Graph-Conditional Flow Matching for Relational Data Generation

Davide Scassola[1,2], **Sebastiano Saccani**[2], **Luca Bortolussi**[1]

[1]AILAB, University of Trieste, Trieste, Italy
[2]Aindo, AREA Science Park, Trieste, Italy
davide.scassola@phd.units.it, sebastiano@aindo.com, lbortolussi@units.it

Abstract

Data synthesis is gaining momentum as a privacy-enhancing technology. While single-table tabular data generation has seen considerable progress, current methods for multi-table data often lack the flexibility and expressiveness needed to capture complex relational structures. In particular, they struggle with long-range dependencies and complex foreign-key relationships, such as tables with multiple parent tables or multiple types of links between the same pair of tables. We propose a generative model for relational data that generates the content of a relational dataset given the graph formed by the foreign-key relationships. We do this by learning a deep generative model of the content of the whole relational database by flow matching, where the neural network trained to denoise records leverages a graph neural network to obtain information from connected records. Our method is flexible, as it can support relational datasets with complex structures, and expressive, as the generation of each record can be influenced by any other record within the same connected component. We evaluate our method on several benchmark datasets and show that it achieves state-of-the-art performance in terms of synthetic data fidelity.

Code — https://github.com/DavideScassola/graph-conditional-flow-matching

Extended version — https://arxiv.org/abs/2505.15668

1 Introduction

Data has become a fundamental resource in the modern world, playing an essential role in business, research and daily life. However, privacy concerns often restrict its distribution. Since most data is stored in relational tables, synthetic data generation is emerging as a solution for sharing useful insights without exposing sensitive information. This approach can ensure compliance with privacy regulations such as the European Union's General Data Protection Regulation (GDPR).

Tabular data synthesis has been subject to research for several years. Early methods were based on Bayesian networks, factor graphs and autoregressive models. Most recent methods leverage the breakthroughs of deep-learning based generative models, from early latent variables models as VAEs (Kingma, Welling et al. 2013) and GANs (Goodfellow et al. 2014), to transformer-based autoregressive models (Vaswani et al. 2017) and diffusion models (Ho, Jain, and Abbeel 2020). Despite the advancements in single table synthesis, generating multiple tables characterized by foreign-key constraints is a considerably more difficult task. Relational datasets can be represented as large graphs, where nodes correspond to records and edges denote foreign-key relationships. This introduces a dual challenge: (1) modeling potentially complex graph structures such as tables with multiple parents, or multiple types of relationships between two tables and (2) modeling statistical dependencies between records linked directly or indirectly through foreign keys. Relational data generation (Patki, Wedge, and Veeramachaneni 2016; Gueye, Attabi, and Dumas 2022; Solatorio and Dupriez 2023; Pang et al. 2024) is a less mature field, with few existing methods capable of properly handling complex database structures. In Xu et al. (2022), they separately model the graph structure and then propose a method for generating the content of the relational database table-by-table. The generation of each record is conditioned on graph-derived node statistics and aggregated information from connected records. In concurrent work, Hudovernik (2024) follow a similar approach, generating the content of the tables using a latent diffusion model conditioned on node embeddings produced by a separate model.

In this work, we propose a method for generating the content of a relational dataset given the graph describing its structure. Inspired by recent advancements in image generation, we employ flow matching to train a flow-based generative model of the content of the entire relational dataset. In order to enable information propagation across connected records, the architecture of the learned denoiser includes a graph neural network (GNN) (Scarselli et al. 2009; Battaglia et al. 2018). This approach aims at maximizing expressiveness in modeling correlation between different records of the database, as information can be passed arbitrarily within a connected component through a GNN. Moreover, our framework is flexible as the conditioning graph can be complex, and scalable as we can generate large datasets.

Using SyntheRela (Hudovernik, Jurkovič, and Štrumbelj 2024), a recently developed benchmarking library, we prove the effectiveness of our method on several datasets, comparing it with several open-source approaches. Experimental re-

sults show our method achieves state-of-the-art performance in terms of fidelity of the generated data.

2 Background

2.1 Continuous Normalizing Flows

Flow matching (Lipman et al. 2022) is a framework for training continuous normalizing flows (CNFs), a deep generative model that learns to transform random noise into target distributions through ordinary differential equations.

Given data $\boldsymbol{x} \in \mathbb{R}^d$ sampled from an unknown data distribution $q(\boldsymbol{x})$ and a time-dependent vector field $v : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$, also called *velocity*, a continuous normalizing flow $\varphi : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ is a transformation defined by the following ODE:

$$\frac{d}{dt}\varphi_t(\boldsymbol{x}) = v_t(\varphi_t(\boldsymbol{x})) \text{ with initial conditions } \varphi_0(\boldsymbol{x}) = \boldsymbol{x}$$

This transformation can be used to map a given tractable distribution $p_0(\boldsymbol{x})$ (e.g., a Gaussian) into a more complex distribution $p_1(\boldsymbol{x})$. The family of density functions $p_t(\boldsymbol{x})$ for $t \in [0,1]$ is known as the *probability density path*, and v_t is said to generate the probability density path $p_t(\boldsymbol{x})$.

Flow matching provides a framework for training a deep neural network to parameterize a velocity field $v_t(\boldsymbol{x})$ such that the resulting ordinary differential equation transforms samples from a tractable noise distribution p_0 into samples approximating the target data distribution $q \approx p_1$.

2.2 The Flow Matching Objective

Since the vector field v_t of a CNF generating the data is unavailable, we cannot directly match a parameterized velocity field v_t^θ against the ground truth v_t. The main idea of flow matching is instead to define the underlying probability path as a mixture of conditional "per-example" probability paths, that can be defined in a tractable way.

Let us denote by $p_t(\boldsymbol{x} \mid \boldsymbol{x}_1)$ a conditional probability path such that $p_0(\boldsymbol{x} \mid \boldsymbol{x}_1) = p_0(\boldsymbol{x})$ and $p_1(\boldsymbol{x} \mid \boldsymbol{x}_1)$ is a distribution concentrated around $\boldsymbol{x} = \boldsymbol{x}_1$, as $p_1(\boldsymbol{x} \mid \boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{x}_1, \sigma^2 I)$ with σ small. We define the conditional velocity $u_t(\boldsymbol{x} \mid \boldsymbol{x}_1)$ as the velocity generating the conditional probability path $p_1(\boldsymbol{x} \mid \boldsymbol{x}_1)$. It is then possible to prove that the marginal velocity, defined as

$$\begin{aligned} u_t(\boldsymbol{x}) &= \int u_t(\boldsymbol{x} \mid \boldsymbol{x}_1) p_t(\boldsymbol{x}_1 \mid \boldsymbol{x}) d\boldsymbol{x}_1 \\ &= \int u_t(\boldsymbol{x} \mid \boldsymbol{x}_1) \frac{p_t(\boldsymbol{x} \mid \boldsymbol{x}_1) q(\boldsymbol{x}_1)}{p_t(\boldsymbol{x})} d\boldsymbol{x}_1 \end{aligned}$$

generates the marginal probability path

$$p_t(\boldsymbol{x}) = \int p_t(\boldsymbol{x} \mid \boldsymbol{x}_1) q(\boldsymbol{x}_1) d\boldsymbol{x}_1$$

that, following the definition of conditional probability path, at $t = 1$ closely matches the target distribution $q(\boldsymbol{x})$. Moreover, it can be shown that a valid loss for learning the marginal velocity is the following:

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t \sim \mathcal{U}[0,1], \boldsymbol{x}_1 \sim q(\boldsymbol{x}_1), \boldsymbol{x} \sim p_t(\boldsymbol{x} \mid \boldsymbol{x}_1)} \| v_t^\theta(\boldsymbol{x}) - u_t(\boldsymbol{x} \mid \boldsymbol{x}_1) \|^2$$

known as the conditional flow matching objective, where a neural network learns to match the marginal velocity by matching conditional velocities.

2.3 Optimal Transport Flows

A common and effective choice of the conditional probability paths is the optimal transport (OT) displacement map between the two Gaussians $p_0(\boldsymbol{x} \mid \boldsymbol{x}_1) = \mathcal{N}(0, I)$ and $p_1(\boldsymbol{x} \mid \boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x}_1, \sigma_{min} I)$:

$$p_t(\boldsymbol{x} \mid \boldsymbol{x}_1) = \mathcal{N}(t\boldsymbol{x}_1, I(1 - t + t\sigma_{min}))$$

generated by the following conditional velocity

$$u_t(\boldsymbol{x} \mid \boldsymbol{x}_1) = \frac{\boldsymbol{x}_1 - (1 - \sigma_{\min})\boldsymbol{x}}{1 - (1 - \sigma_{\min})t} \tag{1}$$

Alternatively, one can write $\boldsymbol{x}_t \sim p_t(\boldsymbol{x} \mid \boldsymbol{x}_1)$ as $\boldsymbol{x}_t = t\boldsymbol{x}_1 + (1 - t + t\sigma_{min})\boldsymbol{x}_0$ and $u_t(\boldsymbol{x} \mid \boldsymbol{x}_1) = \boldsymbol{x}_1 - \boldsymbol{x}_0(1 - \sigma_{min})$ with $\boldsymbol{x}_0 \sim \mathcal{N}(0, I)$.

2.4 Variational Parametrization

In Eijkelboom et al. (2024), they show an alternative to directly matching the conditional velocity. They propose the following parametrization of the learned marginal velocity

$$\begin{aligned} v_t^\theta(\boldsymbol{x}) &:= \int u_t(\boldsymbol{x} \mid \boldsymbol{x}_1) q_t^\theta(\boldsymbol{x}_1 \mid \boldsymbol{x}) \mathrm{d}\boldsymbol{x}_1 \\ &= \mathbb{E}_{\boldsymbol{x}_1 \sim q_t^\theta(\boldsymbol{x}_1 \mid \boldsymbol{x})} u_t(\boldsymbol{x} \mid \boldsymbol{x}_1) \end{aligned}$$

Then the marginal velocity can be learned by matching a variational distribution $q_t^\theta(\boldsymbol{x}_1 \mid \boldsymbol{x})$ to the ground truth $p_t(\boldsymbol{x}_1 \mid \boldsymbol{x})$ by minimizing $D_{\mathrm{KL}}(p_t(\boldsymbol{x}_1 \mid \boldsymbol{x}) \| q_t^\theta(\boldsymbol{x}_1 \mid \boldsymbol{x}))$. This is equivalent to maximum likelihood training:

$$\mathcal{L}_{\mathrm{VFM}}(\theta) = -\mathbb{E}_{t \sim \mathcal{U}[0,1], \boldsymbol{x}_1 \sim q(\boldsymbol{x}_1), \boldsymbol{x} \sim p_t(\boldsymbol{x} \mid \boldsymbol{x}_1)} \log q_t^\theta(\boldsymbol{x}_1 \mid \boldsymbol{x})$$

If the conditional flow is linear in $\boldsymbol{x}_1$ (as in the case of the OT map), then matching the expected value of $p_t(\boldsymbol{x}_1 \mid \boldsymbol{x})$ is enough to learn the marginal velocity $u_t(\boldsymbol{x} \mid \boldsymbol{x}_1)$. This implies that the variational approximation can be a fully factorized distribution, without loss of generality. In this work, we refer to the learned neural network as the *denoiser*.

According to Eijkelboom et al. (2024), this parametrization offers advantages when dealing with one-hot-encoded categorical variables. First, the generative paths become more realistic due to the inductive bias, which improves convergence by avoiding misaligned paths. Second, using cross-entropy loss instead of squared error enhances gradient behavior during training, thereby speeding up convergence.

3 Method

3.1 Relational Data Generation

Relational databases are composed of multiple tables, where each table is a collection of records with a common structure. Tables may include one or more columns containing foreign keys, allowing records to refer to records in other tables. Consequently, a relational database can be represented as a graph, where nodes correspond to records and edges are defined by foreign-key relationships. In particular, the resulting graph is heterogeneous, since the nodes belong to different tables and have different types of features (i.e., the fields of a record). By convention, if a table A contains foreign keys referring to records in table B, A is called the *child* table and B the *parent* table.

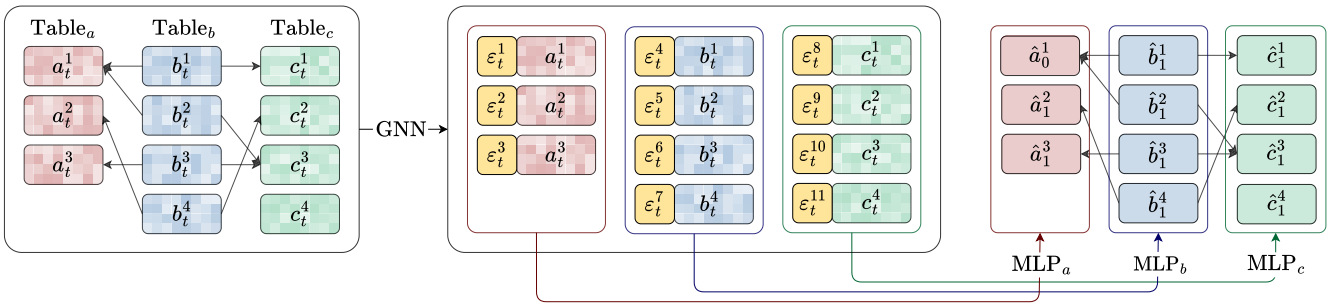

Figure 1: Overview of the architecture of the denoiser for relational data. A relational dataset composed of multiple tables can be seen as a graph, where records are the nodes and foreign keys are the edges. The denoiser takes as input a relational dataset where noise was added to each record with noise level t. Firstly, a graph neural network (GNN) processes the entire graph and computes node embeddings ε_t^i encoding context information for each record. Each record and its corresponding embedding are then processed independently by table-specific multi-layer perceptrons (MLPs), which predict the original clean records ($t = 1$).

Let $\boldsymbol{x}^i$ denote the features of node i, and by g^i the set of nodes to which i is connected to. Then a relational dataset $\mathbb{D}$ is identified by the pair $(\mathbb{X}, \mathbb{G})$, where $\mathbb{X} := \{\boldsymbol{x}^i\}_{i=1}^N$ and $\mathbb{G} := \{g^i\}_{i=1}^N$. We often refer to $\mathbb{X}$ as the *content* of the relational database, and we refer to $\mathbb{G}$ as the *foreign-key graph* or *topology* of the relational database. Moreover, as nodes are grouped into K tables $\mathbb{T}_k$ sharing the same features structure, we can also write $\mathbb{X} = \{\mathbb{T}_k\}_{k=1}^K$.

Several approaches have been explored for relational data generation that differ in the order in which tables and topology are generated. An approach that has been recently proven effective (Xu et al. 2022) is to first generate the topology and then generate the tables one by one, conditioning on the topology and on previously generated tables:

$$p(\mathbb{X}, \mathbb{G}) = p(\mathbb{G}) \prod_{k=1}^{K} p(\mathbb{T}_k \mid \mathbb{T}_{1:k-1}, \mathbb{G})$$

Our approach is similar in the sense that we first generate the topology, but we generate the features contained in the tables all at once:

$$p(\mathbb{X}, \mathbb{G}) = p(\mathbb{G}) p(\mathbb{X} \mid \mathbb{G})$$

In this work we focus on the problem of conditional generation of the features $\mathbb{X}$ given the topology $\mathbb{G}$ of the relational database $p(\mathbb{X} \mid \mathbb{G})$. In this way, the method for generating the foreign-key graph is independent from the method generating the features. This modular approach has advantages, as methods for generating large graphs are often quite various and different from deep generative models. Moreover, the complexity of our conditional generation method scales linearly with the size of the graph, while most methods for graph generation scale quadratically (Zhu et al. 2022). Finally, a conditional generation method is useful when one is interested in anonymizing the content but not the structure of a relational dataset, or when the topology is directly given by the user.

3.2 Foreign-key Graph Generation

Since our work focuses on conditional generation of relational data content given a fixed topology, we adopt a simplified sampling approach for the foreign-key graph $\mathbb{G}$ (treating topology generation as orthogonal to our core contribution). For datasets where $\mathbb{G}$ has a large connected component, containing most or all nodes, we just keep the original topology $\mathbb{G}$. Otherwise, we build a new topology by sampling with replacement the connected components of $\mathbb{G}$ (Hudovernik 2024). This method could be replaced by a dedicated graph sampler as in Xu et al. (2022), which we consider an interesting direction for future work. We remark that the subject of this work is the conditional generation of the content given the foreign-key graph. An advantage of this approach is its modularity, as this method can be combined with any pure graph generation approach. Moreover, resampling the connected components has concrete applications, for instance, when all the observed topologies are well-represented in the training data.

3.3 Generative Modeling from Single-Sample Data

We propose to learn a conditional generative model $p(\mathbb{X} \mid \mathbb{G})$ for the whole set of features $\mathbb{X}$, since in principle, it cannot always be decomposed into several independently and identically distributed (i.i.d.) samples, that in this case would correspond to the connected components of the graph. For example, in the movielens dataset (Harper and Konstan 2015a), almost all records belong to the same connected component. Single-sample scenarios are common in large graph generation problems (e.g., social networks). Time series are another example where identifying i.i.d. samples is problematic. Nevertheless, successful modeling when disposing of only one sample remains feasible when the single sample is composed of weakly interacting components. This is the case of relational data, where records are usually not strongly dependent on all other records belonging to the same connected component. Moreover, the graph is sparse since the number of foreign keys is proportional to the number of records. Our method exploits structural regularities to enable effective learning from what is essentially a single sample $\mathbb{D} = (\mathbb{X}, \mathbb{G})$.

In order to avoid the trivial solution where the model simply learns to reproduce the only available training sample $\mathbb{X}$, we have to carefully handle overfitting. This ensures the generative model generalizes and learns meaningful regularities in the data rather than merely learning to copy the specific instance.

The main motivation for this approach was to develop a maximally expressive generative model for tabular data, addressing the limitations of existing methods. In practice, we achieve this by learning a flow using a modular architecture for the denoiser. This is composed of one denoiser for each table and a GNN. The GNN computes node embeddings for each record, encoding context information. Then, node embeddings are passed to the table-specific denoisers. In this way, the denoising process of each record is made dependent on the other connected records.

Graph-Conditional Flow Matching In order to model $p(\mathbb{X} \mid \mathbb{G})$ using flow matching, we have to define the conditional flow $p_t(\mathbb{X} \mid \mathbb{X}_1, \mathbb{G})$. We use the optimal transport conditional flow described in Section 2 as conditional flow $p_t(\boldsymbol{x}^i \mid \boldsymbol{x}_1^i)$, independent for each node $\boldsymbol{x}^i \in \mathbb{X}$ and for each component of a node (for each field of each record). We refer to the relative conditional velocity as $u_t(\mathbb{X} \mid \mathbb{X}_1)$. This also holds for categorical components of features $\boldsymbol{x}^i$, which are encoded in continuous space using one-hot encodings (Eijkelboom et al. 2024). Notice that this conditional flow does not depend on the topology $\mathbb{G}$, then we can refer to it as $p_t(\mathbb{X} \mid \mathbb{X}_1)$. The objective is to learn the marginal velocity of the whole relational dataset $v_t(\mathbb{X} \mid \mathbb{G})$. We learn this using the variational parametrization discussed above:

$$p_\theta(\mathbb{X}_1 \mid \mathbb{X}_t, \mathbb{G}) \approx q(\mathbb{X}_1 \mid \mathbb{X}_t, \mathbb{G})$$

The training loss is then the following:

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim\mathcal{U}[0,1],\mathbb{X}_t\sim p_t(\mathbb{X}_t\mid\mathbb{X}_1)}\left[-\log p_\theta(\mathbb{X}_1 \mid \mathbb{X}_t, \mathbb{G})\right]$$

Where $(\mathbb{X}_1, \mathbb{G})$ is the original relational dataset. Since the relational dataset $\mathbb{D} = (\mathbb{X}, \mathbb{G})$ is both the dataset and the only sample, using this loss corresponds to doing full-batch training. However, it is also possible to write the loss as an expectation over the different connected components of $\mathbb{D}$ when possible. Finally, the velocity used at generation time is the following:

$$v_t^\theta(\mathbb{X}, \mathbb{G}) = \mathbb{E}_{\mathbb{X}_1\sim p_\theta(\mathbb{X}_1\mid\mathbb{X},\mathbb{G})}\left[u_t(\mathbb{X} \mid \mathbb{X}_1, \mathbb{G})\right] \tag{2}$$

Variational Parametrization Let $x^{k,d,j}$ be the j_{th} value of column d of table k, then we can write $\mathbb{X} = \left\{x^{k,d,j} \;\middle|\; k = 1, \ldots, K;\ d = 1, \ldots, D_k;\ j = 1, \ldots, N_k\right\}$, where D_k and N_k are respectively the number of columns and records in table k. We use a fully factorized distribution as variational approximation. This means that the distribution factorizes into an independent distribution for each component $x^{k,d,j}$. Then we can write the variational approximation as

$$p_\theta(\mathbb{X}_1 \mid \mathbb{X}_t, \mathbb{G}) = \prod_{k=1}^{K}\prod_{d=1}^{D_k}\prod_{j=1}^{N_k} p_\theta^{k,d}(x_1^{k,d,j} \mid \mathbb{X}_t, \mathbb{G})$$

where $p^{k,d}$ represents the variational factor corresponding to column d of table k.

Depending on the nature of variable $x^{k,d,j}$, continuous or categorical, we parametrize a different distribution $p^{k,d}$. In both cases, the distribution is parametrized by a trainable neural network denoiser composed of two modules:

1. A graph neural network η_{θ_1} that computes node embeddings $\varepsilon_t^i = \eta_{\theta_1}(\mathbb{G}, \mathbb{X}_t)^i$ for each node $\boldsymbol{x}^i$, encoding context information.
2. Feedforward neural networks $f_{\theta_2}^{k,d}(x_t^{k,d,j}, t, \varepsilon_t^i)$ using noisy records $x_t^{k,d,j}$, time (noise level) t, and node embeddings ε_t^i to parametrize the distribution $p^{k,d}$.

Categorical Variables. When component $x^{k,d,j}$ is categorical, we use a categorical distribution:

$$p_\theta^{k,d}(x_1^{k,d,j} \mid \mathbb{X}_t, \mathbb{G}) = \text{Categorical}\left(x_1^{k,d,j} \mid \mathbf{p} = f_{\theta_2}^{k,d}(x_t^{k,d,j}, t, \varepsilon_t^i)\right)$$

For categorical variables, the last layer is a softmax function and training corresponds to training a neural network to classify $x_1^{k,d,j}$ using cross-entropy loss.

Continuous Variables. When component $x^{k,d,j}$ is a real number, we use the same architecture to parametrize the mean of a normal distribution with unit variance:

$$p_\theta(x_1^{k,d,j} \mid \mathbb{X}_t, \mathbb{G}) = \mathcal{N}(x_1^{k,d,j} \mid \mu = f_{\theta_2}^{k,d}(x_t^{k,d,j}, t, \varepsilon_t^i), \sigma = 1)$$

In this case training corresponds to training a neural network regressor to predict $x_1^{k,d,j}$ using the squared error loss. We use a fixed unit variance since learning the mean is sufficient to correctly parametrize the velocity field (Eijkelboom et al. 2024).

Velocity Computation. The velocity for each component $x^{k,d,j}$ at time t then follows from Equation 1 and 2:

$$v_t^\theta(x^{k,d,j} \mid \mathbb{X}_t, \mathbb{G}) = \frac{\mathbb{E}_{x_1^{k,d,j}\sim p_\theta^{k,d}(x_1^{k,d,j}\mid\mathbb{X}_t,\mathbb{G})}[x_1^{k,d,j}] - (1-\sigma_{\min})x^{k,d,j}}{1-(1-\sigma_{\min})t}$$

Since we directly parametrize the mean of the distribution in both the categorical and the continuous case, we can just plug the output of the denoiser:

$$v_t^\theta(x^{k,d,j} \mid \mathbb{X}_t, \mathbb{G}) = \frac{f_{\theta_2}^{k,d}(x_t^{k,d,j}, t, \varepsilon_t^i) - (1-\sigma_{\min})x^{k,d,j}}{1-(1-\sigma_{\min})t} \tag{3}$$

Recall that $x^{k,d,j}$ can be either a continuous variable or a one-hot-encoded categorical variable. In this latter case, both $x^{k,d,j}$ and $v_t^\theta(x^{k,d,j} \mid \mathbb{X}_t, \mathbb{G})$ are vectors.

Architecture. Notice that the neural network $f_{\theta_2}^{k,d}$ is specific to the column d of table k. In particular, we use a different multi-layer perceptron for each table k, with a prediction head (the last linear layer) for every component d. The inputs of $f_{\theta_2}^{k,d}$ are concatenated and flattened.

Algorithm 1: Graph-Conditional Generation

Input: $\mathbb{G} \sim p(\mathbb{G})$
Parameter: Number of steps T

Sample $\mathbb{X}_0 \sim \mathcal{N}(0, I)$
for $t = 0$ **to** 1 **step** $\frac{1}{T}$ **do**
    $x_{t+\frac{1}{T}}^{k,d,j} = x_t^{k,d,j} + \frac{1}{T} v_t^{\theta}(x_t^{k,d,j} \mid \mathbb{X}_t, \mathbb{G})$ Eq. (3)
end for
return $(\mathbb{X}_1, \mathbb{G})$

Instead, the neural network η_{θ_1} computing node embeddings $\varepsilon_t^i = \eta_{\theta_1}(\mathbb{G}, \mathbb{X}_t)^i$, refers to a single GNN supporting heterogeneous graph data, i.e., graphs with multiple types of nodes, and as a consequence, different types of edges. We experimented with architectures based on GIN (Xu et al. 2018) and GATv2 (Brody, Alon, and Yahav 2021). In order to build a GNN compatible with heterogeneous graphs, we take an existing GNN model and use a dedicated GNN layer for each edge type, as showed in the documentation of the torch-geometric library (Geometric 2024). The messages produced by each edge-specific layer need to be of the same dimension, so that they can be summed together to obtain a node embedding. Figure 1 shows an overview of the denoiser's architecture.

Computational Complexity Similarly to generative models for single-table data, the complexity of both training and generation of this method scales linearly with the number of records in the relational dataset. First of all, the computational complexity of each layer of the GNN scales linearly with the number of edges. In a relational dataset the total number of foreign keys is proportional to the number of records, since each record (node) has a fixed number of foreign keys (edges). Consequently, the GNN's computational complexity is linear with the number of records. Second, the multi-layer perceptrons are applied independently to each record, meaning their computational complexity also scales linearly and can be efficiently parallelized.

Implementation Details

Data Preprocessing. Following Kotelnikov et al. (2023), during the preprocessing phase we transform continuous features using quantile transformation, so that all marginals of continuous features will be normally distributed. In order to handle missing data in numerical columns, we augment the tables including an auxiliary column containing binary variables indicating if the data is missing, and we fill missing values with the mean. This allows to preserve the information about missing data, and replicate missing data patterns in the generated samples. For categorical columns with missing data, we simply include a new category "NaN". Tables that contain only foreign-key columns (and no features) are considered only when computing node embeddings. Time embedding is implemented according to Dhariwal and Nichol (2021).

Training. In order to avoid overfitting, we randomly split nodes into a training and validation set, computing the loss only on training nodes. The experimental results report performance relative to models achieving the best validation loss during training.

We performed full-batch training, as the entire dataset and the denoiser's computations could be fitted into memory. This allowed us to denoise all record in parallel. For scenarios involving significantly larger datasets, where the graph or the GNN computations exceed available memory, one could employ mini-batching techniques for GNNs, or strategies for scaling to out-of-memory graphs.

We observed relatively short training time: for the largest dataset, training took less than 20 minutes on a single GPU, for the other datasets just a few minutes.

Generation. To solve the ODE generating the data, we used the Euler integration method with 100 steps. In our experiments the generation process was relatively fast: for the largest dataset we experimented with, the generation process took less than 10 seconds. We summarize in Algorithm 1 the graph-conditional generation algorithm.

Hyperparameters. In our experiments, we tuned hyperparameters to some extent depending on the dataset. These primarily include neural network parameters, such as the number of hidden units in feedforward layers. The size of the node embeddings is also a key hyperparameter. Tuning this value was important to balance expressiveness and overfitting, as overly large embeddings can lead the model to memorize structures. Across all experiments, we constrained the embedding size to values between 2 and 10.

4 Experiments

4.1 Experimental Settings

We evaluate our method using SyntheRela (Synthetic Relational Data Generation Benchmark)(Hudovernik, Jurkovič, and Štrumbelj 2024), a recently developed benchmark library for relational database generation. This tool enables comparison of synthetic data fidelity (i.e., similarity to original data) across multiple open-source generation methods and various datasets.

Datasets. We experiment with six real-world relational datasets: AirBnB (Kaggle 2015a), Walmart (Kaggle 2014), Rossmann (Kaggle 2015b), Biodegradability (Blockeel et al. 2004), CORA (McCallum et al. 2000) and the IMDb MovieLens dataset (Harper and Konstan 2015b,a), a commonly used dataset to study graph properties, containing users' ratings to different movies. The last three dataset have tables with multiple parents. Some of these datasets were subsampled to enable comparison with other methods.

Metrics. Our primary objective is generating high-fidelity synthetic relational data. To evaluate fidelity, we adopt a discriminator-based approach where an XGBoost classifier (Chen and Guestrin 2016) (often the preferred classifier for tabular data) is trained to distinguish real from synthetic records. Lower discriminator accuracy indicates higher synthetic data quality, with an accuracy of 0.5 implying indistinguishability. While this is straightforward for single table datasets, where the input of the discriminator is a single row,

this is not for relational data. For relational data, we use the SyntheRela library's Discriminative Detection with Aggregation (DDA) metric, which extends single-table discrimination by enriching the content of the rows of parent tables with aggregate information of its "child" rows. In particular, they add to each row of a parent table the count of children, the mean of real-valued fields of children and the count of unique values of categorical value fields. We believe that discriminator-based metrics are a simple, concise and powerful way to evaluate the fidelity of synthetic data. We compare our method against those present in SyntheRela, that are the leading open-source approaches for relational data generation.

4.2 Results

We generate synthetic data for six of the datasets included in the SyntheRela library, and compare it with other relational data generation methods. In particular, we measure the accuracy of an XGBoost discriminator in the setting described above. Table 1 shows the average accuracy across different runs for each combination of dataset and method when possible. Where the dataset has multiple parent tables, the highest accuracy is reported.

Our method generally outperforms all baselines, often by a large margin. Moreover, it is applicable to all relational datasets considered, as it can handle complex schema structures, including tables with multiple parent tables, multiple foreign keys referencing the same table, and missing data. Missing results for some baselines are due to their limitations: ClavaDDPM cannot synthesize CORA and Biodegradability, as it only supports a single foreign-key relation between two tables, REaLTabF does not support tables with multiple parents and SDV fails to synthesize the IMDB dataset due to scalability issues (Hudovernik, Jurkovič, and Štrumbelj 2024). The performance metrics for other methods are taken from Hudovernik (2024), where the SyntheRela library was also used for fidelity evaluation. Variability in the reported results is due to different initialization seeds used both for training and generation.

To assess the impact of the embeddings produced by the GNN, we evaluate the performance of our method when the embeddings are ablated. This corresponds to training separate single-table models for each table. The results were negatively affected, with the only exception being the CORA dataset. Nevertheless, we observed that ablating the GNN always led to a significant increase in validation loss, thus suggesting potential limitations of the discriminative metric in evaluating certain datasets.

Privacy Evaluation. We evaluated potential privacy leaks in each table, where parent tables were enriched with aggregated information as previously described. For each table, we computed the distance-to-closest-record (DCR) (Park et al. 2018; Steier et al. 2025) of each synthetic and real record comparing to a hold-out set of real records. We consider a synthetic table to exhibit privacy leakage if the percentage of its DCRs falling below the α-percentile of the DCRs of real data is significantly greater than α (Boudewijn et al. 2023; Platzer and Reutterer 2021). Intuitively, this indicates that synthetic records are close to real records more often than expected, suggesting potential privacy risk. As shown in Table 2, when considering the 2% percentile this percentage ($p_{\leq 2\%}$) remains close to the expected value of 2%. The privacy score (Palacios et al. 2025), a derived statistic that takes a value of zero (or slightly lower) when no privacy risk is detected and one when all synthetic records are deemed risky, is consistently close to zero, indicating negligible privacy risk in the content of the synthetic relational data.

5 Related Works

Single Table Generation. Early approaches for tabular data generation include Bayesian networks (Zhang et al. 2017), autoregressive models (Nowok, Raab, and Dibben 2016) and factor graphs (McKenna, Sheldon, and Miklau 2019), that usually required data to be discretized. Early deep latent-variable models as VAEs and GANs were later extended to model heterogeneous tabular data (Xu et al. 2019; Xu and Veeramachaneni 2018). More recently there have been works leveraging transformer-based deep autoregressive model (Castellon et al. 2023) and diffusion models (Kotelnikov et al. 2023; Zhang et al. 2023; Shi et al. 2025). Notably, in Jolicoeur-Martineau, Fatras, and Kachman (2024) they use flow matching where the denoiser is based on trees, which have often been shown to be state of the art for several tasks relating to tabular data (Grinsztajn, Oyallon, and Varoquaux 2022; Shwartz-Ziv and Armon 2022; Borisov et al. 2022).

Relational Data Generation. Synthesizing relational data introduces the additional challenge of preserving inter-table dependencies. The Synthetic Data Vault (SDV) (Patki, Wedge, and Veeramachaneni 2016) pioneered multi-table synthesis via the Hierarchical Modeling Algorithm (HMA), which employs Gaussian copulas and recursive conditional parameter aggregation to propagate child-table statistics into parents. GAN-based relational extensions include Gueye, Attabi, and Dumas (2022), conditioning child-table synthesis on parent and grandparent row embeddings, and Li and Tay (2023), where the generation of single rows is based on GANs, but tables are generated sequentially in an autoregressive way, following the foreign-key topology. Following a similar principle Solatorio and Dupriez (2023) and Gulati and Roysdon (2023) leverage instead a transformer-based autoregressive model. These methods still cannot properly manage the generation of tables with multiple parents, since the number of children per parent's row depends only on one of the parents. In general, these sequential methods can only properly deal with tree-like topologies. Pang et al. (2024) introduce a guided diffusion approach using latent variables to capture long-range dependencies across tables. To handle tables with multiple parents, they generate a version of the child table for each parent and heuristically merge them by selecting similar rows. In Xu et al. (2022) the graph is first generated with a statistical method, and then the content of each table is sequentially generated by conditioning to the content of already generated tables and topological information. A similar approach is proposed in concurrent and

	AirBnB	Biodegradability	CORA	IMDB	Rossmann	Walmart
Ours	$\mathbf{0.58 \pm 0.03}$	$\mathbf{0.59 \pm 0.02}$	0.63 ± 0.02	$\mathbf{0.59 \pm 0.03}$	$\mathbf{0.51 \pm 0.01}$	$\mathbf{0.73 \pm 0.01}$
Ours (no GNN)	0.70 ± 0.005	0.86 ± 0.004	0.62 ± 0.004	0.89 ± 0.002	0.75 ± 0.01	0.91 ± 0.04
Hudovernik (2024)	0.67 ± 0.003	0.83 ± 0.01	$\mathbf{0.60 \pm 0.01}$	0.64 ± 0.01	0.77 ± 0.01	0.79 ± 0.04
ClavaDDPM	≈ 1	-	-	0.83 ± 0.004	0.86 ± 0.01	0.74 ± 0.05
RCTGAN	0.98 ± 0.001	0.88 ± 0.01	0.73 ± 0.01	0.95 ± 0.002	0.88 ± 0.01	0.96 ± 0.02
REaLTabF.	≈ 1	-	-	-	0.92 ± 0.01	≈ 1
SDV	≈ 1	0.98 ± 0.01	≈ 1	-	0.98 ± 0.003	0.90 ± 0.03

Table 1: Average accuracy with standard deviation of an XGBoost multi-table discriminator using rows with aggregated statistics. For datasets with multiple parent tables, the highest accuracy was selected. The CORA dataset is the only one for which using GNN embeddings does not improve the evaluation metric. However, we noticed that the simple post-processing step consisting of removing duplicated records from a child table ($\approx 3\%$ of records), allowed us to obtain a performance of ≈ 0.50. Moreover, we observed a lower validation loss when the GNN was used. Statistics are computed over three different runs.

Dataset	Table	# Records	# Features	$\mathbf{p}_{\leq 2\%}$	Privacy Score
AirBnB	users	10,000	22	$1.95\% \pm 0.08\%$	-0.0005 ± 0.001
Biodegradability	molecule	328	6	$0.00\% \pm 0.00\%$	-0.02 ± 0.000
IMDB MovieLens	movies	3,832	11	$2.44\% \pm 0.04\%$	0.005 ± 0.000
	users	6,039	6	$2.29\% \pm 0.21\%$	0.003 ± 0.002
Rossmann	store	1,115	17	$2.94\% \pm 0.26\%$	0.01 ± 0.003
Walmart	depts	15,047	5	$1.01\% \pm 0.04\%$	-0.01 ± 0.000
	features	225	12	$1.33\% \pm 1.93\%$	-0.007 ± 0.020

Table 2: Privacy results for tables having at least 100 records, at least 2 columns (after aggregation) and no more than 10% of real DCRs equal to zero. Statistics are computed over three different runs.

independent work by Hudovernik (2024), sharing similarities with ours. Data generation is based on latent-diffusion, conditioned on a pre-generated graph by node embeddings encoding topological and neighborhood information, computed using a GNN. Our work differs in three aspects: (1) we employ flow matching rather than latent diffusion; (2) our GNN is integrated into the denoiser, so it is trained end-to-end, whereas theirs uses embeddings precomputed independently from the generative models of the records; (3) we generate tables in parallel rather than sequentially.

6 Limitations

Like other generative models, our approach requires dataset-specific hyperparameter tuning, particularly the depth and width of the employed neural networks, to adapt to the training data size and avoid overfitting. Although the architecture we used is relatively simple, we were able to achieve state-of-the-art performance. Therefore, we believe there is significant room for improvement through more sophisticated model design. This work focuses on the method for training a flexible and powerful generative model for relational data, rather than on the specific architecture of the denoiser.

7 Conclusion

We proposed a novel approach for generating relational data, given the graph describing the foreign-key relationships of the datasets. Our method uses flow matching to build a generative model of the whole content of a relational dataset, exploiting a GNN to increase the expressiveness of the denoiser, by letting information flow across connected records. Our method achieves state-of-the-art performance in terms of synthetic data fidelity across several datasets, outperforming other open-source methods in the SyntheRela benchmark library. Moreover, we did not observe any privacy leakage in the generated synthetic tables, even when parent records were enriched with aggregated statistics from their child tables.

Future Works. An interesting direction of development is the combination with generative models for large graphs, such as exponential random graph models (Robins et al. 2007). We think this approach is promising as the computational complexity of our method scales linearly with the size of the dataset, while deep generative models of graphs often scale quadratically (Zhu et al. 2022). Moreover, foreign-key graphs are often simple enough to be modeled by less powerful but scalable statistical models. Finally, as our method is based on flow matching, one can further exploit properties of diffusion-like models, such as guidance, inpainting, distillation, or the generation of variations of a given dataset.

For a deeper discussion about the experiments, hyperparameters and other technical details we refer readers to the extended version of this article (Scassola, Saccani, and Bortolussi 2025).

Acknowledgements

This research was supported by Aindo, which has funded the PhD of the first author and provided the computational resources. Additionally, this study was carried out within the PNRR research activities of the consortium iNEST funded by the European Union Next-GenerationEU (PNRR, Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043).

References

Battaglia, P.; Hamrick, J. B. C.; Bapst, V.; Sanchez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; Gulcehre, C.; Song, F.; Ballard, A.; Gilmer, J.; Dahl, G. E.; Vaswani, A.; Allen, K.; Nash, C.; Langston, V. J.; Dyer, C.; Heess, N.; Wierstra, D.; Kohli, P.; Botvinick, M.; Vinyals, O.; Li, Y.; and Pascanu, R. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv*.

Blockeel, H.; Džeroski, S.; Kompare, B.; Kramer, S.; Pfahringer, B.; and LAER, W. V. 2004. Experiments in predicting biodegradability. *Applied Artificial Intelligence*, 18(2): 157–181.

Borisov, V.; Leemann, T.; Seßler, K.; Haug, J.; Pawelczyk, M.; and Kasneci, G. 2022. Deep neural networks and tabular data: A survey. *IEEE transactions on neural networks and learning systems*.

Boudewijn, A.; Ferraris, A. F.; Panfilo, D.; Cocca, V.; Zinutti, S.; De Schepper, K.; and Chauvenet, C. R. 2023. Privacy Measurement in Tabular Synthetic Data: State of the Art and Future Research Directions. *arXiv preprint arXiv:2311.17453*.

Brody, S.; Alon, U.; and Yahav, E. 2021. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*.

Castellon, R.; Gopal, A.; Bloniarz, B.; and Rosenberg, D. 2023. DP-TBART: A Transformer-based Autoregressive Model for Differentially Private Tabular Data Generation. *arXiv preprint arXiv:2307.10430*.

Chen, T.; and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Dhariwal, P.; and Nichol, A. 2021. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34: 8780–8794.

Eijkelboom, F.; Bartosh, G.; Andersson Naesseth, C.; Welling, M.; and van de Meent, J.-W. 2024. Variational flow matching for graph generation. *Advances in Neural Information Processing Systems*, 37: 11735–11764.

Geometric, P. 2024. Heterogeneous Graph Learning. PyG Documentation, version 2.6.0.

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

Grinsztajn, L.; Oyallon, E.; and Varoquaux, G. 2022. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35: 507–520.

Gueye, M.; Attabi, Y.; and Dumas, M. 2022. Row Conditional-TGAN for generating synthetic relational databases. *IEEE ICASSP 2023*.

Gulati, M.; and Roysdon, P. 2023. TabMT: Generating tabular data with masked transformers. *Advances in Neural Information Processing Systems*, 36: 46245–46254.

Harper, F. M.; and Konstan, J. A. 2015a. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4): 1–19.

Harper, F. M.; and Konstan, J. A. 2015b. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4): 1–19.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33: 6840–6851.

Hudovernik, V. 2024. Relational Data Generation with Graph Neural Networks and Latent Diffusion Models. In *NeurIPS 2024 Third Table Representation Learning Workshop*.

Hudovernik, V.; Jurkovič, M.; and Štrumbelj, E. 2024. Benchmarking the Fidelity and Utility of Synthetic Relational Data. *arXiv preprint arXiv:2410.03411*.

Jolicoeur-Martineau, A.; Fatras, K.; and Kachman, T. 2024. Generating and imputing tabular data via diffusion and flow-based gradient-boosted trees. In *International Conference on Artificial Intelligence and Statistics*, 1288–1296. PMLR.

Kaggle. 2014. Walmart Recruiting - Store Sales Forecasting. https://www.kaggle.com/competitions/walmart-recruiting-store-sales-forecasting. Accessed: April 29, 2025.

Kaggle. 2015a. Airbnb New User Bookings. https://www.kaggle.com/competitions/airbnb-recruiting-new-user-bookings. Accessed: April 29, 2025.

Kaggle. 2015b. Rossmann Store Sales. https://www.kaggle.com/competitions/rossmann-store-sales. Accessed: April 29, 2025.

Kingma, D. P.; Welling, M.; et al. 2013. Auto-encoding variational bayes.

Kotelnikov, A.; Baranchuk, D.; Rubachev, I.; and Babenko, A. 2023. Tabddpm: Modelling tabular data with diffusion models. In *International Conference on Machine Learning*, 17564–17579. PMLR.

Li, J.; and Tay, Y. 2023. IRG: generating synthetic relational databases using GANs. *arXiv preprint arXiv:2312.15187*.

Lipman, Y.; Chen, R. T.; Ben-Hamu, H.; Nickel, M.; and Le, M. 2022. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*.

McCallum, A. K.; Nigam, K.; Rennie, J.; and Seymore, K. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3: 127–163.

McKenna, R.; Sheldon, D.; and Miklau, G. 2019. Graphical-model based estimation and inference for differential privacy. In *International Conference on Machine Learning*, 4435–4444. PMLR.

Nowok, B.; Raab, G. M.; and Dibben, C. 2016. synthpop: Bespoke creation of synthetic data in R. *Journal of statistical software*, 74: 1–26.

Palacios, M. N. P.; Saccani, S.; Sgroi, G.; Boudewijn, A.; and Bortolussi, L. 2025. Contrastive Learning-Based privacy metrics in Tabular Synthetic Datasets. *arXiv preprint arXiv:2502.13833*.

Pang, W.; Shafieinejad, M.; Liu, L.; Hazlewood, S.; and He, X. 2024. Clavaddpm: Multi-relational data synthesis with cluster-guided diffusion models. *Advances in Neural Information Processing Systems*, 37: 83521–83547.

Park, N.; Mohammadi, M.; Gorde, K.; Jajodia, S.; Park, H.; and Kim, Y. 2018. Data synthesis based on generative adversarial networks. *arXiv preprint arXiv:1806.03384*.

Patki, N.; Wedge, R.; and Veeramachaneni, K. 2016. The synthetic data vault. In *2016 IEEE international conference on data science and advanced analytics (DSAA)*, 399–410. IEEE.

Platzer, M.; and Reutterer, T. 2021. Holdout-based empirical assessment of mixed-type synthetic data. *Frontiers in big Data*, 4: 679939.

Robins, G.; Pattison, P.; Kalish, Y.; and Lusher, D. 2007. An introduction to exponential random graph (p*) models for social networks. *Social networks*, 29(2): 173–191.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1): 61–80.

Scassola, D.; Saccani, S.; and Bortolussi, L. 2025. Graph-Conditional Flow Matching for Relational Data Generation. arXiv:2505.15668.

Shi, J.; Xu, M.; Hua, H.; Zhang, H.; Ermon, S.; and Leskovec, J. 2025. TabDiff: a Mixed-type Diffusion Model for Tabular Data Generation. In *The Thirteenth International Conference on Learning Representations*.

Shwartz-Ziv, R.; and Armon, A. 2022. Tabular data: Deep learning is not all you need. *Information Fusion*, 81: 84–90.

Solatorio, A. V.; and Dupriez, O. 2023. REaLTabFormer: Generating Realistic Relational and Tabular Data using Transformers. arXiv:2302.02041.

Steier, A.; Ramaswamy, L.; Manoel, A.; and Haushalter, A. 2025. Synthetic Data Privacy Metrics. *arXiv preprint arXiv:2501.03941*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Xu, K.; Ganev, G.; Joubert, E.; Davison, R.; Van Acker, O.; and Robinson, L. 2022. Synthetic data generation of many-to-many datasets via random graph generation. In *The Eleventh International Conference on Learning Representations*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Xu, L.; Skoularidou, M.; Cuesta-Infante, A.; and Veeramachaneni, K. 2019. Modeling Tabular data using Conditional GAN. In *Advances in Neural Information Processing Systems*.

Xu, L.; and Veeramachaneni, K. 2018. Synthesizing Tabular Data using Generative Adversarial Networks. *arXiv preprint arXiv:1811.11264*.

Zhang, H.; Zhang, J.; Srinivasan, B.; Shen, Z.; Qin, X.; Faloutsos, C.; Rangwala, H.; and Karypis, G. 2023. Mixed-type tabular data synthesis with score-based diffusion in latent space. *arXiv preprint arXiv:2310.09656*.

Zhang, J.; Cormode, G.; Procopiuc, C. M.; Srivastava, D.; and Xiao, X. 2017. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4): 1–41.

Zhu, Y.; Du, Y.; Wang, Y.; Xu, Y.; Zhang, J.; Liu, Q.; and Wu, S. 2022. A survey on deep graph generation: Methods and applications. In *Learning on Graphs Conference*, 47–1. PMLR.