes it unstable.

E. Feature sorting algorithm

We evaluate the proposed feature sorting algorithm on table-GAN and CTAB-GAN, two CNN-based tabular data synthesizers, because this algorithm is designed to alleviate the limitations of CNN as explained in Section V.

Table-GAN. Table VII shows the effect of the feature sorting algorithm on table-GAN. A negative change in Dif. Corr. or WD means the difference between synthetic and real data becomes smaller. That is, the feature sorting algorithm helps tabular data synthesizers generate more realistic data. The results show that the feature sorting algorithm works best on the Credit dataset, where Dif. Corr. and WD are decreased by 12% and 4%. It also improves the results on the Intrusion dataset, where WD is reduced by 16%, whereas Dif. Corr slightly increases by 3%. However, it does not influences much the Loan and Covtype datasets, where the Dif. Corr. and WD change less than 5%. Moreover, the results on the Adult dataset become worse, with Dif. Corr and WD increase by 24% and 3%.

The algorithm performs best on the Credit dataset because of the simple correlations between its features. Using ± 0.2 as the threshold for high correlation, only *Time* and *Amount* are strongly-correlated with other features. All other features have a close-to-0 correlation. Besides, *Time* and *Amount* are only correlated with 3 and 5 features, respectively. With such a small number of correlated features, capturing their relation in the convolution process is easy once we group them together.

The algorithm also alleviates the CNN boundary effect on the highly-correlated features of the Credit dataset. In the original order, *Time* and *Amount* are the leftmost and rightmost columns in the table, and many of their correlated features are far apart. However, after feature sorting, these features are in the middle of the table therefore reducing the boundary effect.

In contrast to the Credit dataset, the other four datasets have a larger number of correlated features. For example, in the Adult dataset most features are correlated with at least one other feature, and seven features are correlated with more than

TABLE VI
ABLATION STUDY RESULTS ON SYNTHESIS QUALITY AND PERMUTATION INVARIANCE OF AE-GAN

Dataset	WD between real and synthetic data					Sensitivity to column permutations		
	AE-GAN	w/o MSN	w/o one-hot & MSN	w/o classifier	co-train AE & GAN	AE-GAN	w/o MSN	w/o one-hot & MSN
Loan	1.374	2.309	3.749	1.308	1.880	7.28%	3.29%	4.45%
Adult	6.042	6.319	15.432	6.293	6.508	21.77%	39.20%	40.65%
Credit	0.341	1.650	1.505	0.344	0.534	9.09%	3.19%	2.70%
Covtype	1.408	3.099	5.954	1.428	2.437	10.16%	5.48%	0.79%
Intrusion	4.328	14.915	58.241	4.492	4.836	10.28%	26.84%	2.76%
Avg.	2.699	5.658	16.976	2.773	3.239	11.71%	15.60%	10.27%

TABLE VII
TABLE-GAN BEFORE AND AFTER THE FEATURE SORTING ALGORITHM.

Dataset	Before sorting		After sorting		Change	
	Dif. Corr.	WD	Dif. Corr.	WD	Dif. Corr.	WD
Loan	2.284	2.062	2.203	2.087	-4%	1%
Adult	1.563	12.153	1.942	12.502	24%	3%
Credit	3.092	0.420	2.728	0.403	-12%	-4%
Covtype	4.885	1.282	4.915	1.348	1%	5%
Intrusion	6.433	6.486	6.597	5.418	3%	-16%
Avg.	3.651	4.481	3.677	4.352	1%	-3%

TABLE VIII
CTAB-GAN BEFORE AND AFTER THE FEATURE SORTING ALGORITHM.

Dataset	Before sorting		After sorting		Change	
	Dif. Corr.	WD	Dif. Corr.	WD	Dif. Corr.	WD
Loan	1.469	0.356	0.638	0.253	-57%	-29%
Adult	0.448	1.517	0.296	1.205	-34%	-21%
Credit	1.688	0.115	1.660	0.134	-2%	17%
Covtype	1.948	0.539	1.442	0.475	-26%	-12%
Intrusion	3.969	2.668	3.385	1.999	-15%	-25%
Avg.	1.904	1.039	1.484	0.813	-22%	-22%

three features. Due to the limited kernel size, it is challenging for CNNs to capture all the cross-column relations even after putting the highly-correlated features together. Besides, our algorithm is based on pairwise correlation, but putting one pair of highly-correlated features together could possibly separate another pair of highly-correlated features, which explains why sometimes Dif. Corr. and WD become worse after applying the feature sorting algorithm. In this case, domain knowledge is required to effectively group the correlated features and arrange them in a good order.

CTAB-GAN. To understand whether our feature sorting algorithm works when sparsity is involved, we test it on CTAB-GAN, and the results are summarized in Table VIII. Surprisingly, the algorithm can reduce Dif. Corr. and WD by more than 10 % on all datasets except the Credit dataset. On the Loan dataset, the Dif. Corr. and WD are decreased by 57% and 29% after feature sorting, meaning that the algorithm can effectively improve the statistical similarity between synthetic and real data.

Compared with table-GAN, CTAB-GAN has more performance gain after feature sorting. This is due to the sparsity issue caused by the encoding methods of CTAB-GAN, i.e., mode-specific normalization for numerical features and one-hot encoding for categorical features. Since the input data are sparse after encoding, putting the highly-correlated columns together can drastically reduce the distance between correlated columns, and therefore improves CTAB-GAN's ability to capture the relation between highly-correlated columns.

To summarize, our feature sorting algorithm can improve the performance of CNN-based table synthesizers, especially when the input tabular data are sparse. For dense tabular data,

it also works if the relation between correlated features is relatively simple.

VII. CONCLUSION

Motivated by the soaring need of synthetic big data, we discover and analyze the varying performance of AI-based tabular data synthesizers to the input column permutation. The state-of-the-art tabular data synthesizers, especially the ones based on convolution neural networks, are sensitive to the column order of the training input. Through empirical analysis on extensive combinations of column permutations, synthesizers, and datasets, we find the root causes for lacking column permutation invariance are the data representation of tabular data and use of convolution neural networks. To address these limitations, we first propose AE-GAN, a GAN-based synthesizer leveraging the representation capacity of autoencoder. Secondly, we propose a feature sorting algorithm that preserves the correlation across input columns and enhances the-state-of-the-art synthesizers considered. Our evaluation results on five datasets show that AE-GAN makes the excellent trade-off among the sensitivity to the input column permutation, training time and the high synthetic data quality and utility. The proposed feature sorting algorithm, on the other hand, enhances the synthesis quality of exiting CNN-based synthesizers, i.e., the statistical difference and ML utility difference between real and synthetic data, by 22%.

REFERENCES

[1] M. Ryan, *Deep learning with structured data.* Simon and Schuster, 2020.

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

[3] L. Gondara and K. Wang, "Mida: Multiple imputation using denoising autoencoders," in *Pacific-Asia conference on knowledge discovery and data mining*, pp. 260–272, Springer, 2018.

[4] R. Camino, C. Hammerschmidt, and R. State, "Working with deep generative models and tabular data imputation," in *ICML Workshop on the Art of Learning with Missing Values (Artemiss)*, 2020.

[5] J. Engelmann and S. Lessmann, "Conditional wasserstein gan-based oversampling of tabular data for imbalanced learning," *Expert Systems with Applications*, vol. 174, p. 114582, 2021.

[6] M. Quintana and C. Miller, "Towards class-balancing human comfort datasets with gans," in *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, pp. 391–392, 2019.

[7] A. Koivu, M. Sairanen, A. Airola, and T. Pahikkala, "Synthetic minority oversampling of vital statistics data with generative adversarial networks," *Journal of the American Medical Informatics Association*, vol. 27, no. 11, pp. 1667–1674, 2020.

[8] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim, "Data synthesis based on generative adversarial networks," *Proc. VLDB Endow.*, vol. 11, no. 10, p. 1071–1083, 2018.

[9] EU, "General data protection regulation (gdpr)," 2018.

[10] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *CoRR*, vol. abs/2110.01889, 2021.

[11] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, *Modeling Tabular Data Using Conditional GAN*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[12] Z. Zhao, A. Kunar, R. Birke, and L. Y. Chen, "Ctab-gan: Effective table data synthesizing," in *Asian Conference on Machine Learning*, pp. 97–112, PMLR, 2021.

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

[14] M. Tschannen, O. Bachem, and M. Lucic, "Recent advances in autoencoder-based representation learning," *arXiv preprint arXiv:1812.05069*, 2018.

[15] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[17] A. Ng *et al.*, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[18] N. Patki, R. Wedge, and K. Veeramachaneni, "The synthetic data vault," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 399–410, 2016.

[19] Z. Li, Y. Zhao, and J. Fu, "Sync: A copula based framework for generating synthetic data from aggregated sources," in *2020 International Conference on Data Mining Workshops (ICDMW)*, pp. 571–578, IEEE, 2020.

[20] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao, "Privbayes: Private data release via bayesian networks," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 4, pp. 1–41, 2017.

[21] E. Choi, S. Biswal, B. Malin, J. Duke, W. F. Stewart, and J. Sun, "Generating multi-label discrete patient records using generative adversarial networks," in *Machine learning for healthcare conference*, pp. 286–305, PMLR, 2017.

[22] M. K. Baowaly, C.-C. Lin, C.-L. Liu, and K.-T. Chen, "Synthesizing electronic health records using improved generative adversarial networks," *Journal of the American Medical Informatics Association*, vol. 26, no. 3, pp. 228–241, 2019.

[23] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *International conference on machine learning*, pp. 3744–3753, PMLR, 2019.

[24] E. Cohen-Karlik, A. Ben David, and A. Globerson, "Regularizing towards permutation invariance in recurrent models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18364–18374, 2020.

[25] O. S. Kayhan and J. C. v. Gemert, "On translation invariance in cnns: Convolutional layers can exploit absolute spatial location," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14274–14285, 2020.

[26] E. Kauderer-Abrams, "Quantifying translation-invariance in convolutional neural networks," *arXiv preprint arXiv:1801.01450*, 2017.

[27] H. Furukawa, "Deep learning for target classification from sar imagery: Data augmentation and translation invariance," *arXiv preprint arXiv:1708.07920*, 2017.

[28] M. Weiler, F. A. Hamprecht, and M. Storath, "Learning steerable filters for rotation equivariant cnns," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 849–858, 2018.

[29] Y. Zhu, T. Brettin, F. Xia, A. Partin, M. Shukla, H. Yoo, Y. A. Evrard, J. H. Doroshow, and R. L. Stevens, "Converting tabular data into images for deep learning with convolutional neural networks," *Scientific reports*, vol. 11, no. 1, pp. 1–11, 2021.

[30] L. Katzir, G. Elidan, and R. El-Yaniv, "Net-{dnf}: Effective deep modeling of tabular data," in *International Conference on Learning Representations*, 2021.

[31] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, "On the spectral bias of neural networks," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 5301–5310, PMLR, 09–15 Jun 2019.

[32] F. Sultana, A. Sufian, and P. Dutta, "Advancements in image classification using convolutional neural network," in *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pp. 122–129, IEEE, 2018.

[33] J. T. Hancock and T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *Journal of Big Data*, vol. 7, no. 1, pp. 1–41, 2020.

[34] X. Liao, "L1aoxingyu/pytorch-beginner: Pytorch tutorial for beginners," 2020.

[35] Y. Zhao, "Yzhao062/pyod: A comprehensive and scalable python library for outlier detection (anomaly detection)," 2020.

[36] M. Zhou, "Morvanzhou/pytorch-tutorial: Build your neural network easy and fast," 2020.

[37] E. Linder-Noren, "Pytorch-gan/wgan_gp.py," 2019.

[38] J. Yoon, D. Jarrett, and M. Van der Schaar, "Time-series generative adversarial networks," *Advances in neural information processing systems*, vol. 32, 2019.

[39] K. R. Castleman, *Digital image processing*. Prentice Hall Press, 1996.

[40] G. Strang and T. Nguyen, *Wavelets and filter banks*. SIAM, 1996.

[41] D. A. Griffith, "The boundary value problem in spatial statistical analysis.," *Journal of regional science*, vol. 23, no. 3, pp. 377–387, 1983.

[42] D. Griffith and C. Amrhein, "An evaluation of correction techniques for boundary effects in spatial statistical analysis: traditional methods," *Geographical Analysis*, vol. 15, no. 4, p. 352, 1983.