UNIVERSITÀ DEGLI STUDI DI TRIESTE

UNIVERSITÀ DEGLI STUDI DI TRIESTE

XXXVIII CICLO DEL DOTTORATO DI RICERCA IN APPLIED DATA SCIENCE AND ARTIFICIAL INTELLIGENCE

Finanziato dall'Unione europea - NextGenerationEU
Funded by the European Union - NextGenerationEU

Borsa cofinanziata da Aindo SpA

TOWARDS FLEXIBLE AND EXPRESSIVE GENERATIVE MODELS FOR TABULAR AND RELATIONAL DATA

Settore scientifico-disciplinare: **INF/01**

DOTTORANDO / A
DAVIDE SCASSOLA

COORDINATORE
PROF. FRANCESCO PAULI

SUPERVISORE DI TESI
PROF. LUCA BORTOLUSSI

CO-SUPERVISORE DI TESI
DOTT. SEBASTIANO SACCANI

ANNO ACCADEMICO 2024/2025

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA | UNIVERSITÀ DEGLI STUDI DI TRIESTE

Università degli Studi di Trieste

Ph.D. in Applied Data Science & Artificial Intelligence

XXXVIII cycle

Towards Flexible and Expressive Generative Models for Tabular and Relational Data

Candidate
Davide Scassola

Supervisor
Prof. Luca Bortolussi
Co-supervisor
Sebastiano Saccani

A mio padre

Abstract

Generative modeling has advanced significantly over the past decade, driven by methodological innovation and increased computational resources. While domains such as images, text, and audio have seen widespread adoption of advanced techniques, tabular and relational data present distinct challenges: complex marginal distributions, intricate dependencies, heterogeneous data types, missing values, and hard constraints. These challenges intensify in relational databases, where multiple interconnected tables must be modeled jointly while preserving structural dependencies.

Despite recent progress, crucial limitations remain regarding flexibility. State-of-the-art diffusion models generate high-fidelity synthetic data but lack the ability to incorporate user-specified constraints without retraining, perform general probabilistic queries, or handle complex relational structures without restrictive independence assumptions. This thesis addresses these limitations through three main contributions.

First, we develop a training-free conditional sampling method for score-based models that enables users to impose logical constraints by combining neuro-symbolic constraint encoding with conditional score approximation. Second, we propose an expressive flow-matching framework for generating multi-table relational databases with arbitrary graph structures, where independence between any related records is not assumed, achieving state-of-the-art fidelity. Third, we analyze overparameterized probabilistic circuits as tractable generative models for tabular data, achieving competitive performance while enabling exact likelihood computation, principled handling of missing values, exact conditional sampling on partial evidence, and faster training and sampling compared to diffusion models. We also critically evaluate existing metrics and benchmarks, identifying their limitations and proposing more reliable evaluation protocols. Collectively, this work advances the state of the art in flexible and expressive generative modeling for tabular data.

Acknowledgements

First of all, I would like to thank my supervisors: Luca Bortolussi, Sebastiano Saccani, and Antonio Vergari. Luca guided me in choosing my research subject, always granting me plenty of freedom and offering invaluable advice, which I believe was important in making me a better researcher. I am also grateful to Sebastiano for his help and expertise in my field, for providing funding from Aindo, and for his constant support throughout my PhD without imposing too strict a direction on my work. Finally, during my visit to the School of Informatics in Edinburgh, I could not have asked for a better supervisor than Antonio; together we worked on the material discussed in Chapter 5. His knowledge and expertise are truly inspiring, as his ability to mentor his PhDs and to build a laboratory that produces valuable research while remaining enjoyable and informal.

Moreover, I would like to thank my collaborators, starting with Dylan Ponsford, who gave me continuous support on the work I did during my visit in Edinburgh. Working with you was a real pleasure, and I am sure you will do a great job in your PhD. I also thank Ginevra Carbone and Alex Boudewijn for useful advice, and Alessio Valentinis, who, under my supervision, completed a thesis exploring an interesting direction related to the content of this dissertation. I am also grateful to Federico Cornalba, with whom I collaborated on my first publication; he gave me important advice and inspired me to begin a PhD. Finally, I would like to thank my friend Romina, with whom I co-taught the Probabilistic Machine Learning course together with my supervisor. I really liked working with you.

I will surely remember these three years of my PhD as some of the best of my life. This would not have been possible without the support of many people I was lucky to meet along the way.

I would like to thank all my colleagues and friends of the ADSAI PhD program and of the AIlab. You made this journey truly unforgettable and made every single day of my PhD in Trieste enjoyable and exciting. Without all of you, this would not have been the same. I will avoid mentioning anyone in order to prevent any diplomatic incident.

I am also thankful to the colleagues and friends of *april Lab* (and its visitors) whom I had the pleasure of meeting during my visit to the School of Informatics. You made my visit really enjoyable, and I learned a lot from you.

I would also like to mention my dear friends around the world, most of whom are (or were) PhD students too, for supporting me and for spending time together: Pietro Bonardi, Luigi Bonassi, Francesco Cicala, Salvatore Milite, Gabriele Sarti and Alberto Presta.

I am also grateful to the people I met in Edinburgh, in particular Armando and Mariateresa, who hosted me in their apartment and took care of me as if I were a member of their family. To me, you were like second parents. I also cannot forget to mention my friends Mint and Elisa, along with all their friends, and Karin. I miss all of you and I hope to see you again.

I would like to thank all my long-term friends who have always supported me and who cannot wait to spend time together (in alphabetical order): Daniele, Eugenio, Federico, Leonardo, Lorenzo, Matteo, Michela, Nicola.

Finally, I thank my family for the unconditional support they have always given me. I cannot express how grateful I am for the love of my parents, my brothers, and my aunts. If I am here today, it is also thanks to you, especially my mother who relentlessly takes care of me and my family. In particular, I would like to thank my dad, not being able to share this with you is my only regret.

Contents

Chapter 1

Introduction

Over the past decade, generative modeling has seen a surprising advancement. Once restricted to relatively simple parametric families and low-dimensional settings, generative models today are capable of producing high-fidelity samples across a wide variety of domains. This rapid progress has been driven by two major factors. First, the field has benefited from continuous methodological innovation: from early probabilistic models such as mixtures and hidden Markov models (HMMs) (Rabiner, 1989), to deep latent-variable architectures including variational autoencoders (VAEs) (Kingma and Welling, 2014), generative adversarial networks (GANs) (Goodfellow et al., 2014), autoregressive models (van den Oord et al., 2016a, Vaswani et al., 2017), and most recently normalizing flows (Papamakarios et al., 2019, Rezende and Mohamed, 2015) and diffusion models (Ho et al., 2020, Song et al., 2021). Second, the growth of available computational resources, specialized hardware accelerators, distributed training pipelines, and large-scale datasets have enabled the practical training of models of unprecedented scale and expressiveness. As a result, generative modeling now underpins applications spanning healthcare (Chen et al., 2021), finance (Wiese et al., 2020), robotics (Du et al., 2023), computer vision (Karras et al., 2019), natural language processing (Brown et al., 2020), and scientific discovery (Sanchez-Lengeling and Aspuru-Guzik, 2018).

Despite these advances, one domain remains comparatively understudied: tabular data. Although less popular than images, text, or audio, tabular data constitutes the backbone of real-world information systems. Financial transactions, patient records, census data, e-commerce logs, insurance claims, scientific measurements, and relational databases in general all fundamentally rely on structured tables, often with heterogeneous data types and complex dependencies. Consequently, building high-quality generative models for tabular data has substantial practical importance: synthetic data generation for privacy preservation (Liu et al., 2022b, Patki et al., 2016), data augmentation for learning under limited data (Cui et al., 2024), probabilistic inference for decision-making (Shen et al., 2025, Ziarko, 1999), simulation environments for what-if analyses (Haas et al., 2011), and anomaly detection (Tay et al., 2021), among others.

However, despite usually having limited dimensionality compared to other domains, tabular data presents a unique set of modeling challenges. Unlike domains with strong spatial or sequential inductive biases, tabular datasets typically exhibit:

- Complex marginal distributions, including multimodality, heavy tails, and inflated values.
- Intricate dependencies, often involving nonlinear, high-order, and non-monotonic relationships between features.
- Heterogeneous data types, often mixing continuous, ordinal, categorical, binary, and temporal features within a single dataset.
- Missing data that cannot be trivially imputed.
- Hard constraints, such as physical, logical, or business rules (e.g., "age must be non-negative").
- Difficult quality evaluation: unlike images, text, or audio, where human inspection can immediately identify artifacts, evaluating tabular generative models requires assessing both the realism of individual samples and the fidelity of global statistical properties.
- Privacy concerns: synthetic data must avoid leaking sensitive information about individuals while still retaining realism and utility.

These challenges intensify when moving beyond a single table. Relational data, comprising multiple interconnected tables with foreign-key relationships, is ubiquitous in enterprise, governmental, biomedical, and scientific databases. Generating such data requires learning not only the per-table distributions but also the dependencies between related records and the topological structure of the relationships.

Despite significant progress in improving the expressiveness of generative models for tabular and relational data, several crucial limitations remain, especially regarding flexibility. Many models cannot efficiently incorporate user-specified constraints in the generated data. In practical settings, one often wishes to generate synthetic data consistent with known facts, partial observations, or logical rules. However, most existing methods require retraining or fine-tuning the model for each new constraint (Dhariwal and Nichol, 2021, Naderiparizi et al., 2025, Stoian et al., 2024), which is computationally expensive and impractical in dynamic settings. Moreover, one may be interested in sampling rare events that are not well represented in the training data, posing additional challenges for standard generative approaches. General probabilistic queries are generally not possible with most state-of-the-art models for tabular data generation. Beyond sample generation, many real-world tasks require computing likelihoods, performing marginalization, or answering other inference queries that are valuable for applications such as anomaly detection (Liu et al., 2022a, Nachman and Shih, 2020) and decision-making (Venturato et al., 2022). This also includes performing inference under missing data, which is a common occurrence in tabular datasets. Most existing methods impute missingness as a preprocessing step using heuristics (Shi et al., 2025), rather than treating it in a principled way, thereby losing valuable information. When it comes to relational data generation, current approaches struggle to scale to complex multi-table datasets or require restrictive assumptions about the underlying structure (Pang et al., 2024, Solatorio and Dupriez, 2023, Xu et al., 2022). In this case, expressiveness and flexibility are often

traded off. Many current approaches are either incapable of handling arbitrary schemas with complex inter-table dependencies or rely on assumptions (e.g., independence assumptions) that limit their expressiveness and applicability to real-world databases.

Achieving both expressiveness (the ability to model complex distributions) and flexibility (supporting conditional sampling, inference queries, constraint enforcement, and relational structures) remains an open research challenge. Models that excel in sample fidelity (e.g., diffusion models) often lack tractability for inference, while tractable models are often criticized for limited expressiveness. Understanding and bridging this gap is essential for advancing generative modeling in practical settings.

1.1 Contribution and Outline

This thesis seeks to address some of the aforementioned challenges by exploring new techniques for improving the flexibility of tabular and relational data generation models. In particular, we aim to answer the following research questions:

- RQ1: Can we generate data from a pre-trained score-based generative model while enforcing arbitrary user-defined logical constraints?
- RQ2: Can we train tractable yet expressive generative models for tabular data?
- RQ3: Can we design expressive generative models capable of generating relational data with complex structural dependencies without any implicit independence assumptions?

In this thesis, we aim to answer these questions. Our contribution can be summarized as follows:

- We investigate how score-based generative models can be employed for conditional sampling without retraining, enabling users to impose arbitrary logical constraints or partial evidence.
- We develop a framework for training diffusion-like models capable of handling multi-table relational databases with arbitrary structures, pushing the boundaries of relational generative modeling.
- We study the expressive power of overparameterized tractable generative models, analyzing their ability to capture complex tabular distributions while retaining the ability to compute exact probabilistic queries.
- We critically evaluate existing metrics and benchmarks for tabular data generation, identifying their limitations and proposing more reliable evaluation protocols.

In Chapter 2, we provide the background necessary to understand the generative models we use throughout this thesis, with a focus on generative models for tabular data.

In Chapter 3, we discuss a method for conditional sampling under user-defined logical constraints using pre-trained score-based generative models. Chapter 4 introduces a novel approach for training a flow-matching model to generate relational data with complex structures. Finally, in Chapter 5, we study the effectiveness of overparameterized probabilistic circuits as tractable probabilistic models for tabular data and analyze the limitations of some of the metrics used in previous work.

1.2 Publications

We list below the articles that form the basis of this thesis published by the time of writing:

- Scassola et al. (2025b) Davide Scassola, Sebastiano Saccani, Ginevra Carbone, and Luca Bortolussi. Zero-shot conditioning of score-based diffusion models by neuro-symbolic constraints. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(19):20302–20309, 2025b.[1]
- Scassola et al. (2025a) Davide Scassola, Sebastiano Saccani, and Luca Bortolussi. Graph-conditional flow matching for relational data generation. *arXiv preprint arXiv:2505.15668*, 2025a (Accepted at AAAI26)[2]

Chapters 3 and 4 are based on Scassola et al. (2025b) and Scassola et al. (2025a) respectively. Chatper 5 contains material that will be submitted for publication in the near future. The following other publications during the PhD studies do not contribute to the contents of this thesis: Bortolussi et al. (2023), a short paper discussing a potential way to apply conditional sampling with score-based generative models to surrogate stochastic models, and Cornalba et al. (2024), where we investigate the potential of Multi-Objective, Deep Reinforcement Learning for stock and cryptocurrency single-asset trading.

[1] Code available at: https://github.com/DavideScassola/score-based-constrained-generation

[2] Code available at: https://github.com/DavideScassola/graph-conditional-flow-matching

Chapter 2

Generative Modelling

Summary: This chapter introduces generative modeling fundamentals and key deep generative models such as VAEs, GANs, flows, and diffusion models. Diffusion models and probabilistic circuits are described in more detail, as they form the basis for the methods proposed in this thesis. After comparing the main generative modeling techniques, we focus on generative models for tabular data, discussing the main techniques, challenges, and evaluation metrics.

2.1 Generative Modelling

2.1.1 What are Generative Models?

Machine learning aims to build systems that can learn from data and make decisions with minimal human intervention. This field, often considered a branch of artificial intelligence (AI), lies at the intersection of computer science and statistics, since learning from data requires both complex algorithms and probabilistic reasoning.

For a long time, much of the focus of machine learning has been on discriminative modelling, where the goal is to learn a mapping from inputs to outputs, for instance in classification or regression problems. Formally, given a dataset of input-output pairs $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$ drawn from an unknown joint distribution $p_{data}(\mathbf{x}, y)$, the goal is to learn a conditional distribution $p_\theta(y \mid \mathbf{x})$ that approximates the true conditional distribution $p_{data}(y \mid \mathbf{x})$, where θ are learnable parameters. In practice, in regression and classification problems, a function $f_\theta(\mathbf{x})$ parameterized by θ that predicts y from $\mathbf{x}$ is learned to minimize a given error function $e(y, \hat{y})$, in place of a proper distribution capable of modelling the uncertainty.

However, a more general task is to learn the underlying distribution of the data itself. Given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ drawn i.i.d. from an unknown data distribution $p_{data}(\mathbf{x})$, the objective is to learn a model $p_\theta(\mathbf{x})$ that approximates $p_{data}(\mathbf{x})$. This is often referred to as *generative modelling*, since the learned model can be used to generate new samples $\mathbf{x} \sim p_\theta(\mathbf{x})$ that resemble those in the training dataset. Indeed, usually in

the context of generative modelling, the main objective is to sample from a learned distribution $p_\theta(\mathbf{x})$ that is as close as possible to the underlying distribution $p_{data}(\mathbf{x})$ that generated the training data. Although the focus of generative modelling is often on generation, one can also be interested in answering a variety of other probabilistic queries beyond sampling, such as evaluating likelihoods $p_\theta(\mathbf{x})$, evaluating marginals and conditionals, performing conditional generation $\mathbf{x} \sim p_\theta(\cdot \mid c)$ under available information c, and finding the sample $\mathbf{x}$ that maximizes $p_\theta(\mathbf{x})$.

We can then consider generative modelling as a subset of probabilistic modelling, where uncertainty and stochasticity are central aspects. This aspect is often overlooked in discriminative modelling, even though uncertainty estimation through learning a distribution is possible and crucial in many applications.

Deep generative modelling, i.e., the combination of deep learning with generative modelling techniques, is at the forefront of many recent advances in AI, with applications in various domains (Tomczak, 2024). These include image generation (Goodfellow et al., 2014), text generation (Vaswani et al., 2017), audio and music generation (Huang et al., 2019, van den Oord et al., 2016a), graph generation (e.g., molecular generation) (Jin et al., 2018), tabular data generation for anonymization and data augmentation (Xu et al., 2019), scenario generation for reinforcement learning (Agarwal et al., 2020, Ha and Schmidhuber, 2018), active learning (Sinha et al., 2019), anomaly detection (Zong et al., 2018), and scientific simulation (Hashemi and Krause, 2024, Kim et al., 2019).

In some applications, it is important to generate novel objects, possibly following a user's description, or modify existing ones. However, there are tasks for which generation is not the unique goal. In reinforcement learning, a generative model of the environment can be used to generate the next most likely states, i.e., those with the highest $p(\mathbf{x})$, which guides the actions of the agent. In active learning instead, it is important to find rare, unlikely samples that are hard to label. For medical applications or fraud detection systems, the probability of the label and the object can provide useful information to a specialist, instead of simply assisting with a diagnosis label. Generative modelling intended as probabilistic modelling goes beyond the ability to perform a specific task, but it can provide an AI system a general "understanding" of an environment or phenomenon that can be exploited to perform a variety of tasks. Large language models are a good example of such generalist systems.

2.1.2 An Overview of Generative Modelling Techniques

Early probabilistic models included parametric and mixture models (e.g., Gaussian mixtures), Markov and latent-variable models (e.g., HMMs), and undirected graphical models (LeCun et al., 2006, Rabiner, 1989, Tomczak, 2024). Deep generative models then expanded expressiveness and scalability: variational autoencoders (VAEs) introduced amortized variational inference for latent-variable generative modelling (Kingma and Welling, 2014); generative adversarial networks (GANs) provided an implicit-density framework trained via adversarial objectives (Goodfellow et al., 2014); energy-based models (EBMs) offered flexible unnormalized densities and principled learning rules (Du and Mordatch, 2019, LeCun et al., 2006); normalizing flows provided exact likelihoods via invertible transformations (Papamakarios et al., 2019, Rezende and Mohamed, 2015);

autoregressive models (e.g., PixelCNN and Transformer-based models) factorized $p(\mathbf{x})$ into tractable conditionals (van den Oord et al., 2016b, Vaswani et al., 2017); and, most recently, diffusion, score-based models, and flow-matching models learn to map random noise into samples from $p(\mathbf{x})$ by progressively denoising data to which noise was added, enabling state-of-the-art sample quality in images and flexible conditioning procedures (Song and Ermon, 2019, Song et al., 2021).

The following sections will provide a more in-depth overview of some of these generative modelling techniques, with a focus on diffusion-like models and probabilistic circuits.

2.1.3 Autoregressive Models

Autoregressive models (ARMs) are straightforward generative models that factorize the joint distribution of the data into a product of conditional distributions. Yet, they have shown great success in practice, especially when combined with deep learning architectures. Given data $\mathbf{x}$ composed of D dimensions $(x_1, x_2, \ldots, x_D)$, an autoregressive model factors the joint distribution as:

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{D} p_\theta(x_i \mid \mathbf{x}_{<i}), \tag{2.1}$$

where $\mathbf{x}_{<i} = (x_1, x_2, \ldots, x_{i-1})$ are the preceding dimensions.

In this way, learning can be done by maximum likelihood, which in this case is equivalent to learning D univariate conditional distributions. Sampling from an ARM is also straightforward, and it is done by ancestral sampling, i.e., sampling each dimension sequentially conditioned on the previously sampled dimensions:

$$\tilde{x}_1 \sim p_\theta(x_1), \quad \tilde{x}_2 \sim p_\theta(x_2 \mid \tilde{x}_1), \quad \ldots, \quad \tilde{x}_D \sim p_\theta(x_D \mid \tilde{\mathbf{x}}_{<D}). \tag{2.2}$$

The complexity of sampling is linear in the number of dimensions D.

Autoregressive models can leverage deep learning architectures to parametrize expressive conditional distributions $p_\theta(x_i \mid \mathbf{x}_{<i})$. Unfortunately, this would require training D different models, one for each conditional. This is infeasible in practice for high-dimensional data. To overcome this issue, one can use architectures that share parameters across the different conditionals. Recurrent neural networks (RNNs) are a natural choice for sequential data, as they can maintain a hidden state that summarizes the information from previous dimensions (Mikolov et al., 2010). Convolutional neural networks (CNNs) with causal convolutions can also be used to model dependencies in sequences (van den Oord et al., 2016a, Van den Oord et al., 2016). Another solution is to limit the conditioning context to a fixed-size window of previous dimensions, which can still yield good results. An example of this is the Transformer architecture (Vaswani et al., 2017), which uses self-attention mechanisms to capture dependencies within a limited context. This approach has been particularly successful in large language models (LLMs).

Modelling univariate conditionals is straightforward when the data is discrete, as one can parametrize categorical distributions. Autoregressive models can also be applied to

continuous data by modeling the conditional distributions $p_\theta(x_i \mid \mathbf{x}_{<i})$ using continuous probability distributions, such as a Gaussian or mixture of Gaussians (Theis and Bethge, 2015). Alternatively, one can discretize the continuous data into bins and model the conditionals as categorical distributions over these bins (van den Oord et al., 2016b). Although ARMs allow exact likelihood computation and straightforward sampling, they have limitations in terms of conditioning and marginalization. Specifically, ARMs are not well-suited for arbitrary conditioning or marginalization, as the factorization is fixed and depends on the chosen ordering of dimensions.

2.1.4 Variational Autoencoders

Latent variable models introduce unobserved variables $\mathbf{z}$ to capture the underlying structure of the data:

$$p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z}, \tag{2.3}$$

where the so-called prior distribution $p(\mathbf{z})$ is assumed to be a distribution that is easy to sample from and evaluate, such as a standard normal distribution. Often the latent variables $\mathbf{z}$ are chosen to have lower dimensionality than the data $\mathbf{x}$, thereby providing a compressed representation of the data. Thus, sampling from the model can be done by first sampling $\mathbf{z} \sim p(\mathbf{z})$ and then sampling $\mathbf{x} \sim p(\mathbf{x} \mid \mathbf{z})$, which is called the generator or decoder. Therefore learning the model requires learning the decoder $p_\theta(\mathbf{x} \mid \mathbf{z})$, often modelled as a neural network parametrized by θ. However, learning latent variable models is challenging due to the intractability of the marginal likelihood $p(\mathbf{x})$, which involves integrating over the latent variables $\mathbf{z}$ (although this is proposed in recent works under particular conditions (Gala et al., 2024)).

Variational Autoencoders (VAEs) (Kingma and Welling, 2014) address this challenge by introducing an amortized variational posterior distribution $p_\phi(\mathbf{z} \mid \mathbf{x})$, called the encoder, parametrized by ϕ. In the variational autoencoder framework, the parameters of the encoder and decoder are learned by optimizing a lower bound on the marginal likelihood, known as the evidence lower bound (ELBO):

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{p_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - D_{KL}(p_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})), \tag{2.4}$$

Commonly, both the encoder and decoder are modelled as neural networks parametrizing Gaussian distributions. However, extensions of VAEs have been proposed to improve expressiveness. In particular, it is possible to use a parametric distribution as a prior, and more complex distributions can be used to serve as encoder and decoder (Chen et al., 2016, Gulrajani et al., 2016, Tomczak and Welling, 2017, Van Den Oord et al., 2017).

Despite stable training, VAEs can suffer from posterior collapse, overly smooth samples, and a reconstruction-compression trade-off that limits perceptual fidelity relative to GANs or diffusion models (Bowman et al., 2016, Dai and Wipf, 2019, Tomczak, 2024).

2.1.5 Generative Adversarial Networks

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are a class of latent-variable generative models that learn to generate data by training two neural networks

in an adversarial setting: a generator $G_\theta(\mathbf{z})$ that aims to generate realistic data from latent variables $\mathbf{z}$, and a discriminator $D_\phi(\mathbf{x})$ that aims to distinguish real data from generated data. The adversarial training can be formulated as a minimax game:

$$\min_\theta \max_\phi \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})} \left[\log D_\phi(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})} \left[\log(1 - D_\phi(G_\theta(\mathbf{z})))\right]. \tag{2.5}$$

The generator learns to produce data that fools the discriminator, while the discriminator learns to better distinguish real from fake data. Once trained, the generator can be used to generate new samples by sampling $\mathbf{z} \sim p(\mathbf{z})$ and applying the generator function $G_\theta(\mathbf{z})$.

GANs have been successful in generating high-quality samples, especially in image generation tasks (Brock et al., 2018, Karras et al., 2019). However, GANs can be difficult to train due to issues such as mode collapse, vanishing gradients, and sensitivity to hyperparameters (Arjovsky and Bottou, 2017, Salimans et al., 2016). Moreover, GANs do not provide an explicit density model, making it challenging to evaluate likelihoods or perform inference tasks.

2.1.6 Normalizing Flows

Normalizing Flows (NFs) (Papamakarios et al., 2019, Rezende and Mohamed, 2015) are a class of generative models that learn to transform samples from a simple distribution $\mathbf{z} \sim p_z(\mathbf{z})$ (e.g., a standard normal) into samples from the data distribution $\mathbf{x} \sim p_x(\mathbf{x})$ through an invertible and differentiable transformation $f_\theta : \mathbb{R}^D \to \mathbb{R}^D$ with parameters θ. The distribution $p_x(\mathbf{x})$ defined through the transformation of a random variable $\mathbf{x} = f_\theta(\mathbf{z})$ with $\mathbf{z} \sim p_z(\mathbf{z})$ is given by the change of variables formula:

$$p_x^\theta(\mathbf{x}) = p_z(\mathbf{z} = f_\theta^{-1}(\mathbf{x})) \left|\det \left(\frac{\partial f_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)\right|, \tag{2.6}$$

where $\det \left(\frac{\partial f_\theta^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right)$ is the determinant of the Jacobian of the inverse transformation. Sampling can be done by first sampling $\mathbf{z} \sim p_z(\mathbf{z})$ and then applying the transformation $\mathbf{x} = f_\theta(\mathbf{z})$, while learning the parameters θ can be done by maximizing the exact log-likelihood of the data $p_x^\theta(\mathbf{x})$. Notice that either the transformation f_θ or its inverse f_θ^{-1} must have a tractable Jacobian determinant to allow efficient likelihood evaluation. Since inverting a function is usually more costly than evaluating it, one can either choose to have efficient training or efficient sampling, but not both.

Normalizing flows are built by composing multiple simple invertible transformations to form a more complex transformation. Research focused on finding expressive and invertible building blocks and architectures with tractable Jacobian determinants, such as coupling layers (Dinh et al., 2017), autoregressive flows (Papamakarios et al., 2017), residual flows (Behrmann et al., 2019), planar flows (Rezende and Mohamed, 2015), and invertible DenseNets (Perugachi-Diaz et al., 2021).

Normalizing flows provide exact likelihood evaluation and efficient sampling, and have been used for both density estimation and image generation (Kingma and Dhariwal, 2018). However, scaling to high-dimensional data is still relatively difficult due to the architectural constraints.

2.2 Diffusion Models

Diffusion models are a family of probabilistic generative models that progressively destruct data by injecting noise, then learn to reverse this process for sample generation. Mapping noise into data is not a new idea in generative modeling, as it is the core of earlier latent variable models such as GANs (Goodfellow et al., 2014) and VAEs (Kingma and Welling, 2014). The key difference of diffusion models is that they perform this mapping iteratively through a sequence of increasingly refined intermediate representations, rather than in a single step. This proved to be a very effective strategy to generate high-quality samples, and diffusion models are currently among the state-of-the-art generative models for images and other modalities.

The idea of a probabilistic model based on a diffusion process was introduced by the seminal work of Sohl-Dickstein et al. (2015), who took inspiration from non-equilibrium thermodynamics. These principles were later exploited in Ho et al. (2020), which popularized diffusion models in the deep learning community as state-of-the-art generative models. Almost in parallel, Song and Ermon (2019) proposed score-based generative models, which are closely related to diffusion models but based on learning the score function of the data distribution. These two frameworks were later unified and generalized in Song et al. (2021), which drew connections with stochastic differential equations (SDEs). Continuous flow models (Lipman et al., 2023), although departing in formulation from diffusion models, are a natural evolution of these ideas and currently represent the state-of-the-art in diffusion-like generative models.

In the following sections, we will provide an overview of diffusion models, covering the main formulations and their connections.

2.2.1 Denoising Diffusion Probabilistic Models

Forward Diffusion Process

Given a data point $\mathbf{x}_0 \in \mathcal{X}$ sampled from the data distribution $p(\mathbf{x}_0)$, consider a Markov chain $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ with $t \in 1, \dots, T$ satisfying the following properties:

- It is easy to sample from $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$.
- For T large enough, $p(\mathbf{x}_T)$ is approximately a known distribution from which it is easy to sample and that does not depend on $\mathbf{x}_0$, i.e., $p(\mathbf{x}_T) \approx p(\mathbf{x}_T \mid \mathbf{x}_0)$. This distribution is referred to as the *prior*.

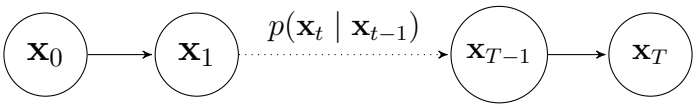


Figure 2.1: Graphical representation of the forward diffusion process.

Such a Markov chain is referred to as the *forward diffusion process*, and $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ is often referred to as the *forward transition kernel*. Figure 2.1 shows a graphical representation of this process. Using the chain rule of probability and the Markov property, we can factorize the joint distribution of $\mathbf{x}_0, \ldots, \mathbf{x}_T$ into:

$$p(\mathbf{x}_0, \ldots, \mathbf{x}_T) = p(\mathbf{x}_0) \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \tag{2.7}$$

In practice, the most used forward process in continuous space is based on the injection of Gaussian noise:

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \coloneqq \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{2.8}$$

where $\beta_t \in (0, 1)$ regulates the amount of noise that is injected at each step (the larger, the faster the convergence to the prior distribution). A useful property of the above process is that we can sample at any arbitrary time step in closed form:

$$p(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, 1 - \bar{\alpha}_t) \tag{2.9}$$

where $\alpha_t \coloneqq 1 - \beta_t$ and $\bar{\alpha}_t \coloneqq \prod_{i=1}^{t} \alpha_i$. This is also useful to show, as $\bar{\alpha}_t \to 0$, that

$$\lim_{t \to \infty} p(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \mathbf{0}, \mathbf{I}) \tag{2.10}$$

So for a sufficiently large T, $\mathbf{x}_T \mathrel{\dot{\sim}} \mathcal{N}(\mathbf{0}, \mathbf{I})$. In other words, this process eventually corrupts the data into white noise.

Backward Process

Now that we are able to gradually corrupt data points $\mathbf{x}_0$ into noise, for generating new data samples we have to revert this process. We have seen that we can easily sample from $p(\mathbf{x}_T)$, but sampling from

$$p(\mathbf{x}_0, \ldots, \mathbf{x}_T) = p(\mathbf{x}_T) \prod_{t=1}^{T} p(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \tag{2.11}$$

is non-trivial, since we do not have access to the reverse transition kernels $p(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$[1] that would allow us to perform ancestral sampling as in the forward process. Therefore, the reverse transition kernels have to be learned. In order to do this, we fit parametric distributions $q_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$, from which it is easy to sample and that can be easily computed, for example:

$$q_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \coloneqq \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \tag{2.12}$$

where the mean and covariance functions are neural networks. The objective is then to find parameters θ such that the backward diffusion process (Figure 2.2) approximates the real reverse process:

$$q_\theta(\mathbf{x}_0, \ldots, \mathbf{x}_T) \coloneqq p(\mathbf{x}_T) \prod_{t=1}^{T} q_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \approx p(\mathbf{x}_0, \ldots, \mathbf{x}_T) \tag{2.13}$$

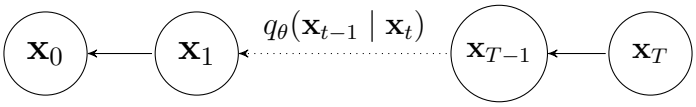


Figure 2.2: Graphical representation of the reverse diffusion process.

The parametric kernels are learned by minimizing the Kullback-Leibler divergence between the forward and backward processes:

$$\begin{aligned} D_{KL}(p(\mathbf{x}_0, \ldots, \mathbf{x}_T) || q_\theta(\mathbf{x}_0, \ldots, \mathbf{x}_T)) &= \mathbb{E}_{p(\mathbf{x}_0, \ldots, \mathbf{x}_T)} \left[\log \frac{p(\mathbf{x}_0, \ldots, \mathbf{x}_T)}{q_\theta(\mathbf{x}_0, \ldots, \mathbf{x}_T)} \right] \\ &= -\mathbb{E}_{p(\mathbf{x}_0, \ldots, \mathbf{x}_T)} \left[\log q_\theta(\mathbf{x}_0, \ldots, \mathbf{x}_T) \right] + \text{const} \\ &= -\mathbb{E}_{p(\mathbf{x}_0, \ldots, \mathbf{x}_T)} \left[\sum_{t=1}^{T} \log q_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \right] + \text{const} \end{aligned} \tag{2.14}$$

Minimizing the KL divergence is equivalent to solving a maximum likelihood estimation problem. We can sample full trajectories starting from samples from the dataset and use them to fit parametric functions q_θ by stochastic optimization, as in a supervised learning setting.

It can be shown that, if forward transition kernels corrupt the data slowly enough (T is large), then the reverse transition kernels will be approximately Gaussian (Sohl-Dickstein et al., 2015). This makes the approximation of reverse transition kernels possible and approachable as a prediction problem. Conversely, if forward transition kernels add too much noise, the reverse transition probabilities can be multimodal, hence a parametric approximation would be intractable.

Although parameters of the forward process (such as β_t) are usually considered fixed (hyperparameters), it is also possible to learn them. In that case, we cannot consider the forward process as constant with respect to the parameters.

Denoising Parametrization

There is empirical evidence that a direct parametrization of the backward transition kernels $q_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is not an effective strategy. Ho et al. (2020) proposed an alternative parametrization, referred to as Denoising Diffusion Probabilistic Models (DDPMs), that proved to be very effective in practice. This parametrization is based on the prediction of the noise $\boldsymbol{\epsilon}_t$ that was added to the original data point $\mathbf{x}_0$ to obtain $\mathbf{x}_t$:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}_t \quad \text{with} \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{2.15}$$

The learning process consists of training a neural network to predict $\boldsymbol{\epsilon}_t$ from $\mathbf{x}_t$ and t, using a squared error loss. The training algorithm described in Ho et al. (2020) is summarized in Algorithm 1.

Once a prediction $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ of $\boldsymbol{\epsilon}_t$ (or equivalently $\mathbf{x}_0$) is available, it can be plugged into the analytical form of the backward transition kernel $p(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ to sample $\mathbf{x}_{t-1}$. It

[1] It is easy to prove that the reverse of a Markov chain is also a Markov chain.

is possible to show that for the Gaussian diffusion process introduced above the distribution $p(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ is also a Gaussian and can be computed analytically:

$$p(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_t(\mathbf{x}_t, \mathbf{x}_0), \sigma_t \mathbf{I}) \tag{2.16}$$

where

$$\mu_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 \,, \sigma_t = \frac{(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\beta_t \tag{2.17}$$

By substituting $\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t))$ (from Equation 2.15) into Equation 2.16 we obtain the sampling scheme summarized in Algorithm 2.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **return** $\mathbf{x}_0$

2.2.2 Score-Based Models

Closely related to diffusion models are score-based generative models (Song and Ermon, 2019), which are based on learning the score function of the data distribution corrupted with different levels of noise. Given a data distribution $p(\mathbf{x})$, its score function is defined as the gradient of the log-density:

$$\mathbf{s}(\mathbf{x}) = \nabla_\mathbf{x} \log p(\mathbf{x}) \tag{2.18}$$

The score function points in the direction of increasing probability density, and it can be used to generate samples from the distribution.

Given the score function, it is possible to generate samples from $p(\mathbf{x})$ using Langevin dynamics (Parisi, 1981, Welling and Teh, 2011):

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{\eta}{2}\nabla_\mathbf{x} \log p(\mathbf{x}_k) + \sqrt{\eta}\boldsymbol{\epsilon}_k \quad \text{with} \quad \boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{2.19}$$

where η is the step size. This iterative procedure refines an initial random sample $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ into a sample from $p(\mathbf{x})$. When the step size η is small enough, the distribution of $\mathbf{x}_k$ converges to $p(\mathbf{x})$ as $k \to \infty$.

There are several methods for estimating the score from data, often referred to as *score matching* (Hyvärinen and Dayan, 2005): *sliced score matching* (Song et al., 2020), *denoising score matching* (Vincent, 2011) and *finite-difference score matching* (Pang et al., 2020). Denoising score matching is the most popular technique, it uses corrupted data samples $\mathbf{x}_t$ in order to estimate the score of the distribution for different levels of added Gaussian noise.

Given a neural network $\mathbf{s}_\theta(\mathbf{x}, t)$ and a diffusion process $q_t(\mathbf{x}_t \mid \mathbf{x}_0)$ defined for $t \in [0, 1]$ such that $q_0(\mathbf{x}_0 \mid \mathbf{x}_0) \approx \delta(\mathbf{x}_0)$ (no corruption) and $q_1(\mathbf{x}_1 \mid \mathbf{x}_0)$ is a fixed prior distribution (e.g., a Gaussian), the denoising score matching loss is:

$$\mathcal{L}_{DSM}(\theta) := \mathbb{E}_{t\sim U(0,1),\mathbf{x}_0\sim p(\mathbf{x}_0),\mathbf{x}_t\sim q_t(\mathbf{x}_t|\mathbf{x}_0)}\lambda(t)\,\|\mathbf{s}_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \ln q_t(\mathbf{x}_t \mid \mathbf{x}_0)\|_2^2 \tag{2.20}$$

where $U(0, 1)$ is a uniform distribution, and $\lambda(t)$ is a weighting function that can be chosen to improve training stability. Often this is set to $\lambda(t) = 1$ or $\lambda(t) \propto 1/\mathbb{E}\big[\,\|\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0)\|_2^2\,\big]$ (Song et al., 2021).

It can be shown that minimizing this loss is equivalent to matching the score of the corrupted data distribution at different noise levels (Vincent, 2011):

$$\mathbf{s}_\theta(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t) \quad \text{where} \quad p_t(\mathbf{x}_t) = \int p(\mathbf{x}_0) q_t(\mathbf{x}_t \mid \mathbf{x}_0)\,\mathrm{d}\mathbf{x}_0 \tag{2.21}$$

Learning the score for different levels of added noise is necessary in practice for sampling in high-dimensional spaces, since the score estimation is inaccurate in low-density regions of the data distribution (Song and Ermon, 2019). Since Langevin dynamics cannot be used to sample from high-dimensional distributions, in Song and Ermon (2019), the authors proposed *annealed Langevin dynamics*, which consists of performing Langevin dynamics at different noise levels, starting from a high noise level and gradually reducing it.

2.2.3 Sampling with Stochastic Differential Equations

The success of diffusion models and score-based generative models is based on perturbing data with multiple levels of noise. In order to generalize this idea, Song et al. (2021) proposed to use a continuous, infinite number of noise scales. This cannot be achieved with discrete-time Markov chains as in DDPMs, but it is possible by defining a continuous-time diffusion process through a Stochastic Differential Equation (SDE).

Perturbing data with SDEs

Given i.i.d. samples $\{\mathbf{x}_i\}$ with $\mathbf{x}_i \in \mathbb{R}^d$ from a data distribution p, the goal is to define a set of distributions p_t continuously indexed by a continuous time variable $t \in [0, 1]$, such that $p_0 = p$ is the data distribution, and p_1 is a tractable distribution for which it is possible to generate samples efficiently, also called the *prior*. This diffusion process can be defined as the solution $\mathbf{x}_t$ to a Stochastic Differential Equation (SDE):

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\,\mathrm{d}t + g(t)\,\mathrm{d}\mathbf{w}, \tag{2.22}$$

where $\mathbf{w}$ is the standard Wiener process (a.k.a.Brownian motion), $\mathbf{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ is a vector-valued function called the *drift* coefficient of $\mathbf{x}_t$, and $g(\cdot) : \mathbb{R} \to \mathbb{R}$ is a scalar function known as the *diffusion* coefficient of $\mathbf{x}_t$.

It can be shown that if the coefficients are globally Lipschitz in both state and time, the SDE has a unique solution (Øksendal, 2003). Thus, we denote by $p_t(\mathbf{x}_t)$ the probability density of $\mathbf{x}_t$, and use $p_t(\mathbf{x}_t \mid \mathbf{x}_0)$ to denote the transition kernel from $\mathbf{x}_0$ to $\mathbf{x}_t$, where $0 \le t \le 1$.

It is convenient to choose an SDE such that:

- The distribution p_1 is a simple prior distribution that contains no information about $\mathbf{x}_0$, such as a Gaussian distribution with fixed mean and variance.
- The transition kernel $p_t(\mathbf{x}_t \mid \mathbf{x}_0)$ is available in closed form, so that we can easily sample $\mathbf{x}_t$ at any time t given $\mathbf{x}_0$.

The second property is particularly important for model training, since denoising score matching requires sampling from $p_t(\mathbf{x}_t \mid \mathbf{x}_0)$ and computing its score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0)$. As an alternative, one can draw samples from $p_t(\mathbf{x}_t \mid \mathbf{x}_0)$ by simulating the SDE, and use sliced score matching to circumvent the computation of the score $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0)$.

There are different ways of designing SDEs that meet these properties. For example, by choosing affine drift coefficients $\mathbf{f}(\cdot, t)$, the transition kernel $p_t(\mathbf{x}_t \mid \mathbf{x}_0)$ becomes Gaussian and can be computed in closed form (Särkkä and Solin, 2019). In particular, Song et al. (2021) proposed some examples, with affine drift coefficients, derived from continuous generalizations of SMLD (Score Matching with Langevin Dynamics (Song and Ermon, 2019)) and DDPM (Ho et al., 2020): the Variance-Exploding (VE) and Variance-Preserving (VP) SDEs.

The Variance-Exploding (VE) SDE is defined as:

$$\mathrm{d}\mathbf{x} = \sqrt{\frac{\mathrm{d}\left[\sigma^2(t)\right]}{\mathrm{d}t}}\,\mathrm{d}\mathbf{w} \tag{2.23}$$

where $\sigma(t)$ is the noise schedule function, possibly with $\sigma^2(0) \approx 0$ and $\sigma^2(1)$ large. The perturbation kernel is then $p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_0, [\sigma^2(t) - \sigma^2(0)]\mathbf{I})$ and $p_1 = \mathcal{N}(\mathbf{x}_0, [\sigma^2(1) - \sigma^2(0)]\mathbf{I})$. It is called Variance-Exploding because the variance of the process increases with time. This simple process just continuously adds Gaussian noise to the data until the original signal is lost. The process defined in Song and Ermon (2019) is a discretization of this SDE with $\sigma(t) = \sigma_{min}(\sigma_{max}/\sigma_{min})^t$.

The Variance-Preserving (VP) SDE is defined as:

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\ \mathrm{d}t + \sqrt{\beta(t)}\ \mathrm{d}\mathbf{w}. \tag{2.24}$$

so that $p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_0 k(t), [1 - k(t)^2]\mathbf{I})$ where $k(t) = e^{-\frac{1}{2}\int_0^t \beta(s)ds}$. In this case $\beta(t)$ has to be chosen such that $k(0) \approx 1$ and $k(1) \approx 0$. Notice that in this case the variance of the process converges to 1. In order to corrupt the data without adding an arbitrary amount of noise, the original signal is progressively scaled down to zero while noise is added. The process defined in Ho et al. (2020) is a discretization of this SDE with $\beta(t) = \beta_{min} + t(\beta_{max} - \beta_{min})$.

Generating samples by reversing the SDE

Once we have defined a forward diffusion SDE that perturbs data into noise, we can use it for generative modeling. In diffusion models, this is done by sampling from the reverse of the forward Markov chain. Similarly, in the continuous-time case, we can generate samples from the data distribution p_0 by reversing the forward SDE. This means sampling from the marginal distribution at time $t = 1$, $\mathbf{x}_1 \sim p_1$, and simulating the reverse-time SDE from $t = 1$ to $t = 0$. According to an important result by Anderson (1982), the reverse of a diffusion process is also a diffusion process, described by the following reverse-time SDE:

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\,\mathrm{d}t + g(t)\,\mathrm{d}\bar{\mathbf{w}}, \tag{2.25}$$

where $\bar{\mathbf{w}}$ is a standard Wiener process, time flows backwards from 1 to 0, and $\mathrm{d}t$ is an infinitesimal negative time step. Notice that in order to simulate this SDE, we need to know the score function $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ of the marginal distribution at each time t. This is where score matching techniques come into play, as we can train a neural network $\mathbf{s}_\theta(\mathbf{x}_t, t)$ to approximate it. Once a score model is trained, we can simulate the reverse-time SDE using numerical SDE solvers (Kloeden and Platen, 2013) to generate samples from p_0, for example using the Euler-Maruyama method:

$$\mathbf{x}_{i-1} = \mathbf{x}_i + [\mathbf{f}(\mathbf{x}_i, t_i) - g(t_i)^2 \mathbf{s}_\theta(\mathbf{x}_i, t_i)]\Delta t + g(t_i)\sqrt{\Delta t}\boldsymbol{\epsilon}_i \tag{2.26}$$

where $\mathbf{x}_1 \sim p_1(\mathbf{x}_1)$, $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, N is the number of time steps, $\Delta t = -\frac{1}{N}$ is a small negative time step, and $t_i = 1 - i\Delta t$ for $i = N, N-1, \ldots, 1$.

In Song et al. (2021), the authors also propose to add one or more Langevin dynamics steps (that they call correction steps) at each time step of the reverse SDE, which can improve sample quality.

Probability flow ODE

In Song et al. (2021), they also show that, for all diffusion processes, there exists a corresponding ODE whose trajectories share the same marginal probabilities $p_t(\mathbf{x}_t)$ as the SDE. This property can be applied to the reverse SDE to define an alternative sampling method based on solving an ODE:

$$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] \mathrm{d}t \tag{2.27}$$

This has been referred to as the *probability flow ODE*. This is an example of a neural ODE (Chen et al., 2018) as the vector field is parameterized by a neural network (the score model). The ODE formulation has several advantages over the SDE formulation:

- **Latent codes**: Since the sampling process is deterministic, the initial noise $\mathbf{x}_1 \sim p_1(\mathbf{x}_1)$ serves as a latent code that fully determines the generated sample $\mathbf{x}_0$. This is useful for applications such as image editing and inpainting.
- **Likelihood computation**: It is possible to approximate the likelihood of data samples using the instantaneous change of variables formula (Chen et al., 2018).
- **Efficient sampling**: It has been observed that the ODE formulation can achieve similar sample quality as the SDE formulation with considerably fewer function evaluations.
- **Invertibility**: The ODE formulation is invertible, allowing data samples to be mapped back to their latent codes.

2.2.4 Flow Matching

Flow matching (Lipman et al., 2023) is an emerging framework for training continuous normalizing flows (CNFs), a deep generative model that learns to transform random noise into target distributions through ordinary differential equations. The benefits of flow matching include those of the aforementioned probability-flow ODE. Moreover, flow matching provides a more flexible choice of flow trajectories, resulting in more efficient training and sampling.

Continuous Normalizing Flows

Given data $\mathbf{x} \in \mathbb{R}^d$ sampled from an unknown data distribution $q(\mathbf{x})$ 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:

$$\frac{d}{dt}\varphi_t(\mathbf{x}) = v_t(\varphi_t(\mathbf{x})), \quad \text{with initial condition } \varphi_0(\mathbf{x}) = \mathbf{x}. \tag{2.28}$$

This transformation can be used to map a given tractable distribution $p_0(\mathbf{x}_0)$ (e.g., a Gaussian) into a more complex distribution $p_1(\mathbf{x}_1)$[2]. The family of density functions $p_t(\mathbf{x}_t)$ for $t \in [0,1]$ is known as the *probability density path*, and v_t is said to generate the probability density path $p_t(\mathbf{x}_t)$.

The objective is then to parametrize a velocity field $v_t^\theta(\mathbf{x}_t)$ with a neural network with parameters θ such that the distribution p_1 resulting from the ODE closely approximates the target data distribution $q(\mathbf{x})$. Generating samples from the learned distribution consists in sampling $\mathbf{x}_0 \sim p_0(\mathbf{x}_0)$ and simulating the ODE from $t = 0$ to $t = 1$ using numerical ODE solvers (Kloeden and Platen, 2013):

$$\mathbf{x}_1 = \mathbf{x}_0 + \int_0^1 v_t^\theta(\mathbf{x}_t)\,\mathrm{d}t. \tag{2.29}$$

Since the likelihood of a given data point can be computed through the instantaneous change of variables formula (Chen et al., 2018), CNFs can be trained by maximizing the likelihood of data samples:

$$\log p_1(\mathbf{x}_1) = \log p_0(\mathbf{x}_0) - \int_0^1 \operatorname{div}\left(v_t^\theta(\mathbf{x}_t)\right)\mathrm{d}t. \tag{2.30}$$

where $\operatorname{div}(v_t^\theta(\mathbf{x}_t))$ is the divergence of the vector field v_t^θ at point $\mathbf{x}_t$. However, maximum-likelihood training of CNFs does not scale well in practice, as it involves expensive ODE simulations for the forward pass and backward propagation for computing gradients.

Flow matching provides an alternative framework for training a neural network to parametrize a velocity field $v_t(\mathbf{x}_t)$ that does not require likelihood computation or ODE simulation during training.

The Flow Matching Objective

Since direct access to the vector field v_t of a CNF generating the data is unavailable, we cannot directly match a parametrized 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 $p_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1)$ a conditional probability path such that $p_{0|1}(\mathbf{x}_0 \mid \mathbf{x}_1) = p_0(\mathbf{x}_0)$ and $p_{1|1}(\mathbf{x}_1 \mid \mathbf{x}_1)$ is a distribution concentrated around $\mathbf{x}_1$, e.g., $p_{1|1}(\mathbf{x}_1 \mid \mathbf{x}_1) = \mathcal{N}(\mathbf{x}_1; \mathbf{x}_1, \sigma^2\mathbf{I})$ with σ small. We define the conditional velocity $u_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1)$ as the velocity generating the conditional probability path $p_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1)$. It is then possible to prove that the marginal velocity, defined as

$$u_t(\mathbf{x}_t) = \int u_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1)\, p_{1|t}(\mathbf{x}_1 \mid \mathbf{x}_t)\,\mathrm{d}\mathbf{x}_1 = \int u_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1)\,\frac{p_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1)\, q(\mathbf{x}_1)}{p_t(\mathbf{x}_t)}\,\mathrm{d}\mathbf{x}_1, \tag{2.31}$$

[2]Notice that in this case the time convention is the opposite of that used in diffusion models. Although potentially confusing, we follow the notation used in the flow matching literature.

generates the marginal probability path

$$p_t(\mathbf{x}_t) = \int p_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1)\, q(\mathbf{x}_1)\, \mathrm{d}\mathbf{x}_1, \tag{2.32}$$

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

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t\sim U(0,1),\, \mathbf{x}_1\sim q(\mathbf{x}_1),\, \mathbf{x}_t\sim p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)} \left\| v_t^\theta(\mathbf{x}_t) - u_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1) \right\|^2, \tag{2.33}$$

known as the conditional flow matching objective, where a neural network learns to match the marginal velocity by matching conditional velocities. This provides a simple and efficient way to train CNFs without requiring likelihood computation or ODE simulation during training.

Optimal Transport Flows

A common and effective choice of the conditional probability paths is the optimal transport (OT) displacement map between the two Gaussians $p_{0|1}(\mathbf{x}_0 \mid \mathbf{x}_1) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $p_{1|1}(\mathbf{x}_1 \mid \mathbf{x}_1) = \mathcal{N}(\mathbf{x}_1, \sigma_{\min}^2 \mathbf{I})$:

$$p_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1) = \mathcal{N}\big(t\,\mathbf{x}_1,\, \mathbf{I}\,(1 - t + t\,\sigma_{\min})\big), \tag{2.34}$$

generated by the following conditional velocity

$$u_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1) = \frac{\mathbf{x}_1 - (1 - \sigma_{\min})\,\mathbf{x}_t}{1 - (1 - \sigma_{\min})\,t}. \tag{2.35}$$

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

Variational Parametrization

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

$$v_t^\theta(\mathbf{x}_t) := \int u_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1)\, p_{1|t}^\theta(\mathbf{x}_1 \mid \mathbf{x}_t)\, \mathrm{d}\mathbf{x}_1 \;=\; \mathbb{E}_{\mathbf{x}_1\sim p_{1|t}^\theta(\mathbf{x}_1|\mathbf{x}_t)} \big[\, u_{t|1}(\mathbf{x}_t \mid \mathbf{x}_1) \,\big]. \tag{2.36}$$

Then the marginal velocity can be learned by matching a variational distribution $p_{1|t}^\theta(\mathbf{x}_1 \mid \mathbf{x}_t)$ to the ground truth $p_{1|t}(\mathbf{x}_1 \mid \mathbf{x}_t)$ by minimizing $D_{\mathrm{KL}}\big(p_{1|t}(\mathbf{x}_1 \mid \mathbf{x}_t) \,\|\, p_{1|t}^\theta(\mathbf{x}_1 \mid \mathbf{x}_t)\big)$ for all $t \in [0, 1]$. This is equivalent to maximum-likelihood training:

$$\mathcal{L}_{\mathrm{VFM}}(\theta) = -\mathbb{E}_{t\sim U(0,1),\, \mathbf{x}_1\sim q(\mathbf{x}_1),\, \mathbf{x}_t\sim p_{t|1}(\mathbf{x}_t|\mathbf{x}_1)} \log p_{1|t}^\theta(\mathbf{x}_1 \mid \mathbf{x}_t). \tag{2.37}$$

If the conditional flow is linear in $\mathbf{x}_1$ (as in the case of the OT map), then matching the expected value of $p_{1|t}^\theta(\mathbf{x}_1 \mid \mathbf{x}_t)$ is enough to learn the marginal velocity $u_t(\mathbf{x}_t)$. This implies that the variational approximation can be a fully factorized distribution, without loss of generality. In this thesis, 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.

2.2.5 Diffusion for Discrete Data

Diffusion models and score-based generative models were originally developed for continuous data, such as images or audio signals. However, many real-world applications involve discrete data, such as text or categorical variables. Adapting these models to discrete data requires addressing several challenges and limitations of diffusion-like models.

Adaptations can be categorized into two main approaches: continuous relaxations and discrete diffusion processes.

Continuous relaxation. A continuous relaxation approach involves embedding discrete data into a continuous space, applying diffusion or score-based methods in that space, and then mapping the generated samples back to the discrete space. For instance, categorical variables can be encoded using one-hot encodings, and text can be encoded using a pre-trained language model. Although in principle this approach is correct and straightforward, in practice working with intrinsically discrete data in a continuous space may lead to suboptimal performance.

As mentioned earlier, Eijkelboom et al. (2024) exploit the variational parametrization to effectively handle one-hot-encoded categorical data. Let us consider without loss of generality the case of a single categorical variable $x \in \{c_1, \dots, c_K\}$, which can be represented as a one-hot vector $\mathbf{h}(x) = \mathbf{h} = (h^1, \dots, h^K)$ where $h^k = 1$ if $x = c_k$, and $h_k = 0$ otherwise. Since the data is now embedded in a continuous space, flow matching with the OT map can be applied as the conditional flow. Note that now, $\mathbf{h}_t = \mathbf{h}(x_0)_t$ is not a one-hot vector for $t > 0$, as it is corrupted by Gaussian noise.

This formulation does not reduce to simply one-hot-encoding the discrete data, the difference is in how the variational distribution $p^\theta_{1|t}(\mathbf{h}_1 \mid \mathbf{h}_t)$ is chosen and learned. Since there is a one-to-one correspondence between the one-hot encoding $\mathbf{h}_1$ and the categorical variable x, $p^\theta_{1|t}(x_1 \mid \mathbf{h}_t)$ can be learned instead, in particular using a categorical distribution:

$$p^\theta_{1|t}(x_1 \mid \mathbf{h}_t) = \text{Categorical}(x_1 \mid \boldsymbol{\pi} = \mathbf{f}_\theta(\mathbf{h}_t, t)) \tag{2.38}$$

where $\mathbf{f}_\theta(\mathbf{h}_t, t)$ is a neural network that outputs a normalized probability vector over the K categories. Thus, the loss for learning the variational distribution becomes

$$\mathcal{L}_{\text{VFM}}(\theta) = -\mathbb{E}_{t\sim U(0,1),\, x_1\sim q(x_1),\, \mathbf{h}_t\sim p_{t|1}(\mathbf{h}_t|x_1)} \log p^\theta_{1|t}(x_1 \mid \mathbf{h}_t). \tag{2.39}$$

The learned velocity field is then:

$$v_t(\mathbf{h}_t) = \mathbb{E}_{\mathbf{h}_1\sim p^\theta_{1|t}(\mathbf{h}_1|\mathbf{h}_t)}\big[\, u_{t|1}(\mathbf{h}_t \mid \mathbf{h}_1)\,\big] = \mathbb{E}_{x_1\sim p^\theta_{1|t}(x_1|\mathbf{h}_t)}\big[\, u_{t|1}(\mathbf{h}_t \mid \mathbf{h}(x_1))\,\big] \tag{2.40}$$

Then, given the linearity of the conditional flow, one can write:

$$v_t(\mathbf{h}_t) = u_{t|1}(\mathbf{h}_t \mid \mathbb{E}_{x_1\sim p^\theta_{1|t}(x_1|\mathbf{h}_t)}\big[\, \mathbf{h}(x_1)\,\big]) = u_{t|1}(\mathbf{h}_t \mid \mathbf{f}_\theta(\mathbf{h}_t, t)) \tag{2.41}$$

In this case, the improvement reduces to using a categorical distribution to denoise corrupted one-hot encodings, which results in a beneficial inductive bias and improved loss (cross-entropy). This has proven sufficient to achieve state-of-the-art performance.

Diffusion in discrete space. An alternative approach is to define diffusion-like processes that operate directly in discrete spaces (Austin et al., 2021, Sohl-Dickstein et al., 2015). For example, Austin et al. (2021) proposed a diffusion process for categorical data based on a Markov chain that randomly replaces tokens with other tokens from the vocabulary. In this case, the learning and sampling procedures are analogous to those of DDPMs. A discrete diffusion process can also be defined in continuous time using a continuous-time Markov jump process (Lou et al., 2024, Meng et al., 2022, Sun et al., 2022). Since the data space is discrete, the score function is not defined in the usual sense. In this case, a "discrete score" function has been defined through probability ratios between neighboring states (Sun et al., 2022). Therefore, the sampling procedure relies on simulating the reverse-time Markov jump process using the learned jumping probability ratios, and more closely resembles Markov chain Monte Carlo methods than Langevin dynamics. Discrete diffusion has been successfully applied to text generation (Lou et al., 2024, Song et al., 2025), achieving competitive results compared to autoregressive models. The main advantage of diffusion-like models over autoregressive models is their ability to generate several tokens in parallel, leading to faster generation times.

As a practical example, we describe the formulation of the generative model for tabular data introduced in Shi et al. (2025), which is based on Austin et al. (2021) and Shi et al. (2024). Here, the diffusion process defines stochastic jumps between discrete states in discrete space, but in continuous time. Moreover, the state space is augmented with a special [MASK] category that does not correspond to any category in real data.

Let $x \in \{c_1, \dots, c_K, [\text{MASK}]\}$ be a categorical variable with $K+1$ categories, including the [MASK] category that is not present in the original data, and let $\mathbf{h}(x) = \mathbf{h} = (h^1, \dots, h^{K+1})$ denote its one-hot encoding. In particular, we denote by $\mathbf{m} = \mathbf{h}([\text{MASK}]) = (0, \dots, 0, 1)$ the one-hot encoding of the [MASK] token. We consider data composed of D categorical features $\mathbf{x} = (x^1, \dots, x^D)$. The diffusion process $p(\mathbf{x}_t), t \in [0,1]$ is defined through the following transition kernel:

$$p(x^d_{t+\epsilon} \mid x^d_t) = \text{Categorical}\left(x^d_{t+\epsilon} \mid \boldsymbol{\pi} = \alpha_\epsilon \mathbf{h} + (1-\alpha_\epsilon)\mathbf{m}\right), \ \forall \epsilon > 0, t+\epsilon \in [0,1], \tag{2.42}$$

independently for each feature, with $p(x_0) = \delta(x_0)$, $\alpha_\epsilon \in [0,1]$ strictly decreasing, $\alpha_0 \approx 1$ and $\alpha_1 \approx 0$. This means that at each step, the state is replaced by the [MASK] token with probability $1-\alpha_\epsilon$, and remains unchanged with probability α_ϵ. Thus, as time progresses, the original data is progressively masked until at $t=1$ all features are masked $\mathbf{x}_1 = ([\text{MASK}], \dots, [\text{MASK}])$. This formulation implies $\alpha_\epsilon = e^{-\sigma_t}$ for some strictly increasing function $\sigma_t : [0,1] \to \mathbb{R}^+$.

The true posterior distribution conditioned on the clean data $\mathbf{x}_0$ can be written in closed form:

$$p(x^d_{t-\epsilon} \mid x^d_t, x^d_0) = \begin{cases} \delta(x^d_{t-\epsilon} = x^d_t) & x^d_t \neq [\text{MASK}], \\ \text{Categorical}\left(x^d_{t-\epsilon} \mid \boldsymbol{\pi} = \frac{(1-\alpha_{t-\epsilon})\mathbf{m} + (\alpha_{t-\epsilon} - \alpha_t)\mathbf{h}(x^d_0)}{1-\alpha_t}\right) & x^d_t = [\text{MASK}]. \end{cases} \tag{2.43}$$

In other words, in the backward process, if the state is not masked, it remains unchanged. If the state is masked, it can be recovered with probability $(\alpha_{t-\epsilon} - \alpha_t)/(1-\alpha_t)$, or remain masked with probability $(1-\alpha_{t-\epsilon})/(1-\alpha_t)$.

To learn the backward process, a denoising model $p_\theta(x^d_0 \mid \mathbf{x}_t) = \text{Categorical}(x^d_0 \mid \boldsymbol{\pi} = \mathbf{f}^d_\theta(\mathbf{x}_t, t))$ where $\mathbf{f}_\theta$ is a neural network, is trained to predict the original category $\mathbf{x}^d_0$ from the corrupted state $\mathbf{x}^d_t$ by minimizing a reweighted version of the following loss:

$$\mathcal{L}(\theta) = -\mathbb{E}_{t \sim U(0,1),\, \mathbf{x}_0 \sim q(\mathbf{x}_0),\, \mathbf{x}_t \sim p(\mathbf{x}_t \mid \mathbf{x}_0)} \log p_\theta(x^d_0 \mid \mathbf{x}_t). \tag{2.44}$$

Once the model is trained, the backward transition kernel is approximated by plugging the expected value of the predicted category $\mathbf{f}_\theta(\mathbf{x}_t)$ into the true posterior:

$$p(x^d_{t-\epsilon} \mid x^d_t) \approx p(x^d_{t-\epsilon} \mid x^d_t, x^d_0 = \mathbf{f}^d_\theta(\mathbf{x}_t, t)). \tag{2.45}$$

Sampling from the model consists in starting from the fully masked state ([MASK], . . . , [MASK]) and iteratively applying the approximated backward transition kernel until $t = 0$ to obtain a clean sample $\mathbf{x}_0$. Literature suggests that this simple formulation is sufficient to achieve competitive performance in language modeling tasks and tabular data generation.

2.3 Probabilistic Circuits

2.3.1 Mixture Models

A mixture is a probability distribution where the density is defined as a weighted sum of other density functions. Let $\mathbf{x} = \{x_1, \ldots, x_d\}$ be a set of variables, each having either the reals as domain ($\mathbb{R}$) or a set of $k \in \mathbb{N}$ categorical values represented as integers ($\{1, \ldots, k\}$). Formally, a mixture $p(\mathbf{x})$ over variables $\mathbf{x}$ has the following form:

$$p(\mathbf{x}) = \sum_{i=1}^{n} w_i q_i(\mathbf{x}) \tag{2.46}$$

where $\sum_{i=0}^{n} w_i = 1$ are the learned weights and $q_i(\mathbf{x})$ are distributions over variables $\mathbf{x}$, which can also have learnable parameters. A mixture is a way to construct complex distributions by combining simpler distributions. Importantly, a mixture over tractable distributions retains the tractability.

2.3.2 Hierarchical Mixture Models

In practice, mixtures of simple distributions (as Gaussians and categoricals) have limited expressiveness. Building a hierarchical mixture can instead boost expressiveness while preserving the tractability. Intuitively, a hierarchical mixture can be defined recursively as either a mixture of other hierarchical mixtures defined over the same set of variables $\mathbf{x}$, or as a product of hierarchical mixtures defined over disjoint sets of variables. Probabilistic circuits (Choi et al., 2020), or Sum-Product Networks (SPNs) (Peharz et al., 2015, Poon and Domingos, 2011), are a computational framework that allows us to build such hierarchical mixtures.

A *circuit* (Vergari et al., 2021) c is a parametrized directed acyclic computational graph over $\mathbf{x}$ encoding a function $c(\mathbf{x})$ and comprising three kinds of computational units: *input*, *product*, and *sum*. Each sum or product n receives the outputs of other units as inputs, denoted as the set $\mathsf{in}(n)$. Each unit n encodes a function c_n defined as: (i) $f_n(\mathsf{sc}(n), \theta)$ if n is an input unit having parameters θ and defined over variables *scope* $\mathsf{sc}(n) \subseteq \mathbf{x}$, called its *scope*; (ii) $\prod_{j \in \mathsf{in}(n)} c_j(\mathsf{sc}(j))$ if n is a product unit; and (iii) $\sum_{j \in \mathsf{in}(n)} w_{n,j} c_j(\mathsf{sc}(j))$ if n is a sum unit, where each $w_{n,j} \in \mathbb{R}$ is a parameter of n. The scope of a sum or product unit n is the union of the scopes of its inputs, i.e., $\mathsf{sc}(n) = \bigcup_{j \in \mathsf{in}(n)} \mathsf{sc}(j)$.

A *probabilistic circuit* (PC) is a circuit c encoding a non-negative function, i.e., $c(\mathbf{x}) \geq 0$ for any $\mathbf{x}$, thus encoding a (possibly unnormalized) probability distribution $p(\mathbf{x}) \propto c(\mathbf{x})$. We show a graphical representation of a probabilistic circuit in Figure 2.3.

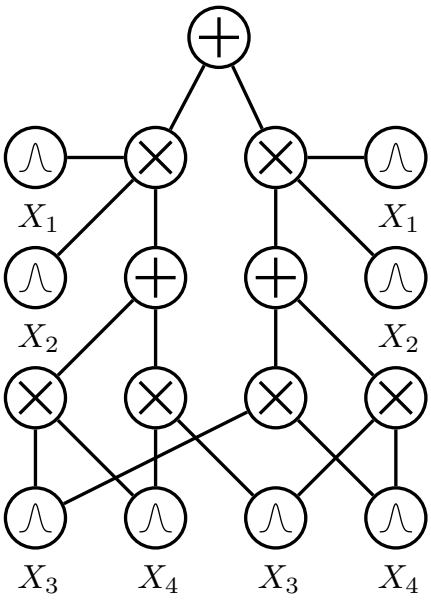


Figure 2.3: Graphical representation of a probabilistic circuit defined over four variables. The evaluation can be performed by first evaluating the input units (the leaves), then evaluating the intermediate units up to the root.

Tractable Queries

Probabilistic circuits, depending on their structure, support several tractable probabilistic queries, as discussed below.

- **Density Evaluation**: Evaluation of the (un-normalized) density function $p(\mathbf{x})$ can be performed in a single forward pass of the computational graph, computing the output of each unit in topological order until the root unit is reached.
- **Marginalization**: A PC c supports the tractable marginalization of any variable subset in a single forward step (Choi et al., 2020) if (i) its input functions f_n can be integrated efficiently and (ii) it is *smooth* and *decomposable* (Darwiche and Marquis, 2002). A circuit is *smooth* if for every sum unit n, all its input units depend on the same variables, i.e, they have the same scope $\forall i, j \in \mathsf{in}(n) \colon \mathsf{sc}(i) = \mathsf{sc}(j)$. A circuit is *decomposable* if the inputs of every product unit n depend on disjoint sets of variables, i.e they have disjoint scopes $\forall i, j \in \mathsf{in}(n)\ i \neq j \colon \mathsf{sc}(i) \cap \mathsf{sc}(j) = \emptyset$.

  This is trivial to prove, since smoothness and decomposability allow summations/integrals to be pushed down/distributed to the input distributions:

$$\int \sum_{i}^{k} w_i p_i(\mathbf{x})\, \mathrm{d}\mathbf{x} = \sum_{i}^{k} w_i \int p_i(\mathbf{x})\, \mathrm{d}\mathbf{x} \tag{2.47}$$

$$\int \dots \int \prod_{i}^{k} p_i(\mathbf{x}_i)\, \mathrm{d}\mathbf{x}_1 \dots \mathrm{d}\mathbf{x}_k = \prod_{i}^{k} \int p_i(\mathbf{x}_i)\, \mathrm{d}\mathbf{x}_i \tag{2.48}$$

  It is easy to show that marginalization can be performed by setting the input units of the marginalized variables to their integral value over their domain (1 for normalized distributions) and then performing a forward pass of the computational graph. Throughout this

dissertation, we will only consider PCs that are smooth and decomposable, taking these two properties for granted. Notice that when input units are normalized density functions and the weights of the sum units are non-negative and sum to 1, then the PC is also a normalized density. In this case there is no need for an additional evaluation of the partition function in another forward step. Indeed, a smooth and decomposable PC encodes a hierarchical mixture model.

- **Sampling**: Leveraging the latent variable interpretation of sum units, sampling from a smooth and decomposable PC requires a single pass of the computational graph:
  - At input units, samples are drawn from the input distributions.
  - At sum units, one of the samples from the input units is randomly selected with a probability proportional to the weights of the sum unit. In other words, a random variable indicating the selected input unit is sampled from a categorical distribution defined by the weights of the sum unit.
  - At product units, the samples from the input units are concatenated.

  Moreover, conditioning on evidence can be performed in a forward pass followed by a "backward" pass. The algorithm for conditional sampling is discussed in Appendix C.1.
- **MAP Inference**: When the circuit is decomposable and *deterministic*, i.e., if the inputs of every sum unit have disjoint supports, maximum a posteriori (MAP) inference can be performed in a single pass of the computational graph.
- **Moments**: A smooth and decomposable PC can compute moment queries tractably as long as the input distribution units support tractable computation of given moment query.
- **Product**: It is possible to define a notion of *compatibility* between two PCs that allows to compute their product as another PC (Loconte et al., 2025b).

Inference with Probabilistic Circuits

The parameters of a PC are the parameters of every input unit and the weights of every sum unit. Since PCs allow for exact computation of the likelihood, training can be done via maximum likelihood estimation. Thanks to the latent variable interpretation of sum units, the expectation-maximization (EM) algorithm represents a valuable alternative (Liu et al., 2025, Zhao et al., 2016). The other characteristic defining a PC is its structure, i.e., the arrangement of its computational units and their connections. This can be either designed manually or learned from data exploiting heuristics (Peharz et al., 2020).

Recent work shows that controlled overparameterization can increase a PC's expressive power while retaining tractable inference (Loconte et al., 2025a). This is done by vectorizing every unit, i.e., making them layers of units that operate over an array of distributions having the same scope instead of a single distribution, until a final aggregation layer combines the distributions into a single output distribution. In particular, each input layer outputs a vector of k different distributions, for example k Gaussians with different means and variances. Sum layers compute k weighted sums of their input distributions, so that the parameters of a sum unit are now a matrix of weights of shape $k \times m$, where m is the number of inputs to the sum unit. Product layers instead do not have parameters, but there are different ways to combine the input distributions of two layers with dimensions a and b. One can either compute all the $a \times b$ products of the input distributions, resulting in a layer of $a \times b$ output distributions (Kronecker product),

or use a more constrained approach as the element-wise product (Hadamard product), which requires $a = b$ and results in a layer of a output distributions. Figure 2.4 shows an example of an overparametrized PC.

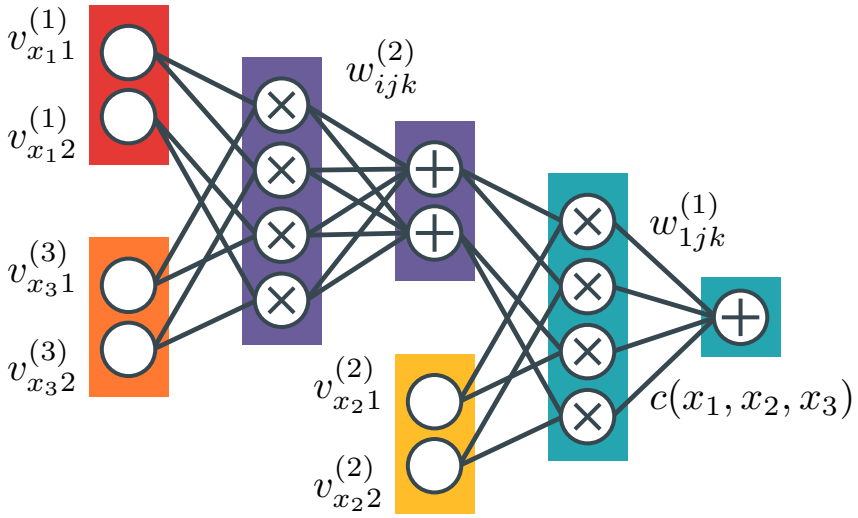


Figure 2.4: Example of overparametrized circuit defined over three variables. In this circuit, each input and sum layer outputs a vector of $k = 2$ distributions, and product layers implement the Kronecker product. Image from Loconte et al. (2025a).

Many classical tractable models such as Gaussian mixture models, hidden Markov models, and trees can be cast as PCs, and PCs generalize several tensor-factorization constructions (Loconte et al., 2025a). Extensions broaden the modeling toolkit: squared circuits permit effective negative mixtures at the cost of losing a simple latent variables interpretation (Loconte et al., 2025b); probabilistic integral circuits introduce continuous latent variables (Gala et al., 2024); and PC layers enable embedding circuits into end-to-end neural architectures for constrained probabilistic prediction (Ahmed et al., 2022).

2.4 Comparison of Generative Models

Generative models can be compared along several axes, including the generation mechanism, sample quality, computational cost, and the availability of likelihoods or other probabilistic quantities. In this section, we provide a unified view of the discussed families of generative models and highlight their conceptual relationships and practical trade-offs.

Connections between generative models. Many popular generative models are based on latent variables. Variational autoencoders (VAEs) and generative adversarial networks (GANs) explicitly introduce a low-dimensional latent space that is mapped to the data space through a neural network. Mixture models can also be viewed as latent variable models, where the latent variable is discrete and indicates the mixture component. In this case, the latent dimensionality may be large, especially for overparameterized probabilistic circuits (PCs).

Normalizing flows (NFs), including continuous normalizing flows (CNFs) and probability ODEs, are technically latent variable models as well, but where transformation is invertible, implying

that the latent space has the same dimensionality as the data space. Autoregressive (AR) models over continuous variables are a special case of normalizing flows with a triangular Jacobian (Papamakarios et al., 2019).

CNFs trained with flow matching have been shown to closely relate to diffusion models. In fact, the training objectives and sampling trajectories of diffusion models can be replicated by CNFs trained with flow matching, indicating that the main differences lie in parameterization and training objectives rather than in the underlying generative process (Lipman et al., 2023).

Autoregressive models can also be interpreted as a form of discrete diffusion, where features are progressively revealed or masked in a fixed order. Both diffusion models and AR models rely on an iterative generation process that refines samples step by step, typically using a single neural network whose behavior is modulated by an additional input such as time in the denoiser of diffusion models (Dhariwal and Nichol, 2021) or positional embedding in transformers (Vaswani et al., 2017). In contrast, GANs, VAEs, NFs, and PCs generate samples in a single forward pass. From a learning perspective, both AR and diffusion models decompose generation into a sequence of supervised learning problems, where it is sufficient to learn a simple distribution (as a Gaussian or categorical).

Performance comparison. In terms of sample quality, diffusion models and autoregressive models generally achieve state-of-the-art results. GANs and VAEs tend to produce lower-quality samples when used in isolation, but many of their techniques have been successfully combined with other generative models, such as adversarial losses applied to diffusion models (Chadebec et al., 2025) or the use of learned latent spaces in latent diffusion approaches (Rombach et al., 2022). Mixture models also typically underperform in sample quality, although highly expressive and overparameterized PCs remain relatively underexplored in this regard. Mixtures have also been combined with other generative paradigms (Grivas et al., 2025). NFs are also not typically competitive with the best-performing models in terms of sample quality, although recent advances have improved their performance (Gu et al., 2025).

Currently, the best-performing model class depends strongly on the data modality. Highly dimensional continuous data, such as images and audio, are most effectively modeled with diffusion-based methods, while discrete data are dominated by autoregressive models. However, recent developments have produced competitive autoregressive models for image generation (Tian et al., 2024) and diffusion-based approaches for text and other discrete domains (Xu et al., 2024).

Generation time varies substantially across model classes. Normalizing flows, VAEs, GANs, and PCs allow for fast sampling through a single neural network evaluation. Sampling with AR models require one evaluation per feature and sequential sampling. Diffusion models are typically slower, requiring tens to hundreds of neural network evaluations. Notably, while autoregressive generation scales with the number of features, the number of diffusion steps is independent of data dimensionality. Recent research has explored reducing the number of diffusion steps to only a few, or even a single step (Chadebec et al., 2025).

Probabilistic queries. The ability to compute probabilistic quantities such as likelihoods also differs significantly between models. Exact likelihoods are available for normalizing flows, probabilistic circuits, and autoregressive models, typically computable in a single neural network evaluation. For VAEs, likelihoods can be approximated via the evidence lower bound (ELBO) (see Equation 2.4), while GANs do not provide likelihoods at all. Diffusion models admit an ELBO-based likelihood approximation, but it is rarely used in practice. For CNFs, likelihoods can in principle be computed via the continuous change-of-variables formula, but this requires solving

an additional ODE and is computationally expensive, with numerical integration introducing further approximations (see Equation 2.30).

Diffusion models, score-based models, and CNFs trained with flow matching provide access to an exact, noise-dependent score function that can be evaluated with a single neural network forward pass. For CNFs trained via flow matching, this score can be recovered explicitly (Eijkelboom et al., 2024). These scores enable likelihood-free inference, sampling, and other downstream probabilistic tasks.

Probabilistic circuits stand out as the only model class that supports exact and efficient marginalization and conditioning, making them particularly attractive for probabilistic querying and reasoning tasks beyond sample generation.

2.5 Tabular Data Generation

2.5.1 Characteristics of Tabular Data

Tabular data consists of structured information organized in rows and columns, where each row represents a data sample and each column corresponds to a specific feature. A key characteristic of tabular data is its heterogeneous column types, meaning that each sample is composed of both categorical and numerical features (integers and real numbers). Unlike high-dimensional data modalities such as images or text, tabular datasets typically exhibit limited dimensionality, commonly containing fewer than a hundred features.

Data has become a fundamental resource in the modern world, playing an essential role in business, research, and daily life. Moreover, most data are stored in relational tables, making tabular data ubiquitous across various domains. Tabular data generation serves two main purposes: privacy preservation and data augmentation, with applications, for instance, in healthcare and finance (Assefa et al., 2021, Fonseca and Bacao, 2023, Giuffrè and Shung, 2023, Gonzales et al., 2023, Hernandez et al., 2022). Synthetic data enables sharing and analysis while protecting sensitive information, which is increasingly relevant given privacy regulations and ethical considerations. This approach can ensure compliance with privacy regulations such as the European Union's General Data Protection Regulation (GDPR). Moreover, augmentation can address the problem of limited training data, particularly in domains where data collection is costly or restricted.

Generating realistic tabular data presents specific challenges. The heterogeneous nature of features requires models to handle different data types simultaneously. Moreover, many tabular datasets exhibit complex marginal distributions that may be multimodal, skewed, or heavy-tailed, and complex relationships between variables, such as non-linear dependencies. Finally, tabular datasets often contain missing values, which complicates the inference process.

2.5.2 Evaluation Metrics

A key difference from other generative modeling tasks is how tabular data generation is evaluated. While image and text generation focuses on sample quality, which can often be assessed by human evaluation, synthetic tabular data must reproduce the statistical properties of the original data. For image data, success is either assessed by visual inspection or by metrics that capture perceptual quality, such as the Fréchet Inception Distance (FID) (Heusel et al., 2017) or the Inception Score (IS) (Salimans et al., 2016). For tabular data, human evaluation is not feasible; success is

measured by the fidelity of the entire dataset's statistical properties rather than only by individual sample quality. Evaluation metrics for synthetic tabular data can be broadly categorized into *fidelity*, *utility*, and *privacy* metrics (Stoian et al., 2025). Most commonly, a metric is a function of the original dataset D and the synthetic dataset S, in other words, it depends on the generative algorithm/model only through the generated data.

Fidelity Metrics

Fidelity metrics assess how similar synthetic data are to the original dataset (either the train set, test set, or both). This is usually done by checking how well the synthetic data replicate the statistical properties of the original dataset, for instance comparing summary statistics (mean, variance, correlations), marginal distributions, and correlations. Some examples for evaluating the similarity of marginals are Wasserstein distance, Maximum Mean Discrepancy (MMD) (Smola et al., 2006), KL divergence, total variation distance, and Jensen-Shannon divergence. In order to examine how well relationships between pairs of features are preserved, some common metrics measure the error (quadratic or absolute) between measures of statistical dependence, such as Pearson correlation, mutual information, or correlation ratio. We provide here a more detailed description of some of the most popular fidelity metrics:

- **Shape**: measures similarity of empirical marginal distributions. For numerical features, the Kolmogorov–Smirnov (KS) test is used:

$$\mathrm{KS}(D, S) = \sup_x |F_D(x) - F_S(x)| \tag{2.49}$$

  where F_D and F_S are the empirical cumulative distribution functions of the original and synthetic datasets, respectively. For categorical features, the total variation distance (TVD) is used:

$$\mathrm{TVD}(D, S) = \frac{1}{2} \sum_{c \in C} |p_D(c) - p_S(c)| \tag{2.50}$$

  where C is the set of categories, and $p_D(c)$ and $p_S(c)$ are the empirical probabilities of category c in the original and synthetic datasets, respectively. To obtain a single metric in $[0, 1]$, the average over all features is computed, and often the complement ($1 -$ KS or $1 -$ TVD) is reported so that higher is better.

- **Trend**: measures similarity of pairwise feature correlations. For pairs of numerical features, the similarity of Pearson correlation coefficients is measured:

$$\frac{1}{2} |\mathrm{corr}_D(X, Y) - \mathrm{corr}_S(X, Y)| \tag{2.51}$$

  referred to as the *Pearson Score* (SDV Developers, DataCebo, 2024). For pairs of categorical features, the total variation distance between the joint distributions is used:

$$\frac{1}{2} \sum_{c_i \in C_i} \sum_{c_j \in C_j} |p_D(c_i, c_j) - p_S(c_i, c_j)| \tag{2.52}$$

  referred to as the *Contingency Score* (SDV Developers, DataCebo, 2024). These metrics are often averaged and complemented to fall in $[0, 1]$. For mixed pairs (numerical and categorical), a practical solution is to discretize the numerical feature.

An important class of fidelity metrics is based on training machine learning models to distinguish between real and synthetic data samples. Lower performance (in terms of accuracy or AUC) suggests higher fidelity, as it implies that the synthetic data are more indistinguishable from the real data. These are called *discriminator-based* metrics or *detection* metrics. In practice, real and synthetic data samples are combined into a single dataset, labeled according to their origin, and a binary classifier is trained to distinguish between them. The performance of the classifier is then evaluated on a held-out test set. In order to obtain a metric in the $[0, 1]$ interval, a transformation is applied to the accuracy or AUC score to obtain a metric referred to as the *classifier two-sample test* (C2ST):

$$\text{C2ST} = 1 - (\max(\text{ROC AUC}, 0.5) \times 2 - 1) \tag{2.53}$$

The most common classifier used in this context is logistic regression, even though there are no established guidelines on the choice of the classifier.

Finally, evaluating the log-likelihood of the real data is another approach to measure fidelity and compare generative models. This is popular in text generation, where the main approach is to evaluate the *perplexity*, that is a metric derived on the likelihood of real data according to the autoregressive generative model. However, most deep generative models for tabular data do not allow tractable and exact likelihood evaluation. Moreover, metrics that only rely on the generated data are often preferred.

A popular package that implements several of the mentioned metrics is *SDMetrics* (SDV Developers, DataCebo, 2024), which is widely used in the literature for evaluating synthetic tabular data.

Utility Metrics

Utility metrics evaluate the usefulness of synthetic data for downstream tasks such as classification or regression. A common approach is to train machine learning models for classification or regression tasks on synthetic data and evaluate their performance on real data, using metrics such as accuracy or mean squared error, and check for performance degradation (Xu et al., 2019). The performance of models trained on synthetic data is often referred to as *Machine Learning Efficiency* (MLE). However, this metric also depends on the particular classification/regression model that is trained, and there is no agreement on which model to use. One possible solution is to use a strong baseline model such as XGBoost (Chen and Guestrin, 2016), as done in a recent series of works (Kotelnikov et al., 2023, Shi et al., 2025, Zhang et al., 2024). Notice that high utility does not imply high fidelity, as synthetic data could be useful for a specific task while not accurately representing the original data distribution. A good example is given by the concept of distillation in deep learning (Hinton et al., 2015, Meng et al., 2023, Salimans and Ho, 2022), where targets generated by a large models are used to train a smaller model in place of the original targets.

Privacy Metrics

Privacy metrics measure the risk of leakage of sensitive information regarding the training data from synthetic data. Notice that if privacy was not a concern, then just copying the original dataset would imply perfect fidelity and utility. A common approach is to design generative models that satisfy *differential privacy* (DP) (Dwork, 2006), which provides formal guarantees on the privacy of individuals in the training data, by adding carefully calibrated randomness so individual records cannot be identified. However, since these often result in poor performance,

an alternative approach is to train a generative model without DP constraints and use empirical privacy metrics to evaluate the risk of information leakage. Indeed, another approach is to evaluate the vulnerability of the model to adversarial attacks, either aimed at predicting whether a specific record was used during training (*membership inference attacks*), inferring sensitive attributes of individuals from some known attributes (*attribute disclosure attacks*), or re-identifying individuals in the training data from synthetic data (*re-identification attacks*) (Hayes et al., 2017, Shokri et al., 2017).

Conceptually derived from this last type of attack, the most popular family of privacy metrics is based on the concept of Distance to Closest Record (DCR) (Jia et al., 2024, Palacios et al., 2025, Zhao et al., 2021). The idea is to measure how close each synthetic record is to the nearest real record of the training data, with respect to some distance metric. The intuition is that if synthetic data points are closer to real data points than expected, then there is a higher risk of information leakage. Using DCR has become a standard practice in the evaluation of synthetic tabular data, but the particular algorithm for evaluating the metric often varies.

As a relevant example, we report the protocol described in Palacios et al. (2025) that we also adopt later. Let $\text{Dist} : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ be a distance metric, let $D = (D_s, D_t)$ be a partitioning of the real dataset into source and target sets, and let S be the synthetic dataset. The DCR for a synthetic record $\mathbf{x}$ with respect to the target set D_t is defined as:

$$\text{DCR}(\mathbf{x}, D_t) = \min_{\mathbf{y} \in D_t} \text{Dist}(\mathbf{x}, \mathbf{y}) \tag{2.54}$$

Where Dist is often chosen as the L1 or L2 distance for numerical features and the Hamming distance for categorical features, after appropriate normalization. The metric then compares the empirical distributions (sets of values) of DCR values of synthetic records: $\text{DCR}_S = \{\text{DCR}(\mathbf{s}, D_t) \mid \mathbf{s} \in S\}$ with those of real source records: $\text{DCR}_D = \{\text{DCR}(\mathbf{d}, D_t) \mid \mathbf{d} \in D_s\}$. Since the interesting values are the smallest ones (records that are very close to target records), usually the left tails of the two distributions are compared (Boudewijn et al., 2023, Platzer and Reutterer, 2021). This is often done by choosing a small fraction p (for instance 2%) and comparing p with the fraction of values in DCR_S that are lower than the p quantile of DCR_D. If this fraction is significantly higher than p, then it means that there are more synthetic records very close to target records than expected, indicating a potential privacy risk. We suggest choosing as target set D_t the whole training set used by the generative model, and as source set D_s a test set (data that the generative model has not seen during training), such that we can check if any of the training records were leaked, and compare the distances with independent records D_s.

2.5.3 Tabular Data Generation Techniques

Tabular data generation has rapidly evolved in the last years, spanning many underlying machine learning techniques. Early approaches for tabular data generation were based on probabilistic models as Bayesian networks (Zhang et al., 2017), factor graphs (McKenna et al., 2019) and copula models (Patki et al., 2016). The first models exploiting deep learning generative models were based on VAEs (Kingma and Welling, 2014) and GANs (Goodfellow et al., 2014), both based on latent variables. The most popular methods in this category are TVAE and CTGAN (Xu et al., 2019), often used as baselines when evaluating newer methods. Methods based on deep autoregressive models were also developed, as Borisov et al. (2023), Solatorio and Dupriez (2023) and Gulati and Roysdon (2023). More recently there have been many works exploiting diffusion models (Ho et al., 2020, Song et al., 2021) and flow-based models (Lipman et al., 2023) as TabDDPM (Kotelnikov et al., 2023), STaSy (Kim et al., 2023), CoDi (Lee et al., 2023), TabSyn (Zhang et al., 2024),

ForestFlow (Jolicoeur-Martineau et al., 2024), TabDiff (Shi et al., 2025), and TabbyFlow (Guzmán-Cordero et al., 2025). The most recent models falling in this category are currently considered the state of the art for tabular data generation.

The adaptation of these machine learning models often involves additional techniques to tackle the challenges of tabular data generation. In order to manage heterogeneous data, CTGAN and TVAE adapt the output layer of the underlying neural network to output a Gumbel-Softmax transformation for categorical variables and a tanh activation for numerical variables. Categorical variables in input are instead embedded with a one-hot encoding. In models based on graphical and autoregressive models, the numerical features are embedded in a discrete space using different techniques. Notably, in LLM-based models numerical values are embedded as language tokens. Diffusion-based models instead mainly work with continuous dimensions. In this case categorical variables are either handled separately with a hybrid continuous-discrete diffusion process (Shi et al., 2025) or they are simply embedded in continuous space using a one-hot encoding (Kotelnikov et al., 2023, Zhang et al., 2024). Models adopting this last solution often adapt the loss for these variables in order to improve performance (Guzmán-Cordero et al., 2025, Kotelnikov et al., 2023).

In the context of models explicitly dealing with continuous distributions, some additional steps are needed to effectively handle complex marginal distributions. In Xu et al. (2019), columns characterized by multimodal distributions are encoded using a special hybrid encoding based on a mixture of Gaussians. In most recent diffusion models for tabular data, each numerical feature is preprocessed using a quantile normalizer, that has the effect of transforming the data such that any marginal distribution is distributed as a normal distribution. This preprocessing step is essential to the overall performance of these methods, despite being often only marginally discussed. Let us denote with CDF_D the empirical cumulative distribution function (CDF) relative to a single feature of the dataset, and with $\mathrm{CDF}_{\mathcal{N}}$ the CDF of a standard normal distribution. The quantile normalization of a value x is defined as:

$$\mathrm{QN}(x) = \mathrm{CDF}_{\mathcal{N}}^{-1}(\mathrm{CDF}_D(x)) \tag{2.55}$$

Notice that this transformation is invertible, and, when applied to each feature, it embeds data in a space where each feature is (marginally) normally distributed.

Chapter 3

Training-Free Conditioning of Score-Based Diffusion Models by Neuro-Symbolic Constraints

Summary: This chapter addresses conditional generation from pre-trained score-based diffusion models without retraining. We develop a training-free method enabling sampling under arbitrary logical constraints by combining neuro-symbolic constraint encoding with conditional score approximation. Experiments demonstrate accurate conditional distribution approximation on tabular data and time series, outperforming the main baseline, though results on images are mixed. Indeed, while optimization-based methods generate high-quality samples, they introduce biases when modeling true conditional distributions, which is critical for tabular data.

This chapter is based on the original work *Zero-Shot Conditioning of Score-Based Diffusion Models by Neuro-Symbolic Constraints* (Scassola et al., 2025b).

3.1 Introduction

Score-based (Song and Ermon, 2019) and diffusion (Ho et al., 2020, Sohl-Dickstein et al., 2015) generative models based on deep neural networks have proven effective in modeling complex high-dimensional distributions in various domains. Controlling these models in order to obtain desirable features in samples is often required, still most conditional models require additional constraint-specific training in order to perform conditional sampling.

This represents a limit since either one needs to train an extremely flexible conditional model (as those based on text prompts), or alternatively to train a conditional model for any specific constraint to be enforced. Moreover, these conditional models often lack robustness, since the constraint is only learned through labelled data, even when the constraint is a user-defined function. As a consequence, training-free conditional generation with arbitrary but formal logical constraints specified at inference time is currently hard. This would be useful in different contexts, for example:

- *Tabular data*: generation of entries that obey formal requirements described by logical formulas, without a specific training for every formula.
- *Surrogate models*: using unconditional surrogate models to efficiently sample imposing physical constraints, or exploring scenarios defined by additional constraints.
- *Image generation*: controlling image generation conditioning on formally specified visual features.

Text prompt conditioned image generation with diffusion models (Rombach et al., 2022) has proven extremely flexible and effective, still it lacks fine-grained control and requires a massive amount of labelled samples. Imposing user-defined constraints can alternatively be performed by manipulating the loss function of the generator (regularization), but in this way a specific training is required for any constraint. Recent work focuses on methods to perform guided diffusion on images, without the need to retrain a noise-dependent classifier (Bansal et al., 2023, Graikos et al., 2022, Kadkhodaie and Simoncelli, 2021, Nair et al., 2023), but the performance of the approximation of the conditional distribution is never evaluated.

In this chapter we develop a method for sampling from pre-trained unconditional score-based generative models, enforcing arbitrary user-defined logical constraints, that does not require additional training. Despite being originally designed for tabular data, we also show the application of our method to images and time series. In summary, we present the following key contributions:

- We develop a training-free method for applying constraints to pre-trained unconditional score-based generative models. The method enables sampling approximately from the conditional distribution given a soft constraint.
- We define a general neuro-symbolic language for building soft constraints that corresponds to logical formulas. These constraints are numerically stable, satisfy convenient logical properties, and can be relaxed/hardened arbitrarily through a control parameter.
- We test our method on different types of datasets and constraints, showing good performance on *approximating conditional distributions* on tabular data, while previous methods were rather meant to obtain high-quality samples that satisfy some given constraints.
- Comparing our method with a state-of-the-art method (Bansal et al., 2023), we gather evidence that plug-and-play conditioning techniques designed for images are not necessarily suited for modelling the true conditional distribution. Moreover, we show our neuro-symbolic language to be useful for defining constraints also within this other method.
- We show that our method allows to sample conditioning on constraints that involve multiple data instances.

3.2 Conditional sampling with score-based models

Score-based generative models (Song and Ermon, 2019, Song et al., 2021) and diffusion models (Ho et al., 2020, Sohl-Dickstein et al., 2015) have been shown to be effective generative models in various domains, and different methods for conditional sampling have been proposed. Given a joint distribution $p(\mathbf{x}_0, \mathbf{y})$, one is often interested in sampling from the conditional distribution

$p(\mathbf{x}_0 \mid \mathbf{y})$, where $\mathbf{y}$ is for example a label. For instance, $p(\mathbf{x}_0 \mid \mathbf{y})$ could be the distribution of CelebA faces given the labels $\mathbf{y}$ including, for example, whether the face is smiling and the hair color. Let $p_t(\mathbf{x}_t \mid \mathbf{y}) = \int p_t(\mathbf{x}_t \mid \mathbf{x}_0) p(\mathbf{x}_0 \mid \mathbf{y}) \, \mathrm{d}\mathbf{x}_0$ be the conditional distribution of noisy samples at time t obtained by corrupting samples from $p(\mathbf{x}_0 \mid \mathbf{y})$ with a diffusion kernel $q(\mathbf{x}_t \mid \mathbf{x}_0)$. While it is possible to directly model the conditional distribution (as usually done in many generative models) by estimating $\nabla_{\mathbf{x}} \ln p_t(\mathbf{x}_t \mid \mathbf{y})$, score-based generative models allow conditional sampling without explicit training of the conditional generative model. Applying the Bayesian rule $p_t(\mathbf{x}_t \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{x}_t) p_t(\mathbf{x}_t)}{p(\mathbf{y})}$ one can observe that:

$$\nabla_{\mathbf{x}_t} \ln p_t(\mathbf{x}_t \mid \mathbf{y}) = \nabla_{\mathbf{x}_t} \ln p_t(\mathbf{x}_t) + \nabla_{\mathbf{x}_t} \ln p(\mathbf{y} \mid \mathbf{x}_t) \tag{3.1}$$

It follows that one can obtain the conditional score from the unconditional score by separately training a noise-dependent classifier $p_t(\mathbf{y} \mid \mathbf{x}_t)$. This technique is known as "guidance", and it has been used for class conditional image generation (Dhariwal and Nichol, 2021).

In many applications, one is interested in sampling from a distribution $p(\mathbf{x})$ conditioning on, or maximizing, some desired measurable properties of the data, that are not necessarily class labels or parts of the given data $\mathbf{y}$. For example, one may want to sample an image $\mathbf{x}$ knowing its downscaled version $\mathbf{d}(\mathbf{x})$, or a molecule $\mathbf{x}$ that maximizes the binding affinity $a(\mathbf{x})$ to a target protein. In tabular data, one may want to sample entries $\mathbf{x}$ where features are consistent with a set of logical or geometrical constraints, as in Stoian et al. (2024) where the compliance is guaranteed by the structure of the output layers (Constrained Deep Generative Models). This task can often be formalized as the problem of sampling from the target product distribution:

$$p^c(\mathbf{x}) \propto p(\mathbf{x}) c(\mathbf{x}) \tag{3.2}$$

where $c(\mathbf{x}) : \mathbb{X} \to \mathbb{R}_+$ is a function expressing the degree of satisfaction of the desired property.

Several methods based on score-based generative models and diffusion models have been proposed to tackle this problem. In Sohl-Dickstein et al. (2015), the article that introduced diffusion models, they suggest a method to sample from the product of a distribution learned with a diffusion model $p(\mathbf{x})$ and a given distribution or bounded positive function $r(\mathbf{x})$. The method consists in modifying the reverse diffusion process by multiplying the reverse kernels $q(\mathbf{x}_{t-1} | \mathbf{x}_t)$ by $r(\mathbf{x})$, using an approximation when this cannot be done analytically. In practice they apply this method only to the particular case of inpainting, that is easily solvable with heuristics. Controllable generation with score-based models is also discussed in Song et al. (2021), where they both treat the case when the noise conditional classifier $p_t(\mathbf{y} \mid \mathbf{x}_t)$ can be trained and when it is not available. In that case, a method for obtaining such an estimate without the need of training auxiliary models is discussed and applied to conditional generation tasks such as inpainting and colorization. Still, the estimate is applicable only assuming the possibility to define $\mathbf{y}_t$ such that $p(\mathbf{y}_t | \mathbf{y})$ and $p(\mathbf{x}_t | \mathbf{y}_t)$ are tractable. In Janner et al. (2022) they cast a reinforcement learning problem of finding optimal state-action trajectories into a conditional sampling problem. Trajectories that maximize a reward function are generated, using the guidance of a noise-dependent classifier trained to predict such reward. One of the first works on general training-free conditional generation with diffusion models is Graikos et al. (2022), where samples from $p_c(\mathbf{x})$ are obtained by optimizing an initial random sample $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ with respect to a free energy term including the constraint value $\log c(\mathbf{x})$ and another term depending on the denoising network. The authors show applications on image generation tasks and combinatorial optimization.

Another series of works on inverse problems focused on modifying the sampling process of pretrained diffusion models by adding to the score a guidance term dependent on the constraint $\nabla_{\mathbf{x}_t} c(\mathbf{x}_t, t)$ with increasingly complex heuristics (Bansal et al., 2023, Chung and Ye, 2022, He

et al., 2023, Song et al., 2023, Yu et al., 2023). We discuss in Section 3.4.4 the approach of Bansal et al. (2023) that represents a synthesis of previous techniques (Ye et al., 2024).

Earlier works explored the combination of unconditional latent variables generative models such as Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) or Variational Autoencoders (VAEs) (Kingma and Welling, 2014) with constraints to produce conditional samples (Engel et al., 2017). Given a latent variables model $f(\mathbf{z})$ that maps latent codes $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to data points $\mathbf{x} \sim p(\mathbf{x})$, they optimize the latent code $\mathbf{z}$ to maximize $c(f(\mathbf{z}))$. The advantage consists in performing the optimization in the latent space, that is smoother and easier to sample valid samples from. This framework has been extended to diffusion-like models and flow matching, where a latent variables interpretation is possible (Draxler et al., 2024, Liu et al., 2023b). In flow matching, the integration of an Ordinary Differential Equation (ODE) can be interpreted as the map between latent codes and the data space. In this case optimizing the objective $c(f(\mathbf{z}))$ requires backpropagating through the ODE solver, that is computationally challenging for a large number of steps in a high-dimensional space.

Despite these works focused on sampling high-quality samples that satisfy some given properties, it was not verified if samples followed the correct conditional distributions, a more difficult task that is relevant for example when dealing with tabular data and time series. Given that these methods are often based on the introduction of an optimization phase in the original sampling process, it is not guaranteed that the true conditional distribution will be well approximated. Moreover, these methods are often fitted for imaging problems, and were not tested on different kinds of data and constraints. Despite the existence of many related prior works with different focuses, our goal is different and more challenging: obtaining samples distributed according to the target conditional distribution, while previous methods rather aim at obtaining high-quality samples. To our knowledge, there are no works focusing on the correct approximation of the conditional distribution for tabular data as we do. The best comparison we can do is with Bansal et al. (2023), since it is the state of the art for diffusion-based training-free conditional generation and a synthesis of previous techniques. We show that our method is significantly better with tabular data when the objective is the approximation of the conditional distribution.

3.3 Method

3.3.1 Problem formalization

Given a set of observed samples $\mathbf{x}_i \in \mathbb{R}^d$, the goal is to sample from the distribution $p(\mathbf{x})$ that generated $\mathbf{x}_i$, conditioning on a desired property. Let $\pi(\mathbf{x}) : \mathbb{R}^d \to \{0, 1\}$ be the function that encodes this property, such that $\pi(\mathbf{x}) = 1$ when the property is satisfied and $\pi(\mathbf{x}) = 0$ otherwise. Then the target conditional distribution can be defined as:

$$p(\mathbf{x} \mid \pi) \propto p(\mathbf{x})\pi(\mathbf{x}) \tag{3.3}$$

Alternatively, one can also define soft constraints, expressing the degree of satisfaction as a real number. Let $c(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function expressing this soft constraint. In this case we define the target distribution as:

$$p^c(\mathbf{x}) \propto p(\mathbf{x})e^{c(\mathbf{x})} \tag{3.4}$$

It can be shown that p^c is the distribution that maximizes $\mathbb{E}_{p^c}[c(\mathbf{x})] - D_{KL}(p^c||p)$ (Ganchev et al., 2010, Hu et al., 2018). Moreover, since the form is analogous to the previous formulation, given a

hard constraint $\pi(\mathbf{x})$ one can build a soft constraint $c(\mathbf{x})$ such that $p^c(\mathbf{x}) \approx p(\mathbf{x} \mid \pi)$ and $c(\mathbf{x})$ is differentiable in $\mathbb{R}^d$. This will prove useful in the following, since the proposed method requires differentiable constraints. We then consider $p^c(\mathbf{x})$ as the target distribution we want to sample from.

3.3.2 Constraint-based guidance

Our method exploits score-based generative models as the base generative model. As previously introduced, a stochastic process that gradually adds noise to the original data $p_t(\mathbf{x}_t \mid \mathbf{x}_0)$ is defined such that at $t = 0$ no noise is added so $X_0 \sim p_0(\mathbf{x}_0 \mid \mathbf{x}_0) = \delta(\mathbf{x}_0)$ and at $t = 1$ the maximum amount of noise is added such that $X_1 \sim p_1(\mathbf{x}_1 \mid \mathbf{x}_0)$ is a known prior distribution (for example a Gaussian). Given the possibility to efficiently sample from $p_t(\mathbf{x}_t \mid \mathbf{x}_0)$, the time-dependent score of $p_t(\mathbf{x}_t)$ is estimated by score matching using a neural network, denoted $\mathbf{s}(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \ln p_t(\mathbf{x}_t)$. As discussed in Section 2.2.3, there are different possible sampling schemes once the score is available. Given the target distribution:

$$p^c(\mathbf{x}) := \frac{p(\mathbf{x})e^{c(\mathbf{x})}}{Z} \tag{3.5}$$

where Z is the unknown normalization constant, and the distribution of samples from $p^c(\mathbf{x})$ that are successively corrupted by $p_t(\mathbf{x}_t \mid \mathbf{x}_0)$:

$$p_t^c(\mathbf{x}_t) := \int p_t(\mathbf{x}_t \mid \mathbf{x}_0) p^c(\mathbf{x}_0) \mathrm{d}\mathbf{x}_0 \tag{3.6}$$

we observe the following relationship:

$$\nabla_{\mathbf{x}_0} \ln p_0^c(\mathbf{x}_0) = \nabla_{\mathbf{x}} \ln p^c(\mathbf{x}) = \nabla_{\mathbf{x}} \ln \frac{p(\mathbf{x})e^{c(\mathbf{x})}}{Z} \tag{3.7}$$

$$= \nabla_{\mathbf{x}} [\ln p(\mathbf{x}) + c(\mathbf{x}) - \ln Z] = \nabla_{\mathbf{x}} \ln p(\mathbf{x}) + \nabla_{\mathbf{x}} c(\mathbf{x}) \tag{3.8}$$

It follows that at $t = 0$ one can easily obtain an estimate of the score by summing the gradient of the constraint to the estimate of the score of the unconstrained distribution. Notice that this is possible since taking the gradient of the logarithm eliminates the intractable integration constant Z. At $t = 1$ instead one can safely assume $\nabla_{\mathbf{x}_1} \ln p_1^c(\mathbf{x}_1) = \nabla_{\mathbf{x}_1} \ln p_1(\mathbf{x}_1)$, since the forward diffusion process is designed to add enough noise to make samples from any distribution distributed as the prior $p_1(\mathbf{x}_1)$. In general, there is no analytical form for $\nabla_{\mathbf{x}_t} \ln p_t^c(\mathbf{x}_t)$, and it cannot be estimated by score matching because we do not assume samples from $p^c(\mathbf{x})$ are available in the first place.

3.3.3 Conditional score approximation

Given this limitation, we resort to approximations $\tilde{\mathbf{s}}_c(\mathbf{x}_t, t)$ for $\mathbf{s}_c(\mathbf{x}_t, t) = \nabla_{\mathbf{x}_t} \ln p_t^c(\mathbf{x}_t)$[1]. The approximations we use are constructed knowing the true value of the score for $t = 0$ and $t = 1$:

$$\tilde{\mathbf{s}}_c(\mathbf{x}_0, 0) = \mathbf{s}(\mathbf{x}_0, 0) + \nabla_{\mathbf{x}_0} c(\mathbf{x}_0) \tag{3.9}$$

$$\tilde{\mathbf{s}}_c(\mathbf{x}_1, 1) = \mathbf{s}(\mathbf{x}_1, 1) \tag{3.10}$$

[1]Here we explicitly show t as an argument of the score function to highlight the dependence, while for density functions this is often implicit in the name of the variable $\mathbf{x}_t$.

A simple way to obtain this is by weighting the contribution of the gradient of the constraint depending on time:

$$\tilde{\mathbf{s}}_c(\mathbf{x}_t, t) = \mathbf{s}(\mathbf{x}_t, t) + g(t)\nabla_{\mathbf{x}_t} c(\mathbf{x}_t) \tag{3.11}$$

where $g(t) : [0,1] \to [0,1]$ satisfies $g(0) = 1$ and $g(1) = 0$. This is equivalent to extending the domain of the constraint to noisy data points $c(\mathbf{x}_t, t)$ and then approximating it with $c(\mathbf{x}_t, t) = g(t)c(\mathbf{x}_t)$. Sampling from the target distribution then reduces to substituting the score of the base model with the modified score $\tilde{\mathbf{s}}_c(\mathbf{x}_t, t)$ and then using any of the aforementioned sampling methods. Notice that this approach does not require any retraining of the model. The only necessary ingredients are the unconditional score model $\mathbf{s}(\mathbf{x}_t, t)$ and the differentiable constraint $c(\mathbf{x}_t)$ encoding the degree of satisfaction of the desired property. Intuitively, this method tries to approximate the guidance of Dhariwal and Nichol (2021) without training a classifier on noisy instances. We instead approximate it by weighting the contribution of the gradient of the constraint with respect to the data point, depending on its noise level t. We justify the weighting as a way to smoothly interpolate between the two unapproximated scores at the extremes $t = 0$ and $t = 1$. Intuitively, we want to weight the direction given by the gradient of the constraint more as the constraint is evaluated on clean data, and less when evaluated on corrupted data, since the evaluation of the constraint on corrupted data is less meaningful.

Multiple instances constraints

We may also want to sample multiple instances $\mathbf{v} = (\mathbf{x}^1, \dots, \mathbf{x}^n)$ that are tied together by a single multivariate constraint $c(\mathbf{v})$, i.e., sampling from: $p^c(\mathbf{v}) \propto p(\mathbf{v})e^{c(\mathbf{v})} = e^{c(\mathbf{v})} \prod_{i=1}^n p(\mathbf{x}^i)$. In this case, it is easy to show that

$$\nabla_{\mathbf{x}_t^i} \ln p_t^c(\mathbf{x}_t^i) = \nabla_{\mathbf{x}_t^i} \ln p_t(\mathbf{x}_t^i) + \nabla_{\mathbf{x}_t^i} c(\mathbf{x}_t^1, \dots, \mathbf{x}_t^n) \tag{3.12}$$

Therefore, the approximated score for each instance $\mathbf{x}_t^i$, $i \in \{1, \dots, n\}$ is:

$$\tilde{\mathbf{s}}_c(\mathbf{x}_t^i, t) = \mathbf{s}(\mathbf{x}_t^i, t) + g(t)\nabla_{\mathbf{x}_t^i} c(\mathbf{x}_t^1, \dots, \mathbf{x}_t^n) \tag{3.13}$$

This can be computed in parallel for each instance $\mathbf{x}_t^i$, as the computation of $\mathbf{s}(\mathbf{x}_t^i, t)$ can be parallelized by batching the score network and $\nabla_{\mathbf{x}_t^i} c(\mathbf{x}_t^1, \dots, \mathbf{x}_t^n)$ is just a component of $\nabla_{\mathbf{v}} c(\mathbf{v})$. When sampling, the instances will also be generated in parallel, as if they were a single instance. This is similar to what already happens with batches of samples, but in this case these are made dependent through a constraint that involves them all.

Finally, we remark that the additional computation at each step only consists in the computation of the gradient of the constraint, making the time complexity equal to the original sampling algorithm.

Langevin dynamics correction.

Depending on the type of data, we found it useful to perform additional Langevin dynamics steps (these are referred to as "corrector steps" in Song et al. (2021)) at time $t = 0$ when the score of the constrained target distribution is known without approximation. Langevin dynamics can be used as an approximate Monte Carlo method for sampling from a distribution when only the score is known (Parisi, 1981), performing the following update, where ϵ is the step size and $\mathbf{z}^i$ is sampled from a standard normal with the same dimensionality as $\mathbf{x}$:

$$\mathbf{x}^{i+1} = \mathbf{x}^i + \epsilon\, \tilde{\mathbf{s}}_c(\mathbf{x}, 0) + \mathbf{z}^i\sqrt{2\epsilon} \tag{3.14}$$

In the limit $i \to \infty$ and $\epsilon \to 0$, Langevin dynamics samples from $p^c(\mathbf{x})$. Nevertheless, as with most Monte Carlo techniques, Langevin dynamics struggles to explore all the modes when the target distribution is highly multimodal. Algorithm 3 summarizes the modified sampling algorithm. Notice that the time complexity of the sampling algorithm is unaffected. Training an accurate score model at $t \approx 0$ is crucial to make this correction effective in practice. This is particularly challenging and we discuss potential solutions in Section 3.3.5.

Choice of score approximation scheme.

We observed the results to be sensitive to the choice of the approximation scheme for $\nabla_{\mathbf{x}} \ln p_t^c(\mathbf{x}_t)$. One should choose a $g(t)$ that is strong enough to guide samples towards the modes of $p_0^c(\mathbf{x}_0)$ but at the same time does not disrupt the reverse diffusion process in the early steps. We experimented with various forms of $g(t)$, mostly with the following two functions:

- **Linear**: $g(t) = 1 - t$
- **SNR**: $g(t)$ depends on the signal-to-noise ratio of the diffusion kernel at time t. More specifically, given a diffusion kernel of the form $p(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; k_t\mathbf{x}_0, \sigma_t)$, we define it as

$$g(t) = \frac{k_t}{\sqrt{k_t^2 + \sigma_t^2}} \tag{3.15}$$

  This can be interpreted as the ratio between the signal of the original data point and the total magnitude of the noisy data point, assuming normalized data. Intuitively, $g(t)$ defined in this way is a proxy for how much information about the original data point is still present in the noisy data point at time t. Notice that for both the variance-exploding and the variance-preserving diffusion kernels $g(0) = 1$ and $g(1) \approx 0$. Figure 3.1 shows the behavior of $g(t)$ for the two kernels.

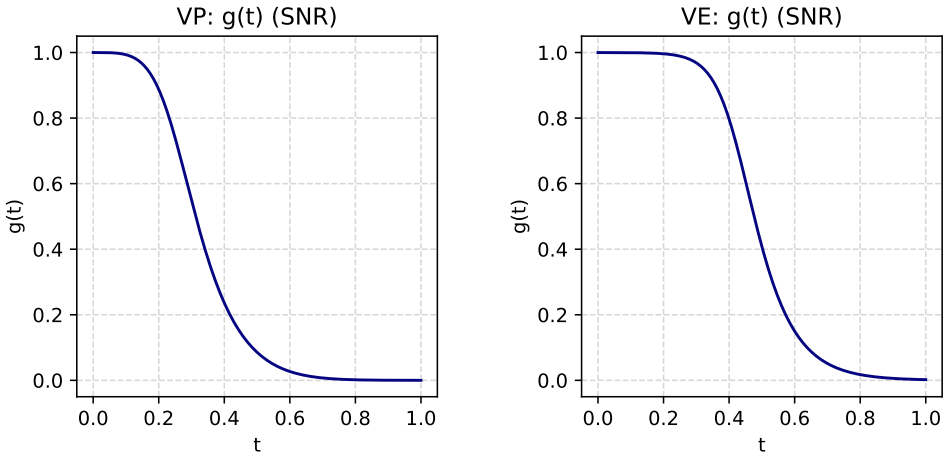


Figure 3.1: Behavior of $g(t)$ with the SNR strategy for the variance-exploding and variance-preserving diffusion kernels. Notice that for both kernels, $g(0) = 1$ and $g(1) \approx 0$.

In many of our experiments we found **SNR** to be the most effective, so we suggest using it as the first choice.

Algorithm 3 Constraint guidance sampling

Input: constraint $c(\mathbf{x})$, score $\mathbf{s}(\mathbf{x}, t)$, score-based sampling algorithm $A(\mathbf{s})$
Parameters: $g(t)$, ϵ, n
$\tilde{\mathbf{s}}_c(\mathbf{x}, t) \leftarrow \mathbf{s}(\mathbf{x}, t) + g(t)\nabla_{\mathbf{x}} c(\mathbf{x})$
$\mathbf{x} \leftarrow A(\tilde{\mathbf{s}}_c)$
for $i = 1$ **to** n **do**
  $\mathbf{z} \leftarrow \mathcal{N}(0, 1)$ (with the same dimensionality as $\mathbf{x}$)
  $\mathbf{x} \leftarrow \mathbf{x} + \epsilon\tilde{\mathbf{s}}_c(\mathbf{x}, 0) + \mathbf{z}\sqrt{2\epsilon}$
return $\mathbf{x}$

3.3.4 Neuro-Symbolic logical constraints

We will consider a general class of constraints expressed in a logical form. Hard logical constraints cannot be directly used in the approach presented above, hence we turn them into differentiable soft constraints leveraging neuro-symbolic ideas (Badreddine et al., 2020). More specifically, we consider predicates and formulae defined on the individual features $\mathbf{x} = (x_1, \ldots, x_d)$ of the data points we aim to generate. Given a Boolean property $\pi(\mathbf{x})$ (i.e., a predicate or a formula), with features $\mathbf{x}$ as free variables, we associate with it a constraint function $c(\mathbf{x})$ such that $e^{c(\mathbf{x})}$ approximates the corresponding non-differentiable hard constraint $\mathbf{1}_{P(\mathbf{x})}$.[2] In this article, we consider constraints that can be evaluated on a single or on a few data points, hence we can restrict ourselves to the quantifier-free fragment of first-order logic. Therefore, we can define by structural recursion the constraint $c(\mathbf{x})$ for atomic propositions and Boolean connectives. As atomic propositions, we consider here simple equalities and inequalities of the form $a(\mathbf{x}) \geq b(\mathbf{x})$, $a(\mathbf{x}) \leq b(\mathbf{x})$, $a(\mathbf{x}) = b(\mathbf{x})$, where a and b can be arbitrary differentiable functions of feature variables $\mathbf{x}$. Following Badreddine et al. (2020), we refer to such sets of functions as *real logic*.

In particular, we define the semantics directly in log-probability space, obtaining the **Log-probabilistic logic**. The definitions we adopt are partially in line with those of the newly defined *LogLTN* in concurrent work by (Badreddine et al., 2023), and are reported in Table 3.1.

Table 3.1: Semantic rules of log-probabilistic logic. In the table, $c[\varphi](\mathbf{x})$ is the soft constraint associated with the formula φ.

Formula	Differentiable function
$c[a(\mathbf{x}) \geq b(\mathbf{x})]$	$-\ln(1 + e^{-k(a(\mathbf{x})-b(\mathbf{x}))})$
$c[a(\mathbf{x}) \leq b(\mathbf{x})]$	$-\ln(1 + e^{-k(b(\mathbf{x})-a(\mathbf{x}))})$
$c[a(\mathbf{x}) = b(\mathbf{x})]$	$a(\mathbf{x}) \geq b(\mathbf{x}) \wedge a(\mathbf{x}) \leq b(\mathbf{x})$
$c[\varphi_1 \wedge \varphi_2]$	$c[\varphi_1] + c[\varphi_2]$
$c[\varphi_1 \vee \varphi_2]$	$\ln(e^{c[\varphi_1]} + e^{c[\varphi_2]} - e^{c[\varphi_1]+c[\varphi_2]})$
$c[\neg\varphi]$	$\ln(1 - e^{c[\varphi]})$

Atomic predicates.

We choose to define the inequality $a(\mathbf{x}) \geq b(\mathbf{x})$ as $c(\mathbf{x}) = -\ln(1 + e^{-k(a(\mathbf{x})-b(\mathbf{x}))})$, introducing an extra parameter k that regulates the "hardness" of the constraint. Indeed, in the limit $k \to$

[2] $\mathbf{1}_{P(\mathbf{x})}$ is the indicator function equal to 1 for each $\mathbf{x}$ such that $P(\mathbf{x})$ is true.

∞ one has $\lim_{k\to\infty} e^{c(\mathbf{x})} = \mathbf{1}_{a(\mathbf{x})\geq b(\mathbf{x})}$. Although this definition allows consistency with the negation, the gradient of the inequality is nonzero when the condition is satisfied, although this did not create issues in the experiments for sufficiently large values of k. This is due to the fact that samples are pushed slightly past the satisfaction threshold, where the gradient approaches zero more rapidly for large values of k. In order to have a null gradient when the condition is satisfied, one can define a simplified version: $a(\mathbf{x}) \geq b(\mathbf{x}) \equiv k(a(\mathbf{x}) - b(\mathbf{x}))\mathbf{1}_{a(\mathbf{x})<b(\mathbf{x})}$, however such a definition will no longer be consistent with negation. The definition of the equality was based on inequalities in order to maintain consistency. We also experimented with a definition based on the L2 distance $-(a(\mathbf{x}) - b(\mathbf{x}))^2$, corresponding to a Gaussian kernel, even though this form does not benefit from a bounded gradient.

Boolean connectives.

The conjunction and the disjunction correspond to the product t-norm and its dual t-conorm (probabilistic sum) (van Krieken et al., 2022) but in logarithmic space. We use the material implication rule to reduce the logical implication to a disjunction: $a \to b \equiv \neg a \vee b$. The negation, instead, is consistent with the semantic definition of inequalities: negating one inequality, one obtains its flipped version. For numerical stability reasons, however, we choose to avoid using the soft negation function in practice. Instead, we reduce logical formulas to the negation normal form (NNF) as in Badreddine et al. (2023), where negation is only applied to atoms, for which the negation can be computed analytically or imposed by definition. In order to simplify the notation, in the following we will also use quantifiers ($\forall$ and $\exists$) as syntactic sugar (in place of *finite* conjunctions or disjunctions) only when the quantified variable takes values in a finite and known domain (e.g., time instants in a time series or pixels in an image). So $\forall i \in \{1, \ldots, n\}\ p_i$ is used as a shorthand for $p_1 \wedge p_2 \wedge \ldots \wedge p_n$ and $\exists i \in \{1, \ldots, n\} : p_i$ for $p_1 \vee p_2 \vee \ldots \vee p_n$.

The difference in our definition with respect to *LogLTN* is in the logical disjunction ($\vee$): they define it using the LogMeanExp (LME) operator, an approximation of the sum in logarithmic space that is numerically stable and suitable for differentiation. They do it at the cost of losing the possibility to reduce formulas to the NNF exactly (using De Morgan's laws) that follows from having as disjunction the dual t-conorm of the conjunction. We choose instead to use the log-probabilistic sum as the disjunction, since in the domain of our experiments it proved numerically stable and effective. In Appendix A.6 we show how to implement the logical disjunction in a numerically stable way.

When sampling, we can regulate the trade-off between similarity with the original distribution and strength of the constraint by tuning the parameter k of inequalities in log-probabilistic logic. Alternatively, we can multiply by a constant λ the value of the constraint in order to scale its gradient.

Notice that the flexibility of this logical framework allows the definition of constraints involving multiple data instances. Figure 3.2 shows an example of inequality and equality for different values of k, while Figure 3.3 shows examples of logical formulas combining inequalities with conjunctions and disjunctions.

3.3.5 Training a score model for tabular data

Score-based models have mainly been used for image generation; adapting them to tabular data and time series requires special care. In particular, the challenge is to correctly model the noise of the target distribution, implying the tricky task of estimating the score at $t \approx 0$ (no noise).

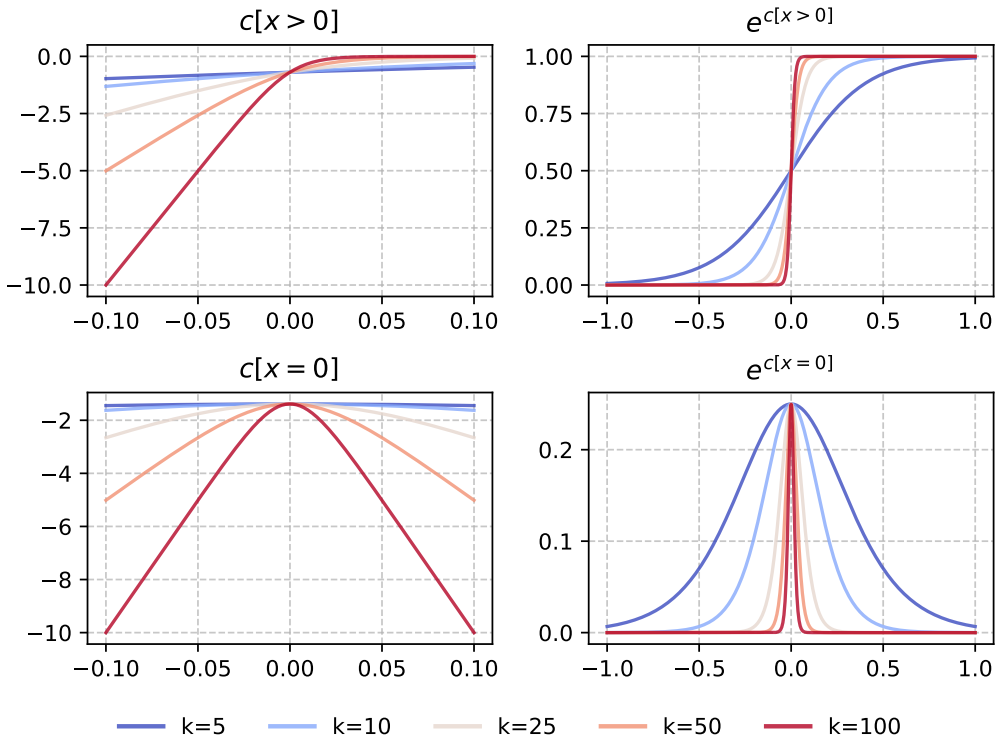


Figure 3.2: Examples of log-probabilistic logic functions for different values of k. In the right panels we show the exponential of these functions.

We improved the score estimate mainly by carefully parametrizing the score network and using large batches. We provide more details in Appendix A.3. Correctly estimating the score at $t \approx 0$ is crucial to make our method work in practice, since it allows us to perform Langevin dynamics at $t \approx 0$, where the conditional score is known without approximation. Combining a correct score estimation at $t \approx 0$ with many steps of Langevin dynamics allows us to (asymptotically) sample from the exact conditional distribution, in particular when the data distribution is not particularly multimodal.

3.4 Experiments

We tested our method on several datasets, though evaluating the quality of conditionally generated samples is challenging. First of all, one should compare conditionally generated samples with another method that generates them exactly. We then chose to compare our approach with rejection sampling, which can be used to sample exactly from the product of two distributions $p(\mathbf{x})$ and $q(\mathbf{x})$, where sampling from $p(\mathbf{x})$ is tractable and the density of $q(\mathbf{x})$ is known up to a normalization constant. In our case $p(\mathbf{x})$ is the unconditional generative model and $q(\mathbf{x}) = e^{c(\mathbf{x})}$. Assuming constraints are such that $\forall \mathbf{x}\ c(\mathbf{x}) \leq 0$, then $q(\mathbf{x})$ is upper-bounded by 1. This upper bound is guaranteed by the real logic we defined previously. Rejection sampling then reduces to sampling from $p(\mathbf{x})$ and accepting each sample with probability $q(\mathbf{x})$. This can be problematic when the probability of a random sample from $p(\mathbf{x})$ having a nonzero value of $q(\mathbf{x})$ is low.

Secondly, comparing the similarity of two samples is not trivial, as discussed in Section 2.5. For relatively low-dimensional samples, we compare the marginal distributions and the correlation

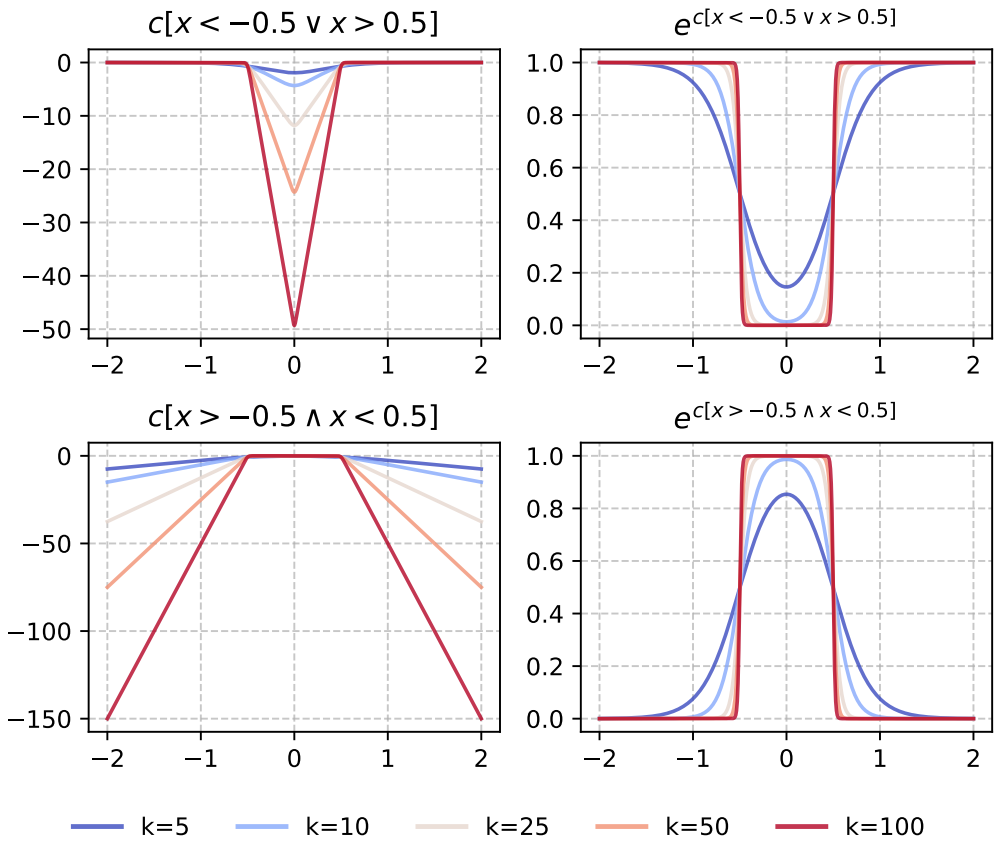


Figure 3.3: Examples of log-probabilistic logic formulas for different values of k. In the right panels we show the exponential of these functions.

matrix. To compare one-dimensional distributions between two samples X and Y, we use the Total Variation Distance (TVD), defined as:

$$\mathrm{TVD}(X, Y) := \frac{1}{2} \sum_i |x_i - y_i| \tag{3.16}$$

where x_i and y_i are the empirical probabilities for a given common binning. This distance is upper-bounded by 1. When computationally feasible, we consider rejection sampling as the baseline method. By reporting the acceptance rate of rejection sampling, we show the satisfaction rate of the constraint on data generated by the original model. Moreover, in Section 3.4.4 we discuss a comparison with Bansal et al. (2023), which is arguably the state-of-the-art method for training-free conditional generation of images.

We mostly used unconditional models based on denoising score matching and SDEs, following closely Song et al. (2021). Conditional generation hyperparameters and more details about models are described in Appendix A.5 and A.2.

As an explanatory example, we show in Figure 3.4 the result of our method on a toy dataset in $\mathbb{R}$, consisting of samples from a mixture of two Gaussians, with a simple inequality constraint.

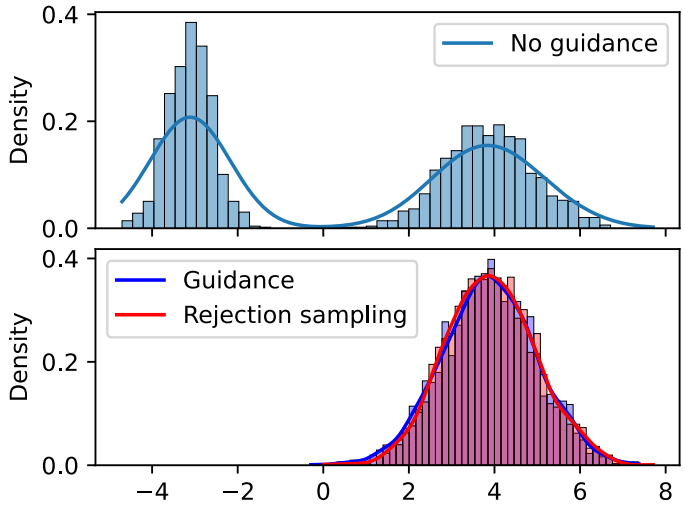


Figure 3.4: Constraint guidance on a toy dataset. We fitted a score-based generative model on data generated according to a mixture of two Gaussian distributions $\mathcal{N}(\mu = -3, \sigma = 0.5)$, $\mathcal{N}(\mu = 4, \sigma = 1)$ with equal mixing coefficients. Given the constraint $x \geq 0$, our method generates samples that are not distinguishable from those generated with rejection sampling. We also show the generated distribution without guidance.

3.4.1 Tabular data

White Wine Dataset. We conducted experiments with the white wine table of the popular UCI Wine Quality dataset (Paulo et al., 2009), consisting of 11 real-valued dimensions ($\mathbb{R}^{11}$) and one discrete dimension, the quality, which we discarded. In order to evaluate the effectiveness with categorical variables, we also conducted experiments with the Adult dataset (Becker and Kohavi, 1996), consisting of 5 numerical dimensions and 10 categorical dimensions, which we embedded in a continuous space using one-hot encodings. We fitted unconditional score-based diffusion models based on SDEs, then we generated samples under illustrative logical constraints.

First, we generated samples from the white wine model under the following complex logical constraint:

$$(\text{fixed acidity} \in [5.0, 6.0] \vee \text{fixed acidity} \in [8.0, 9.0]) \wedge \\ \text{alcohol} \geq 11.0 \wedge (\text{residual sugar} \leq 5.0 \rightarrow \text{citric acid} \geq 0.5) \quad (3.17)$$

We show in Figure 3.5 the marginals of the features mentioned in the constraint of the generated samples, compared with samples generated by rejection sampling. Table 3.2 shows the TVD of the marginal of each dimension. There is a high overlap between marginals for most dimensions, and we measured an average L1 distance between correlation coefficients of ≈ 0.07. The largest error, which is associated with one of the dimensions heavily affected by the constraint, is still relatively small. For that constraint, the acceptance rate of rejection sampling was only $\approx 1.67\%$, meaning that our method samples efficiently in low-probability regions. The satisfaction rates of the relative hard constraint were similar: $\approx 92\%$ for rejection sampling and $\approx 86\%$ for our method (these can be increased by increasing the parameter k).

Notice that the satisfaction rate of samples generated with rejection sampling and soft constraints is not 100%, highlighting the approximation error of the soft constraint with respect to the hard one. This is probably amplified by the complexity of the constraint, as we observed inferior approximation error for simpler constraints. In order to compensate for the fuzzy nature of the

constraints, one can either increase the parameter k, add a "safety margin" to the thresholds appearing in the formula, or simply discard the few samples not satisfying the hard constraint.

Table 3.2: White wine experiment: TVD for different columns. This table compares the marginal distributions of the 5000 samples generated with constraint guidance with the same number of samples generated with rejection sampling.

Column	**TVD**
Fixed Acidity	0.049
Volatile Acidity	0.052
Citric Acid	0.13
Residual Sugar	0.15
Chlorides	0.077
Free Sulfur Dioxide	0.094
Total Sulfur Dioxide	0.11
Density	0.10
pH	0.087
Sulphates	0.12
Alcohol	0.11

White Wine Dataset: Multi-Instance Constraint. Additionally, we tested the application of a simple multi-instance constraint acting on a pair of data points $\mathbf{x}_1$ and $\mathbf{x}_2$:

$$\text{alcohol}(\mathbf{x}_1) > \text{alcohol}(\mathbf{x}_2) + 1 \tag{3.18}$$

Our method was able to generate samples that follow closely the correct target distribution, as we can see from the low errors in terms of TVD of the marginals and average L1 distance of correlation coefficients reported in Table 3.3. The relative hard constraint was met in 99% of the generated samples, compared to the 27% of the unconditioned model.

Table 3.3: Multiple-instances constraint on wine dataset. Considering the constraint $\text{alcohol}(\mathbf{x}_1) > \text{alcohol}(\mathbf{x}_2) + 1$, the first column shows the error measures relative to the dimensions of the first element of the pair $\mathbf{x}_1$, while the second shows them for $\mathbf{x}_2$.

Metric	$\mathbf{x}_1$	$\mathbf{x}_2$
Average TVD	0.06	0.061
Median TVD	0.058	0.063
Max TVD	0.074	0.086
Average L1 corr.	0.025	0.026

Adult Dataset. We further assessed the effectiveness of our method by generating samples from the Adult model under the following logical constraint:

$$\text{age} \geq 40 \wedge (\text{race} \neq \text{"White"} \vee \text{education} = \text{"Masters"}) \tag{3.19}$$

In this case, equalities and inequalities involving discrete components are obtained by imposing the desired component of the corresponding one-hot encoding equal to 1 for equality, or to 0 for

inequality. The median TVD was ≈ 0.05, the maximum was ≈ 0.13 and the error in correlations was negligible. The relative hard constraint was met in all samples while the acceptance rate of rejection sampling was 1.73%.

We show in Table 3.4 a summary of the results for tabular data.

Table 3.4: Summary of results for tabular data experiments.

Metric	White wine	White wine (multi inst.)	Adult
Avg TVD	0.1	0.069	0.067
Max TVD	0.153	0.118	0.13
Avg L1 corr	0.07	0.027	0.046
Hard sat. rate	86%	99%	100%
Hard sat. rate (rej. sampling)	92%	100%	100%
Rej. sampling acceptance rate	1.67%	27%	1.73%

3.4.2 Time series surrogate models

A surrogate model is a simplified and efficient representation of a complex, eventually computationally expensive model. It is possible to learn a surrogate model of a complex stochastic dynamical system by fitting a statistical model to a dataset of trajectories observed from it. Following our approach, one can use a score-based generative model to learn an unconditional surrogate model, and then apply constraints to enforce desirable properties. These can be physical constraints the system is known to respect, or features that are rare in unconditioned samples. So we can exploit this method to both assure consistency of trajectories and explore rare (but not necessarily low-density) scenarios. As a case study, we apply our proposed method for the conditional generation of ergodic SIRS (eSIRS) trajectories. The eSIRS model (Kermack et al., 1927) is widely used to model the spreading of a disease in an open population (open in the sense of having infective contacts with external individuals, not part of the modelled population). The model assumes a fixed population of size N composed of Susceptible (S), Infected (I), and Recovered (R) individuals. More formally, we have $S(t) + I(t) + R(t) = N,\ \forall t$.

We consider trajectories with H discretized time steps, therefore the sample space is $\mathcal{X}_{\textbf{eSIRS}} := \left(\mathbb{N}_0{}^2\right)^H$, where the two dimensions are S and I (R is implicit since $R = N - S - I$).

First we train a score-based generative model to fit trajectories that were generated by a simulator, with $H = 30$ and $N = 100$. Then we experimented with the application of different constraints, including the following consistency constraints:

- *Non-negative populations*: $\forall t\ S(t) \geq 0 \wedge I(t) \geq 0$
- *Constant population*: $\forall t\ S(t) + I(t) \leq N$

We show in Figure 3.6 and Figure 3.7 two experiments with two different constraints. In both experiments the consistency constraints (positive and constant population) were always met, with a small improvement over the unconditional model. This means that this technique can be useful to enforce constraints that the data is known to satisfy, using prior knowledge to compensate for imperfections of the unconditional model. In the two experiments we additionally imposed a bridging constraint and an inequality, which were also met with minimal error.

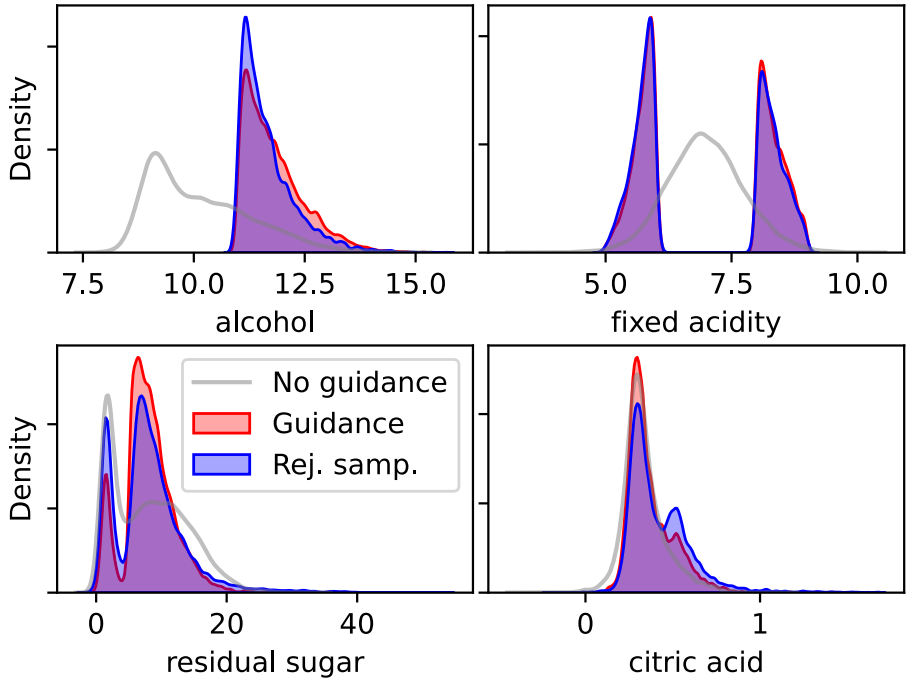


Figure 3.5: Marginals of white wine data experiment. We generated 5000 samples using our constrained sampling algorithm and as many by rejection sampling. The plot compares the marginals of the dimensions directly involved in the constraint. The last two dimensions are the ones with the largest TVD with respect to rejection sampling marginals: 0.15 and 0.13, while the median distance across all dimensions is ≈ 0.1. In order to evaluate the noise of the distance, we also measured the self-distance between two equally sized samples obtained by rejection sampling and observed a median across dimensions of ≈ 0.05.

3.4.3 Images

We test our method on image datasets in order to investigate the potential in high-dimensional data. In this case, the validation of results cannot rely on statistics; therefore, we consider it satisfactory to validate the results by visual inspection of the generated images. First of all, marginals and correlation matrices relative to single pixels are not meaningful indicators of the fidelity of generated samples. Indeed, evaluation methods for generative models for images usually rely on classifier-based metrics such as FID (Heusel et al., 2017) or Inception score (Salimans et al., 2016). Secondly, in the context of images, the quality of individual samples is often considered more important than matching the true underlying distribution. In the following experiments, if the number of invalid or unrealistic samples is relevant, we report the percentage of samples that are invalid or unrealistic over a total of ≈ 100 manually inspected samples. Finally, rejection sampling is often not computationally feasible in this case, due to the extremely low probability of generating valid samples by chance according to most constraints we use. Hence, we cannot compare metrics such as FID or Inception score with samples generated with an exact method.

We use as pre-trained unconditional models a model based on a U-Net that we trained on the MNIST dataset, and a pre-trained model for CelebA (Liu et al., 2015) 64×64 images made available by Song and Ermon (2020).

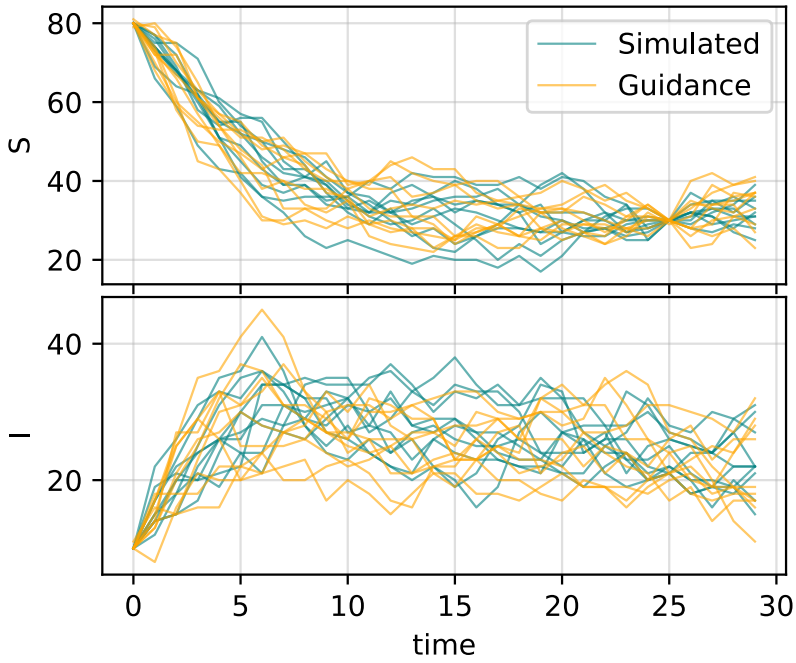


Figure 3.6: Bridging with eSIRS time series. We show here a subsample of the 5000 time series generated with constraint guidance (orange) along with a subsample of the > 50000 time series generated by rejection sampling from the simulator (green). In addition to the consistency constraints, which are always met, we imposed the following equalities: $S(0) = 95$, $I(0) = 5$, $S(25) = 30$. Constraints are generally met: the average L1 absolute difference with all three target values is below 0.19. The TVD for each time step marginal is relatively small, considering that it also accounts for the error of the unconditional model: for S and I, the median TVD across time are ≈ 0.11 and ≈ 0.13. The median self-distance across time between two samples of 5000 instances of rejection sampling was ≈ 0.04 for both S and I.

Digits sum. Using a pre-trained MNIST classifier, we define a multi-instance constraint that forces pairs of MNIST digits to sum up to ten. Given pairs of images $(\mathbf{x}, \mathbf{y})$, we define the constraint in the following way:

$$\bigvee_{i=1}^{9} \text{class}(\mathbf{x}, i) \wedge \text{class}(\mathbf{y}, 10 - i) \tag{3.20}$$

where $\text{class}(\mathbf{x}, i) := P\{\mathbf{x} \text{ is classified as } i\} = 1$, and P is obtained from a pre-trained classifier. The generated pairs of digits add up to ten in $\approx 96\%$ of cases[3]; however, only 2-8 and 4-6 pairs were generated, and $\approx 12\%$ of digits were not visually valid, though still mostly classified correctly relative to the sum constraint. This demonstrates the potential of this method when dealing with constraints involving multiple instances and pre-trained classifiers.

Restoration. Given a differentiable function $f(\cdot)$ that represents a corruption process in which information is lost, we define the following constraint:

$$\forall i \; f(\mathbf{x})_i = \tilde{\mathbf{y}}_i \tag{3.21}$$

where i is the pixel index and $\tilde{\mathbf{y}}$ is a corrupted sample, possibly such that there is a $\mathbf{y}$ that satisfies $\forall i \; \tilde{\mathbf{y}}_i \approx f(\mathbf{y})_i$. Such a constraint has the effect of sampling possible $\mathbf{x}$ such that $\forall i \; f(\mathbf{x})_i = \tilde{\mathbf{y}}_i$,

[3] according to classes assigned by the classifier

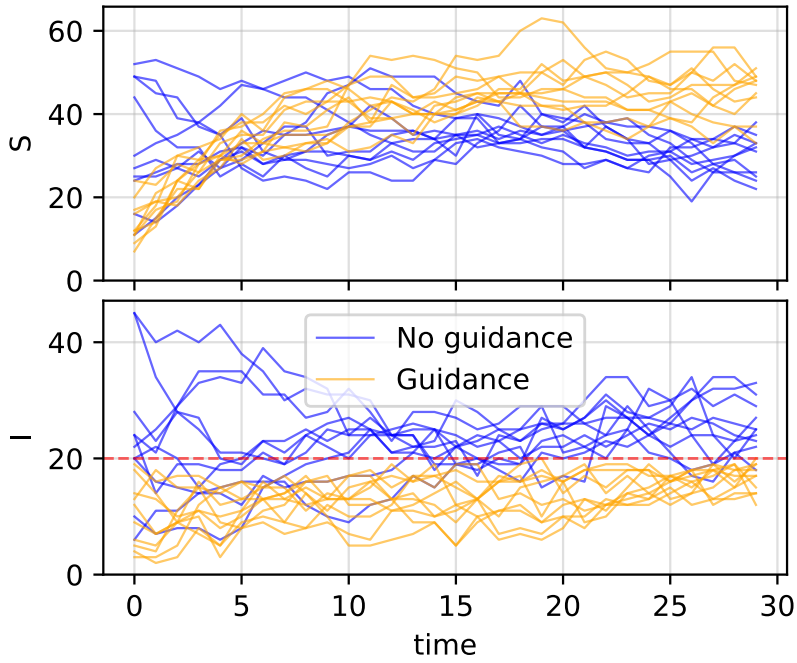


Figure 3.7: Imposing an inequality on eSIRS time series. We show here a subsample of the 100 time series generated with and without constraint guidance. In addition to the consistency constraints, which are always met, we imposed $\forall t\ I(t) \leq 20$, which is perfectly met in 99% of the samples.

i.e., "inverting" f or reconstructing the original $\mathbf{y}$. f can be any degradation process, such as downsampling, blurring or adding noise. So our approach can be used for image restoration for a known degradation process.

In Figure 3.8 we show the results of image restoration experiments with a blurred and a downsampled image. Samples are realistic and we report a low error with respect to the target corrupted image (the absolute error per channel is approximately less than 0.015).

MNIST: Symmetry. We impose vertical or horizontal symmetry constraints on generated images by imposing pixel-by-pixel equality between the image and its mirrored version. In Figure 3.9 we show resulting samples using a pre-trained model for MNIST images.

MNIST: relative filling. We construct constraints that force the generated image to have more white pixels on a certain part of the image with respect to the remaining. In order to do this, we impose the average pixel value of a given region to be larger than the average pixel value of the remaining pixels by a value of 1.0 (considering normalized images). We show in Figure 3.10 some samples generated imposing this constraint, in particular imposing the average pixel value of the upper half to be greater than the average of the lower part by one (or vice versa).

CelebA: color conditioning. With the objective of controlling the coloring of samples, we use a simple equality constraint between a target RGB color and the mean RGB color of the image. One can also think about more sophisticated constraints that aim at matching a target distribution in the color space. The resulting samples shown in Figure 3.11 demonstrate the possibility to manipulate the coloring of an image while preserving the quality of the samples.

Figure 3.8: Restoration experiments with CelebA. The first image on the left is a sample image from the CelebA dataset. Each row shows the corrupted image followed by samples generated imposing the restoration constraint.

Figure 3.9: MNIST samples generated imposing a horizontal symmetry constraint. In many cases ($\approx 30\%$) the generated image did not correspond to an actual digit; this is probably due to the unconditional model, which also generated a significant amount of invalid digits ($\approx 10\%$).

Nevertheless, it seems the color is excessively manipulated through background, hair, shades, or a "filter" effect. This is probably due to the naive approach of targeting the mean color.

CelebA: symmetry. The constraint we used in order to obtain images with vertical symmetry is analogous to the one we used for MNIST digits. Figure 3.12 shows how the symmetry soft constraint is successfully met in most samples, and how perfect symmetry is traded off with realism. As in most tasks involving images, we found it necessary to tune the constraint intensity k or λ, in order to regulate the trade-off between sample quality and constraint satisfaction.

In general, we observed the effectiveness of our method to depend on the task. For most experiments involving tabular data, keeping a large constraint strength k (such as $k = 30$) was sufficient. However, for some experiments, such as most tasks involving images, we needed to tune the constraint strength until a satisfactory trade-off between constraint satisfaction and sample quality was reached.

3.4.4 Comparison with universal guidance

We compared our training-free conditioning method with Universal Guidance, introduced in Bansal et al. (2023). Universal Guidance was successfully used in the context of image generation and it is arguably one of the state-of-the-art methods for training-free conditional generation. Their method is based on three improvements over the standard guidance technique (summing

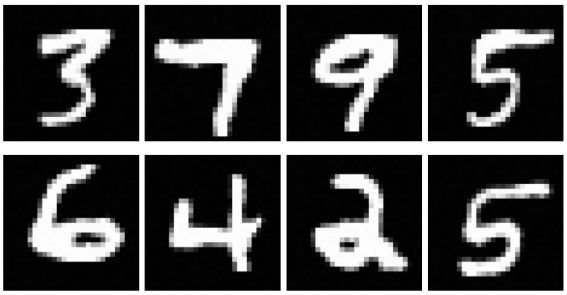


Figure 3.10: Relative filling experiments. The upper row of digits is generated imposing more white pixels on the upper half than the lower; the lower row is generated imposing more white pixels on the lower half.

Figure 3.11: Color conditioning on CelebA. The first row of images is generated imposing the mean color to be equal to a given dark target color, extracted from an image of the dataset. For the second row, instead, the target was a light color.

the constraint gradient to the score), that they call *forward universal guidance*, *backward universal guidance* and *per-step self-recurrence*:

- *Forward universal guidance* consists in using the gradient of the constraint with respect to the predicted clean data point $\hat{\mathbf{x}}_0$ (prediction based on the trained denoising net), instead of the current data point $\mathbf{x}_t$. The guidance term is thus computed as $\nabla_{\mathbf{x}_t} c(f_\theta(\mathbf{x}_t))$ where $\tilde{\mathbf{x}}_0 = f_\theta(\mathbf{x}_t)$ is the clean data point predicted by the denoising network f_θ.

- *Backward universal guidance* involves an optimization of $\hat{\mathbf{x}}_0$ according to the constraint and modifying the score used for the reverse process as a consequence. First, the predicted clean data point $\hat{\mathbf{x}}_0 = f_\theta(\mathbf{x}_t)$ is optimized with respect to the constraint through gradient ascent. Then, the optimized $\hat{\mathbf{x}}_0$ is plugged in the diffusion posterior $p(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \hat{\mathbf{x}}_0)$ to sample the following step $\mathbf{x}_{t-1}$ (or equivalently, to obtain the modified score).

- *Per-Step Self Recurrence* consists in "going back" after applying the backward diffusion step by sampling $\mathbf{x}_t \sim p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$, which corresponds to re-injecting Gaussian noise. This is done multiple times before the actual backward step, intuitively in order to allow more computational budget for finding good solutions. It has been observed to improve the quality of samples in image generation tasks.

Figure 3.13 shows a schematic representation of the three techniques.

Applying (to the best of our implementation) the three universal guidance techniques in some of the experiments described above, we observed the following:

Figure 3.12: CelebA samples generated imposing a vertical symmetry constraint. Notice the bias for frontal face position and uniform background. While the constraint was useful to improve the symmetry of images, reaching a perfect symmetry by increasing the constraint intensity was not possible without compromising the samples' quality.

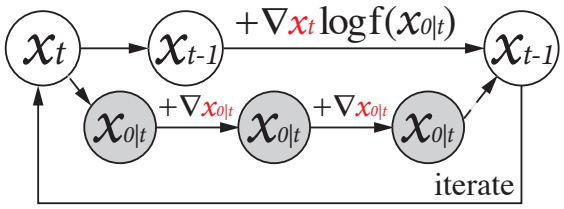


Figure 3.13: Universal Guidance Sampling Algorithm. Here $\log f(\mathbf{x}_{0|t})$ represents the constraint evaluated on the predicted clean data point $\mathbf{x}_{0|t}$. Image from Ye et al. (2024).

- On tabular data and time series, the effect of forward universal guidance was mostly negligible, except for models trained with sliced score matching, where it led to failure. Thus, for our experiments with universal guidance we used only models trained with denoising score matching.
- Backward universal guidance and per-step self-recurrence instead led to a significant worsening of the performances in tabular and time series data. We show in Table 3.5 and Table 3.6 statistics about the comparison. Figure 3.14 shows how using backward guidance introduces strong biases in some marginals in the white wine experiment. We think this may be due to the optimization process involved, which focuses on optimizing samples but ignores the noise, thus failing to converge to the true conditional distribution. We also verified the effect of Langevin dynamics correction in these experiments, and we observed a clear reduction of the bias introduced by the application of backward universal guidance, compared with the same configuration without Langevin correction steps.
- In MNIST experiments, the application of forward guidance led to an improvement in the digits' sum task, as we observed all digits between 1 and 9 in the generated samples (instead of only 2-8, 4-6 pairs generated with our method) and a higher quality (see Figure 3.16 and Figure 3.15). In image symmetry experiments, instead, the generated images were mostly ones ($\approx$ 87.5% with $\approx$ 15.6% of invalid digits). On the relative filling experiment we obtained similar results.
- In CelebA experiments, we did not notice relevant differences when applying forward universal guidance. We did not conduct further experiments with the other two techniques.

In summary, applying these techniques on tabular data and time series, we observed little effect of *forward universal guidance*, and a significantly detrimental effect of *backward universal*

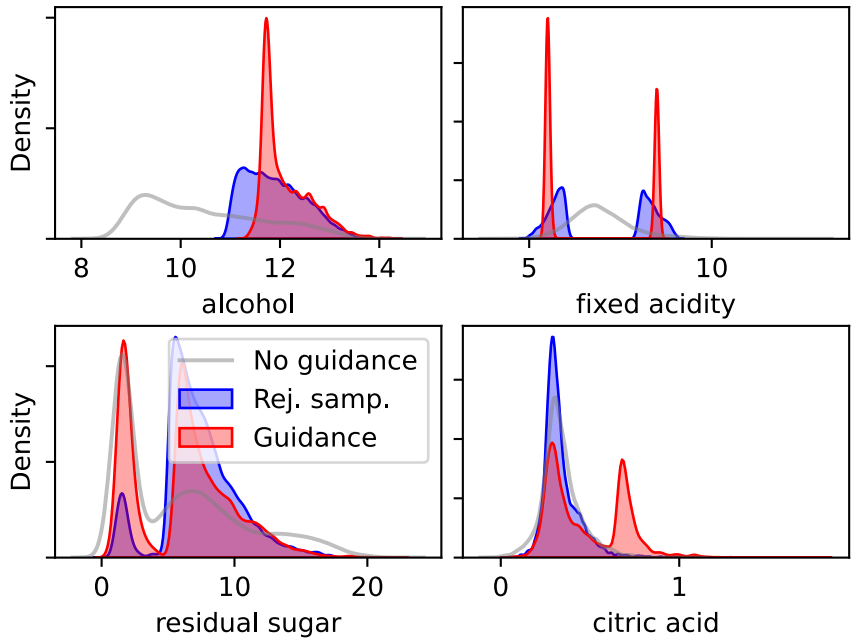


Figure 3.14: Effect of *backward universal guidance* on white wine experiment. We generated 5000 samples using forward universal guidance and backward universal guidance (without Langevin dynamics correction) and as many by RS. The plot compares the marginals of the dimensions directly involved in the constraint. We can see that the distribution of generated samples is highly biased. In particular, it seems that samples are distributed more densely around the modes.

Figure 3.15: Digits sum experiment. The columns are the pairs of generated digits. Using our method we were only able to generate 2-8 and 6-4 pairs. The classes assigned by the classifier added up to ten in 96% of cases, but visual inspection reveals $\approx 12\%$ of invalid digits.

guidance. Metrics reported in Table 3.7 summarize the comparison of our method for two of the previously discussed tasks. We think this significant difference is due to the fact that Universal Guidance, as most recent plug-and-play techniques, modifies the sampling algorithm of the original unconditional diffusion model by introducing an optimization phase. In this way the resulting sampling algorithm is not guaranteed to sample from the true conditional distribution.

On the other hand, we noticed a sensible improvement in some of our image-based conditional generation tasks. For example, with respect to the MNIST digits' sum experiment, universal guidance allowed us to generate all possible pairs that always summed up to ten. We infer that universal guidance can be useful when the objective is to generate high-quality samples that satisfy a given constraint, but it can lead to large biases when modelling the conditional distribution, which is of primary interest in some contexts, as with tabular data and time series. Moreover, with these experiments we verified that our *Log-probabilistic logic* is effective also within other guidance techniques.

Table 3.5: Applying universal guidance to white wine experiment. We compare here the application of universal guidance techniques (FW: forward, BW: backward and SR: per-step self-recurrence) to our method for the white wine constrained generation task discussed in Section 3.4.1. The results previously discussed above were obtained with a model trained with sliced score matching (SSM), but in order to apply universal guidance, we also trained a model with denoising score matching (DSM) and reduced the strength of the constraint (from $k = 50$ to $k = 30$). We also show the effect of applying Langevin dynamics (LD) correction steps. We found forward universal guidance to have little effect on the results. Instead, using backward universal guidance significantly increased the error in terms of histogram distance (TVD) and also in terms of average L1 distance of correlation coefficients. Average, median and maximum are computed by aggregating the errors over the marginals of all the 11 dimensions.

Method	**Avg TVD**	**Median TVD**	**Max TVD**	**Avg L1 corr.**
SSM + LD	0.1	0.1	0.153	0.07
DSM	0.1	0.081	0.205	0.059
DSM + FW	0.123	0.112	0.262	0.059
DSM + FW + BW	0.238	0.13	0.827	0.093
DSM + FW + BW + SR	0.288	0.162	0.850	0.094
DSM + LD	0.132	0.13	0.221	0.084
DSM + FW + LD	0.127	0.114	0.232	0.082
DSM + FW + BW + LD	0.141	0.117	0.3	0.088
DSM + FW + BW + SR + LD	0.167	0.124	0.377	0.099

3.5 Limitations

Despite the promising results, we observed concrete limitations of our approach. Our method degrades as constraint complexity increases, particularly when constraints involve many variables or impose sharp boundaries. In high-dimensional settings such as images, we often observe underperformance and a poorer trade-off between sample quality and constraint satisfaction. In these regimes, alternative methods tailored to images or based on optimization may be preferable. Moreover, the constraint strength typically requires tuning to balance fidelity and satisfaction, especially in difficult tasks. Our empirical evaluation is limited to a modest number of tabular datasets, so broader validation would be beneficial. Finally, we did not test our approach with few-step samplers such as flow matching, as the samplers we used instead often needed at least 1000 steps to be effective.

3.6 Discussion

We have shown how we can exploit pre-trained unconditional score-based generative models to sample under user-defined logical constraints, without the need for additional training. Our experiments demonstrate the effectiveness in several contexts, such as tabular and high-dimensional data. Nevertheless, in some high-dimensional settings such as images, we had to trade off between sample quality and constraint satisfaction by tuning the constraint strength. More sophisticated methods specifically designed for images are probably necessary; still, these could fail to model the true conditional distribution. In fact, we show the superiority of our method in approximating conditional distributions for tabular and time-series data with respect

Table 3.6: Applying universal guidance to eSIRS bridging experiment. We compare here the application of universal guidance techniques to our method for the eSIRS bridging task discussed in Figure 3.6. We found forward universal guidance to have a small but positive effect on the results; instead, using backward universal guidance increased the error, especially when Langevin dynamics correction was not applied. Average, median and maximum are computed by aggregating the errors over the marginals of all dimensions $(S(t), I(t))$.

Method	**Average TVD**	**Median TVD**	**Max TVD**
DSM	0.152	0.142	0.371
DSM + FW	0.113	0.102	0.221
DSM + FW + BW	0.197	0.153	1.0
DSM + FW + BW + SR	0.467	0.485	1.0
DSM + LD	0.129	0.114	0.252
DSM + FW + LD	0.107	0.094	0.208
DSM + FW + BW + LD	0.126	0.116	0.288
DSM + FW + BW + SR + LD	0.399	0.342	0.77

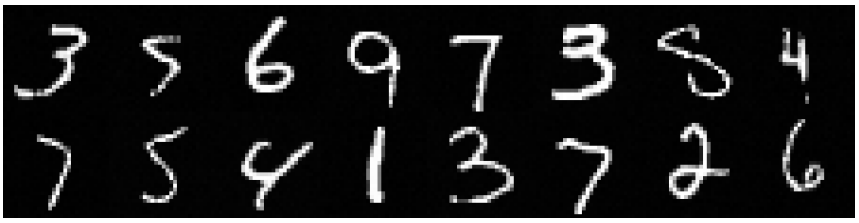


Figure 3.16: Digits sum experiment with universal guidance. The columns are the pairs of generated digits. Using forward universal guidance we were able to generate all possible pairs that summed up to ten. The classes assigned by the classifier added up to ten in 100% of cases, but visual inspection reveals $\approx 8\%$ of invalid digits. Using also backward universal guidance and per-step self-recurrence led to very similar results.

to a state-of-the-art method for training-free conditioning. Future work will aim at finding better approximation schemes, starting from works focused on general plug-and-play guidance for images.

Table 3.7: Comparison with Universal Guidance. On the white wine and eSIRS bridging tasks our method performs significantly better in terms of TVD (we report average and maximum distance over the marginals).

Method	White wine		eSIRS bridging	
	Avg TVD	**Max TVD**	**Avg TVD**	**Max TVD**
Ours	**0.1**	**0.15**	**0.13**	**0.25**
Univ. Guidance	0.29	0.85	0.47	1.0

Chapter 4

Graph-Conditional Flow Matching for Relational Data Generation

Summary: This chapter addresses the generation of relational databases, where multiple tables are interconnected through foreign-key relationships. We propose a graph-conditional flow-matching approach that generates the entire database content given its relational structure. Whereas existing methods generate tables sequentially or assume independence between records, our method models the entire relational dataset jointly by leveraging a flow-matching framework conditioned on the relational graph, and a GNN-based denoiser that propagates information across related records. The method handles complex schemas, including tables with multiple parents and multiple foreign-key types between tables. Experiments on six real-world datasets using the SyntheRela benchmark demonstrate state-of-the-art fidelity of the generated data.

This chapter is based on the original work *Graph-Conditional Flow Matching for Relational Data Generation* (Scassola et al., 2025a) (accepted at AAAI 2026).

4.1 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 (Shi et al., 2025, Xu et al., 2019) has been subject to research for several years. Early methods were based on Bayesian networks (Zhang et al., 2017), factor graphs (McKenna et al., 2019) and autoregressive models (Nowok et al., 2016). Most recent methods leverage the breakthroughs of deep-learning based generative models, from early latent variable models such as VAEs (Kingma and Welling, 2014) and GANs (Goodfellow et al., 2014), to transformer-based autoregressive models (Vaswani et al., 2017) and diffusion models (Ho et al., 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 (Gueye et al., 2022, Pang et al., 2024, Patki et al., 2016, Solatorio and Dupriez, 2023) is a less mature field, with few existing methods capable of properly handling complex database structures.

In this chapter, 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) (Battaglia et al., 2018, Scarselli et al., 2009). 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 et al., 2024), a recently developed benchmarking library, we prove the effectiveness of our method on several datasets, comparing it with several open-source approaches. Experimental results show our method achieves state-of-the-art performance in terms of fidelity of the generated data.

4.2 Relational Data Generation

Tabular data is often found in the context of relational databases, where multiple related tables are stored together. Tables in relational databases may include one or more columns containing foreign keys, allowing records to refer to records in other tables. This is done with the purpose of reducing redundancy and improving data integrity.

The problem of generating synthetic relational data, or multi-table generation, is an emerging research area that aims to generate realistic synthetic data while preserving the relationships between tables. This task is more complex than single-table generation, as it requires modeling not only the distributions of individual tables, but also the dependencies and constraints imposed by foreign keys.

A record in a table can be connected to records in one or more different tables through foreign keys, i.e., primary keys of records in other tables. Commonly, it is assumed that each column containing foreign keys refers to a specific table, and foreign keys cannot refer to records in the same table. Consequently, we propose to represent a relational database 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).

More formally, we can define a relational database $\mathbb{D}$ as a pair $\mathbb{D} := (\mathbb{X}, \mathbb{G})$ where $\mathbb{X}$ is a set of K tables $\mathbb{X} := \{\mathbb{T}_k\}_{k=1}^{K}$, where each table $\mathbb{T}_k$ is a collection of records (rows) $\mathbf{x}_i^k$ sharing the same structure (columns), and $\mathbb{G}$ is a foreign-key graph defining how records are connected through

foreign keys. We refer to $\mathbb{X}$ as the *content* or the *features* of the relational database, and we refer to $\mathbb{G}$ as the *foreign-key graph*, *topology* or *structure* of the relational database. 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. We show in Figure 4.1 an example of a schema of a relational database and in Figure 4.2 an example of tables following the schema.

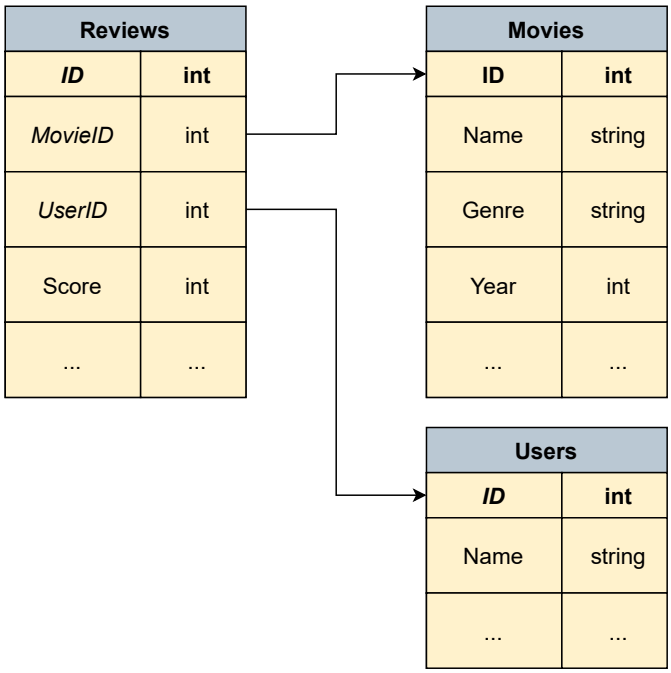


Figure 4.1: Example of relational database schema and tables. In this example tables *Movies* and *Users* are both parents of the table *Reviews*. Notice the emerging graph describing the foreign-key relationships in the schema of the database.

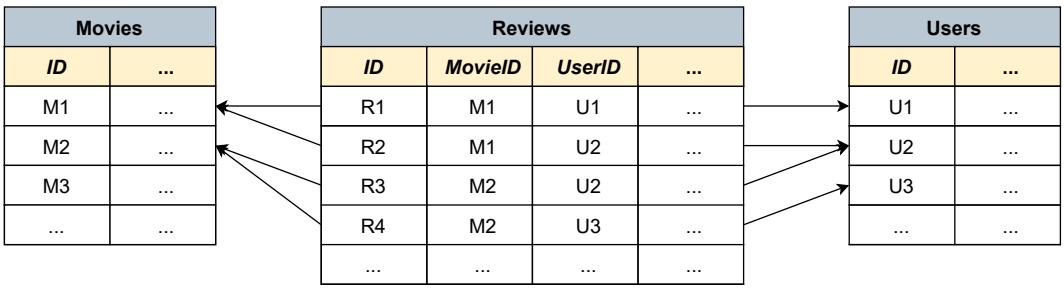


Figure 4.2: Example of tables in a relational database. The tables follow the schema described in Figure 4.1, highlighting the foreign key relationships. The emerging foreign-key graph (in this case, relative to the records, not to the schema) is shown more explicitly in Figure 4.4a.

Therefore, relational data generation can be seen as a particular type of graph generation problem. The additional complexities in relational data generation compared to single-table generation are twofold. First, the generated relational dataset should have a foreign-key graph $\mathbb{G}$ that has statistical properties similar to those of the original one. Second, the dependencies between records induced by foreign keys should be preserved in the synthetic data.

The difficulty of the task depends highly on the structure of the foreign-key graph. In the simplest case, there are only *one-to-one* relationships between tables, i.e., each record in a child table refers to at most one record in a parent table, and records in a parent table are referred to by at most one record from the same child table (but they can still be referred to by many children records from distinct tables). In this case, the relational database can be flattened (de-normalized) into a single table by merging child tables into their parent tables (join operation), so the problem is reduced to a single-table generation task.

A more complex case is given by *one-to-many* relationships, where a record in a parent table can be referred to by multiple records in a child table, but each record in a child table still refers to at most one record in a parent table. Figure 4.3 shows an example of one-to-many relationships between tables. This corresponds to a tree-like foreign-key graph, where each parent node can have multiple child nodes, but each child node has only one parent. In this case, flattening is not possible, at least in a traditional way. Many methods handle this case by generating parent tables first, and then generating child tables conditioned on the generated parent records, traversing the "tree" of tables emerging from the foreign-key relationships.

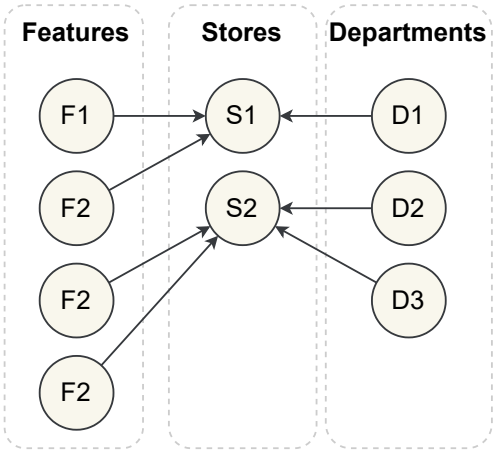


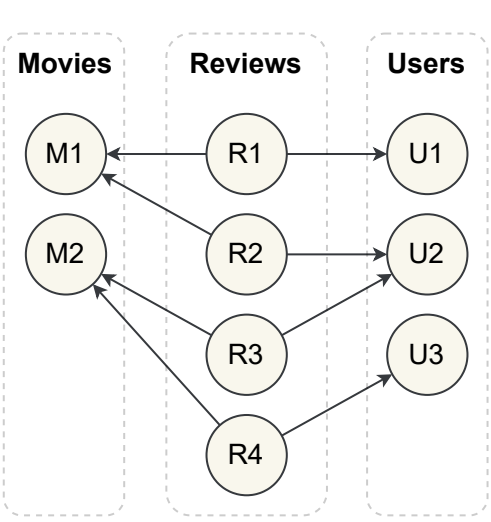
Figure 4.3: Example of foreign-key graph characterized by *one-to-many* relationships. The table *Stores* is the parent table of the other two, and arrows represent foreign-key relationships from child to parent tables. Indeed, records in children tables refer to only one parent record in the *Stores* table. When inverting the direction of the arrows (from parent to children), the resulting graph is a tree. This example is taken from a database of stores (Kaggle, 2014) that will be used later.

In order to generate the graph structure $\mathbb{G}$, it is possible to model the number of children per parent record, conditioning on the ancestor record features.

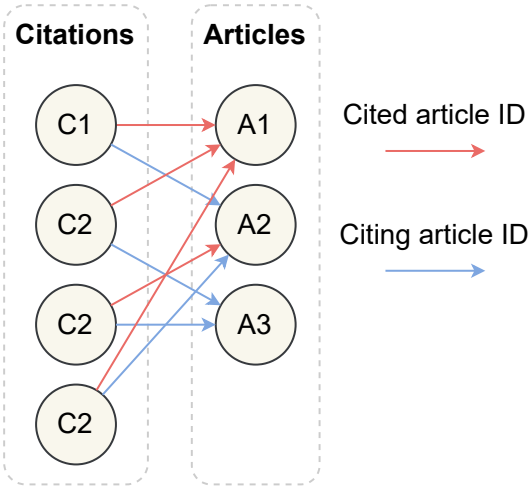
Many of the proposed methods for relational data generation follow this approach. The Synthetic Data Vault (SDV) (Patki et al., 2016) pioneered multi-table synthesis via the Hierarchical Modeling Algorithm (HMA), which employs Gaussian copulas and recursive conditional parameter aggregation to propagate child-table statistics into parents. GAN-based relational extensions include Gueye et al. (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.

Indeed, the most complex case is given by *many-to-many* relationships, where a record in a child table can refer to multiple records in parent tables. Moreover, it is also possible that a child table has two different columns having foreign keys referring to the same parent table. For example, a child table "Students" may have two columns "mother ID" and "father ID" referring to the same parent table "Parents". In other words, each record in the table "Students" is linked to two records in table "Parents". When modelling this kind of graph, one can use different edge labels to distinguish between the two different foreign-key relationships among the same two tables. In practice, each edge is labelled with the name of the foreign key column it represents. Figure 4.4 contains examples of many-to-many relationships between tables.



(a) Table with multiple parent tables.



(b) Table with multiple parent relationships toward the same table.

Figure 4.4: Examples of foreign-key graphs with *many-to-many* relationships. The arrows represent foreign-key relationships from child to parent tables. The graph from (b) is taken from the schema shown in Figure 4.2 (inspired from the dataset *MovieLens* (Harper and Konstan, 2015) that we will use later), illustrating a many-to-many relationship where a child table has multiple parent tables. In (c) we show a many-to-many relationship where a child table has multiple parents from the same table (inspired from the dataset *CORA* (McCallum et al., 2000) that we will use later). In this case the table *Citations* has two different types of foreign-key relationships towards the parent table. Notice how in both cases, when inverting the direction of the arrows (from parent to children), the resulting graphs are no longer trees, so it is not possible to generate tables and records sequentially in a tree-like fashion.

In this case, when a child table has multiple parents it is not clear how to sample the foreign keys connecting the parents, since one cannot simply sample the number of children per parent of a single table. In other words, generating children records from parents recursively can only generate tree structures, but not arbitrary DAGs. One of the first works explicitly dealing with this was (Pang et al., 2024), where to handle tables with multiple parents, they generate a version of the child table for each parent. These are then heuristically merged, by finding similar rows and selecting only one version with the union of the foreign keys. The model is based on diffusion, and the way information is propagated from parents to children is based on latent embeddings and guidance.

More recently, the predominant approach has been to generate the entire foreign-key graph $\mathbb{G}$ independently from the content $\mathbb{X}$, in order to guarantee more flexibility and expressiveness, and to generate the content of the records only once the structure is fully determined. This also came after an increasing awareness of the "graph" nature of relational data. This approach was introduced in Xu et al. (2022) where the graph is first generated using a statistical method that aims at preserving the degree distributions of bipartite graphs describing the connections between two tables (Boroojeni et al., 2017). Then, the tables are generated in order. When a table is generated, each record is generated conditioning on connected records through aggregated information about connected records and topological information (e.g., the node degree).

Despite the increasing complexity and expressiveness of these methods, they all show a common limitation: all records in the same table are generated independently from each other, conditioning at best only on connected records in other tables. This represents a limit when children records of the same parent are strongly correlated given the parent record. For instance, given again the example of students and parents, the birth dates of siblings are often correlated, even after conditioning on the parents' information (for example, the birth dates of siblings cannot be too close, unless they are twins). Concurrent work by Hudovernik (2024) also follows this approach, generating the content of the tables using a latent diffusion model conditioned on node embeddings produced by a separate model.

One possible approach is to apply more general deep generative models for graphs, which do not assume any independence between nodes in the same table. However, these methods struggle to scale to large graphs, since they usually need to instantiate the entire adjacency matrix, which scales quadratically with the number of nodes (Zhu et al., 2022). Moreover, a simple application of these methods would not exploit the particular structure of relational databases, which could be useful to improve generation quality and scalability. For example, graphs emerging from relational databases are usually multipartite and sparse. The number of edges is usually linear in the number of nodes, since each record usually has a fixed number of foreign keys referring to other records in a specific table.

4.3 Method

4.3.1 Graph-Conditional Generation

As introduced above, a relational database $\mathbb{D} = (\mathbb{X}, \mathbb{G})$ can be represented as a graph, where nodes correspond to records and edges are defined by foreign-key relationships. Let $\mathbf{x}^i$ denote the features of node i, and by g^i the set of nodes to which i is connected. Thus, we define $\mathbb{X} := \{\mathbf{x}^i\}_{i=1}^N$ and $\mathbb{G} := \{g^i\}_{i=1}^N$ where N is the total number of records/nodes in the relational database. Since records 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:

$$p(\mathbb{X}, \mathbb{G}) = p(\mathbb{G}) \prod_{k=1}^{K} p(\mathbb{T}_k \mid \mathbb{T}_{1:k-1}, \mathbb{G}) \tag{4.1}$$

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:

$$p(\mathbb{X}, \mathbb{G}) = p(\mathbb{G})p(\mathbb{X} \mid \mathbb{G}) \tag{4.2}$$

In this chapter 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 varied 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, when the topology is given by the user or when a model for structures is trivial to obtain.

4.3.2 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 chapter 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.

4.3.3 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, 2015), 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 $\mathbb{D} = (\mathbb{X}, \mathbb{G})$ from a high-dimensional joint distribution.

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 $p_t(\mathbb{X}_t \mid \mathbb{X}_1, \mathbb{G})$. We use the optimal transport conditional flow described in Section 2.2.4 as conditional flow $p_t(\mathbf{x}_t^i \mid \mathbf{x}_1^i)$, independent for each node $\mathbf{x}_t^i \in \mathbb{X}$ and for each component of a node (for each field of each record). We refer to the relative conditional velocity as $u_t(\mathbb{X}_t \mid \mathbb{X}_1)$. This also holds for categorical components of features $\mathbf{x}^i$, 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 $\mathbb{G}$, so we can refer to it as $p_t(\mathbb{X}_t \mid \mathbb{X}_1)$. The objective is to learn the marginal velocity of the whole relational dataset $v_t(\mathbb{X}_t \mid \mathbb{G})$. We learn this using the variational parametrization discussed above:

$$p_\theta(\mathbb{X}_1 \mid \mathbb{X}_t, \mathbb{G}) \approx q(\mathbb{X}_1 \mid \mathbb{X}_t, \mathbb{G}) \tag{4.3}$$

Thus, the training loss is the following:

$$\mathcal{L}(\theta) = \mathbb{E}_{t\sim U(0,1), \mathbb{X}_t \sim p_t(\mathbb{X}_t \mid \mathbb{X}_1)} \left[-\log p_\theta(\mathbb{X}_1 \mid \mathbb{X}_t, \mathbb{G}) \right] \tag{4.4}$$

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:

$$v_t^\theta(\mathbb{X}_t, \mathbb{G}) = \mathbb{E}_{\mathbb{X}_1 \sim p_\theta(\mathbb{X}_1 \mid \mathbb{X}_t, \mathbb{G})} \left[u_t(\mathbb{X}_t \mid \mathbb{X}_1, \mathbb{G}) \right] \tag{4.5}$$

Variational Parametrization

Let $x^{k,d,j}$ be the j-th value of column d of table k, then we can write

$$\mathbb{X}_t = \left\{ x^{k,d,j} \;\middle|\; k = 1, \dots, K;\ d = 1, \dots, D_k;\ j = 1, \dots, N_k \right\} \tag{4.6}$$

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 $x^{k,d,j}$. Therefore we can write the variational approximation as

$$p_\theta(\mathbb{X}_1 \mid \mathbb{X}_t, \mathbb{G}) = \prod_{k=1}^{K} \prod_{d=1}^{D_k} \prod_{j=1}^{N_k} p_\theta^{k,d}(x_1^{k,d,j} \mid \mathbb{X}_t, \mathbb{G}) \tag{4.7}$$

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

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

$$p_\theta^{k,d}(x_1^{k,d,j} \mid \mathbb{X}_t, \mathbb{G}) = \text{Categorical}\left(x_1^{k,d,j} \mid \mathbf{p} = f_{\theta_2}^{k,d}(x_t^{k,d,j}, t, \boldsymbol{\varepsilon}_t^i)\right) \tag{4.8}$$

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:

$$p_\theta(x_1^{k,d,j} \mid \mathbb{X}_t, \mathbb{G}) = \mathcal{N}(x_1^{k,d,j} \mid \mu = f_{\theta_2}^{k,d}(x_t^{k,d,j}, t, \boldsymbol{\varepsilon}_t^i), \sigma = 1) \tag{4.9}$$

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). This worked well in practice, but learning the variance or using more sophisticated functions σ_t could be an interesting direction for future work.

Velocity Computation. The velocity for each component $x^{k,d,j}$ at time t follows from Equations 2.35 and 4.5:

$$v_t^\theta(x^{k,d,j} \mid \mathbb{X}_t, \mathbb{G}) = \frac{\mathbb{E}_{x_1^{k,d,j} \sim p_\theta^{k,d}(x_1^{k,d,j} \mid \mathbb{X}_t, \mathbb{G})}[x_1^{k,d,j}] - (1 - \sigma_{\min})x^{k,d,j}}{1 - (1 - \sigma_{\min})t} \tag{4.10}$$

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

$$v_t^\theta(x^{k,d,j} \mid \mathbb{X}_t, \mathbb{G}) = \frac{f_{\theta_2}^{k,d}(x_t^{k,d,j}, t, \boldsymbol{\varepsilon}_t^i) - (1 - \sigma_{\min})x^{k,d,j}}{1 - (1 - \sigma_{\min})t} \tag{4.11}$$

Recall that $x^{k,d,j}$ can be either a continuous variable or a one-hot-encoded categorical variable. In the 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 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 $f_{\theta_2}^{k,d}$: the features $x^{k,d,j}$, the noise level t and the node embeddings $\boldsymbol{\varepsilon}_t^i$, are concatenated and flattened in order to be fed to the MLP.

Instead, the neural network η_{θ_1} computing node embeddings $\boldsymbol{\varepsilon}_t^i = \eta_{\theta_1}(\mathbb{G}, \mathbb{X}_t)^i$, 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 two architectures, one based on GIN (Xu et al., 2018) and one based on GATv2 (Brody et al., 2021). In order to build a GNN compatible with heterogeneous graphs, we take an existing GNN model and use a dedicated

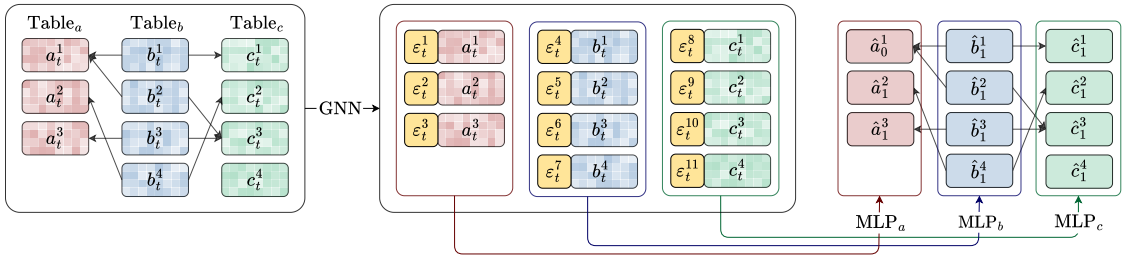


Figure 4.5: Overview of the architecture of the denoiser for relational data. A relational dataset composed of multiple tables can be seen as a graph, where records are the nodes and foreign keys are the edges. The denoiser takes as input a relational dataset where noise was added to each record with 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 ($t = 1$).

GNN layer for each edge type, as shown 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. Using edge-specific layers allows to seamelessly handle the case of multiple foreign-key relationships between the same two tables, which can be represented as multiple edge types between the same two node types. A detailed description of the employed GNN architectures is provided in Appendix B.2. Figure 4.5 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 normalization, so that all marginals of continuous features will be normally distributed. In order to handle missing data in numerical columns, we augment the tables by including an auxiliary column containing binary variables indicating if the data is missing, and we fill missing values with the mean. This allows us 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 perform early stopping and 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 implies all records could be denoised in parallel. In principle, one can also use batches corresponding to connected components of the graph. 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 (see Appendix B.1 for more details), for the other datasets just a few minutes. We summarize in Algorithm 4 the graph-conditional training algorithm[1].

Algorithm 4 Graph-Conditional Flow Matching Training

Input: $\mathbb{D} = (\mathbb{X}, \mathbb{G})$
Parameter: Number of iterations n
Output: Model parameters θ

1: **for** $i = 0$ **to** n **do**
2: Sample connected component $\mathbb{D} = (\mathbb{X}_1, \mathbb{G})$ from training set
3: Sample $t \sim U(0, 1)$
4: Sample $\mathbb{X}_t \sim p_t(\mathbb{X}_t \mid \mathbb{X}_1)$
5: Compute loss $\mathcal{L}(\theta) = -\log p_\theta(\mathbb{X}_1 \mid \mathbb{X}_t, \mathbb{G})$ Eq. (4.7)
6: Update parameters θ using $\nabla_\theta \mathcal{L}(\theta)$
7: **return** θ

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 5 the graph-conditional generation algorithm.

Algorithm 5 Graph-Conditional Flow Matching Generation

Input: $\mathbb{G} \sim p(\mathbb{G})$
Parameter: Number of steps T
Output: Samples $(\mathbb{X}_1, \mathbb{G})$

1: Sample $\mathbb{X}_0 \sim \mathcal{N}(0, I)$
2: **for** $t = 0$ **to** 1 **step** $\frac{1}{T}$ **do**
3: $x_{t+\frac{1}{T}}^{k,d,j} = x_t^{k,d,j} + \frac{1}{T} v_t^\theta(x_t^{k,d,j} \mid \mathbb{X}_t, \mathbb{G})$ Eq. (4.11)
4: **return** $(\mathbb{X}_1, \mathbb{G})$

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

[1] In practice, we used a fixed set of equally spaced values for the noise level t, instead of sampling it from a uniform distribution.

layers and 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. In Appendix B.2 we discuss how the validation loss changes as a function of this hyperparameter for two of the datasets.

4.4 Experiments

4.4.1 Experimental Settings

We evaluate our method using SyntheRela (Hudovernik et al., 2024) (Synthetic Relational Data Generation Benchmark), 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, 2015), a commonly used dataset to study graph properties, containing users' ratings of different movies. The last three datasets have tables with multiple parents. AirBnB, Walmart and Rossmann were subsampled in order to enable comparison with other methods. More details are provided in Appendix B.3.

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 the case 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 from 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 the SyntheRela library, which are the leading open-source approaches for relational data generation. In this study, we prefer discriminator-based metrics over utility metrics. First, selecting a target is often an arbitrary choice, and the datasets we used have no clear targets for classification or regression tasks. Second, depending on the predictive model used, more realistic data can result in inferior utility. Moreover, in the SyntheRela benchmark library, the computation of the MLE metric is only provided for three of the simplest datasets and requires dataset-specific feature extraction. Although we are interested in evaluating our method on utility metrics, we leave this for future work, as it would require a more standardized and comprehensive evaluation framework.

4.4.2 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 4.1 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 et al., 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.

Table 4.1: 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.

	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

Privacy Evaluation. We evaluated potential privacy leaks in each table, where parent tables were enriched with aggregated information as previously described. In order to do this, we use a DCR-based privacy evaluation methodology (Boudewijn et al., 2023, Palacios et al., 2025, Park et al., 2018, Platzer and Reutterer, 2021, Steier et al., 2025). In particular, we follow the methodology previously described in Section 2.5. 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). As shown in Table 4.2, 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.

Table 4.2: 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.

Dataset	Table	Records	Features	$p_{\leq 2\%}$	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

4.5 Limitations

Foreign-key Graph Generation. As this method permits the generation of the content of a relational dataset, but not the foreign-key graph, it has to be combined with a graph generation algorithm in order to properly generate novel relational data. However, we think separating the two problems is a promising approach. For example, in Xu et al. (2022) they use a statistical method to generate the graph. We follow the same simple approach of Hudovernik (2024) for sampling the graphs, that is resampling the original connected components. This could potentially raise a privacy issue related to the leaked structure, which is, however, outside the scope of this chapter, as we aim at building an effective generative model conditioning on a given foreign-key graph. Nevertheless, we did not observe any privacy leaks in the analysis of parent tables enriched with aggregated information. Moreover, structures are often over-represented in training data.

Scaling. Our method requires a GNN to process whole connected components. This is potentially problematic when these are very large. There are many approaches to deal with this such as advanced batching strategies, graph partitioning, or out-of-core processing. Nevertheless, in our case it was never an issue, as we were able to fit the whole datasets into memory. So the computational complexity of our method scales linearly with the size of the largest connected component. In theory, oversmoothing could also be an issue when very deep GNNs are used (Li et al., 2018, Oono and Suzuki, 2020). However, in our experiments we did not need very deep GNNs (between 2 and 3 message passing layers). We hypothesize that the iterative denoising process itself helps propagate information across distant nodes.

Hyperparameters. Our method is relatively sensitive to the size of the embedding produced by the GNN, which has to be tuned for each dataset in order to achieve a good trade-off between

expressiveness and overfitting. 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.

4.6 Discussion

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.

Concurrent Work. Concurrent and independent work by Hudovernik (2024) shares 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.

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.

As previously mentioned, we believe further engineering the denoiser architecture can lead to additional performance improvements, as we used relatively simple neural network architectures. Another interesting theme is the development of better discriminators for relational data, potentially based on GNNs, that could provide a more accurate evaluation of synthetic data fidelity. The current discriminators are based on hand-crafted features and may not fully capture the complexities of relational data. Moreover, these could be used to include a discriminator loss during training, encouraging the model to generate more realistic data in fewer steps (Chadebec et al., 2025). We also consider it interesting to use more sophisticated parametrizations of the variational distributions, for example with probabilistic circuits.

As our method is based on flow matching, one can further exploit properties of diffusion-like models, such as guidance, inpainting, or the generation of variations of a given dataset.

Finally, one could explore the integration of neuro-symbolic relational constraints into the training process, to enforce logical consistency in the generated data. This could be done by feeding

the neuro-symbolic features to a discriminator-based loss, by including a regularization term in the loss function or by using special architectures (Stoian et al., 2024). Since it is possible to obtain a score-based formulation from a flow-matching model (Eijkelboom et al., 2024), one could also explore the use of the technique we developed in Chapter 3.

Chapter 5

Tabular Data Generation with Probabilistic Circuits

Summary: This chapter proposes probabilistic circuits as competitive, efficient, and tractable generative models for tabular data. We benchmark overparameterized PCs against state-of-the-art deep generative models, particularly diffusion-based approaches, using established datasets and evaluation protocols. Our empirical results show that PCs achieve comparable performance while offering faster training and sampling. We also critically examine commonly used evaluation metrics, identifying limitations such as saturation and limited informativeness, and propose more principled alternatives for assessing generative models for tabular data.

This chapter is based on original work in collaboration with the *april Lab*.

5.1 Introduction

Tabular data is ubiquitous across scientific and industrial domains, forming the backbone of real-world information systems across healthcare, finance, public administration, and scientific domains (Assefa et al., 2021, Fonseca and Bacao, 2023, Giuffrè and Shung, 2023, Gonzales et al., 2023, Hernandez et al., 2022). The ability to generate high-fidelity synthetic tabular data has therefore become increasingly important, in particular for privacy-preserving data sharing and data augmentation under limited samples. Despite the typically moderate dimensionality of tables, tabular data generation is challenging: features are heterogeneous (categorical and numerical), marginals are often multimodal or heavy-tailed, dependencies are non-linear, and data may have missing values.

In the last few years, several deep generative models have been proposed for tabular data generation, including GAN-based models (Xu and Veeramachaneni, 2018, Xu et al., 2019), VAE-based models (Liu et al., 2023a), language model-based approaches (Borisov et al., 2023) and diffusion-based models (Guzmán-Cordero et al., 2025, Jolicoeur-Martineau et al., 2024, Kotelnikov et al., 2023, Shi et al., 2025, Zhang et al., 2024), which currently dominate state-of-the-art benchmarks. However, these complex deep generative models tend to be computationally expensive, and only allow for unconditional sampling. Moreover, a significant portion of the recent literature relies

on metrics that, upon closer inspection, suffer from several fundamental issues: (i) they are inflated, overstating the apparent quality of generated samples; (ii) they are saturated, with many disparate models achieving essentially indistinguishable high scores; and (iii) they are often uninformative, failing to meaningfully describe real capabilities. Nevertheless, these metrics are becoming standard for reporting progress in tabular data generation.

In this chapter, we propose probabilistic circuits (PCs) (Vergari et al., 2021) as a competitive, efficient, and tractable alternative for tabular data generation. This is a natural choice as hierarchical mixtures can be seen as the generative analogs of tree-based learners in supervised tabular ML (Choi et al., 2020), that have remained among the strongest supervised learners for tabular data for decades. Tractability is particularly relevant in the context of tabular data generation, as it enables exact density computation, marginalization (enabling principled handling of missing data), conditional sampling, and, under suitable structural properties, efficient MAP inference and moment queries (Gala et al., 2024, Loconte et al., 2025a,b, Peharz et al., 2020).

Empirical evidence shows our overparameterized probabilistic circuits models achieve competitive performance with respect to the state of the art, while being characterized by a faster training time and fast sampling. Furthermore, we critically re-examine these commonly used metrics, analyze their limitations, and propose more principled alternatives.

5.2 TabPC: Tabular Data Generation with Probabilistic Circuits

In this section we detail the construction of probabilistic circuits adapted for tabular data generation. Probabilistic circuits (under various names) have previously been used on tabular data, mostly for density estimation and inference, but to the best of our knowledge, never specifically for tabular data generation. To gradually increase model complexity, we begin with fully factorized models, proceed to shallow mixture models (mixtures of fully factorized models), and finally move to deep hierarchical mixtures, i.e., probabilistic circuits. This allows us to study the advantages of hierarchical mixtures over shallow mixtures and to better understand the limitations of shallow mixtures for tabular data generation.

5.2.1 Fully Factorized Model

Let $\mathbf{x} = \{x_1, \ldots, x_d\}$ be a d-dimensional vector of variables representing a table record, where each variable has either the real numbers as its domain ($\mathbb{R}$) or a set of $h \in \mathbb{N}$ categorical values represented as integers ($\{1, \ldots, h\}$). The simplest model we can construct in the probabilistic circuit framework is a fully factorized model, where each factor is a parametric distribution:

$$p(\mathbf{x}) = p(x_1, \ldots, x_d) = \prod_{i=1}^{d} p_i(x_i) \tag{5.1}$$

The distributions $p_i(x_i)$ are parametric distributions whose form depends on the variable type: for numerical variables we use Gaussian distributions $p_i(x_i) = \mathcal{N}(x_i \mid \mu_i, \sigma_i^2)$ with learnable mean μ_i and variance σ_i^2, while for categorical variables we use categorical distributions $p_i(x_i) = \text{Categorical}(x_i \mid \boldsymbol{\pi}_i)$ with learnable probability vector $\boldsymbol{\pi}_i = (\pi_{i,1}, \ldots, \pi_{i,h})$. Fitting a fully factorized model reduces to fitting each factor independently, which can be done in closed

form by computing the empirical sufficient statistics of the training data for each variable separately. Sampling is similarly straightforward: sample from each factor independently and concatenate the samples.

5.2.2 Shallow Mixtures

A mixture is a probability distribution where the density is defined as a weighted sum of other density functions. Formally, a mixture $p(\mathbf{x})$ over variables $\mathbf{x}$ has the following form:

$$p(\mathbf{x}) = \sum_{i=1}^{k} w_i q_i(\mathbf{x}) \tag{5.2}$$

where $\sum_{i=1}^{k} w_i = 1$ are the mixture weights and $q_i(\mathbf{x})$ are component distributions over variables $\mathbf{x}$, both of which may have learnable parameters. Mixtures construct complex distributions by combining simpler ones. Importantly, mixtures of tractable distributions remain tractable. In our case, we define each mixture component $q_i(\mathbf{x})$ as a fully factorized distribution:

$$p(x_1, \ldots, x_d) = \sum_{j=1}^{k} w_j \prod_{i=1}^{d} p_{j,i}(x_i) \tag{5.3}$$

where k is the number of mixture components. As before, the distributions $p_{j,i}(x_i)$ are parametric distributions (Gaussian for numerical variables, categorical for categorical variables). Figure 5.1 shows a graphical representation of a shallow mixture model.

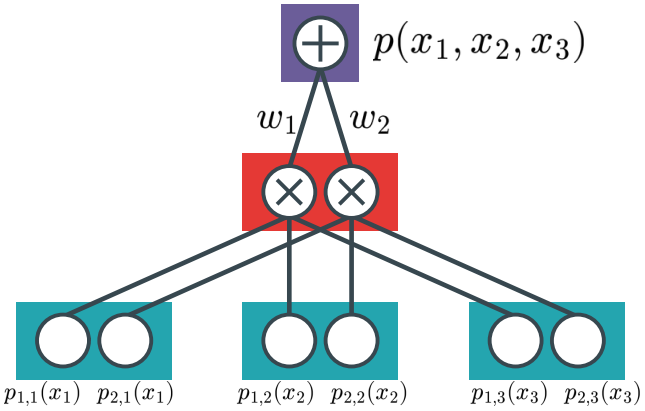


Figure 5.1: Shallow mixture model represented as a probabilistic circuit. This probabilistic circuit represents a mixture of two fully factorized components defined over three variables.

Training a shallow mixture involves learning both the mixture weights w_i and the parameters of each factor in each component $q_i(\mathbf{x})$, which is done by maximizing the log-likelihood of the training data. Sampling from a mixture is straightforward: first sample a component index $i \sim \text{Categorical}(w_1, \ldots, w_k)$, then sample from the selected component $\mathbf{x} \sim q_i(\mathbf{x})$, which is itself a fully factorized distribution.

Further details on training and optimization are provided in Appendix C.2. In our experiments, we overparameterize the shallow mixture model by using a large number of components ($10,000 \leq k \leq 50,000$). Note that both the fully factorized model and the shallow mixture model are special cases of probabilistic circuits (PCs) (Vergari et al., 2021).

5.2.3 Deep Probabilistic Circuits

We follow the framework for building overparameterized probabilistic circuits introduced in Section 2.3 and described in Loconte et al. (2025a). In particular, we use the Python library *cirkit* (Lab, 2024). The construction of a deep probabilistic circuit can be divided into two main components: (i) the definition of overparameterized computational units, and (ii) the specification of the region graph structure.

Overparameterized Computational Units. Overparameterization is essential for increasing the expressiveness of mixture models. It can be understood analogously to increasing the number of mixture components K in the shallow mixture model described above.

First, we define vectorized **input units**. Given a set of variables $\mathbf{x} = \{x_1, \dots, x_d\}$ representing the features of a table record, we use K parametric input units for each variable, meaning we have K different distributions defined over the same variable. Formally, for each variable x_i, we define a vectorized input unit $\mathbf{p}(x_i)$ as a vector of K distributions over x_i:

$$\mathbf{p}(x_i) := (p_{j,i}(x_i))_{j=1}^{K} \tag{5.4}$$

Since tabular data is typically heterogeneous, we use Gaussian input units for numerical variables

$$\mathbf{p}(x_i) := (\mathcal{N}(x_i \mid \mu_{j,i}, \sigma_{j,i}))_{j=1}^{K} \tag{5.5}$$

and categorical distributions for categorical variables

$$\mathbf{p}(x_i) := (\text{Categorical}(x_i \mid \boldsymbol{\pi}_{j,i}))_{j=1}^{K} \tag{5.6}$$

where all parameters are learnable.

Second, we define **sum-product layers** that specify how mixtures are constructed from disjoint sets of variables, which we denote with the operator $\otimes$. In this work, we use the CP sum-product layer (Loconte et al., 2025a), even though alternative sum-product layers exist. Without loss of generality, we describe how to combine two vectors of distributions, though this generalizes to any number of vectors. Let $\mathbf{p}(\mathbf{x}) = (p_i(\mathbf{x}))_{i=1}^{I_1}$ and $\mathbf{q}(\mathbf{y}) = (q_i(\mathbf{y}))_{i=1}^{I_2}$ be two vectors of distributions defined over disjoint variable sets $\mathbf{x}$ and $\mathbf{y}$. The CP layer is defined as:

$$\mathbf{p}(\mathbf{x}) \otimes \mathbf{q}(\mathbf{y}) := \mathbf{cp}(\mathbf{p}(\mathbf{x}), \mathbf{q}(\mathbf{y})) = \Big(Q_1 \mathbf{p}(\mathbf{x})\Big) \odot \Big(Q_2 \mathbf{q}(\mathbf{y})\Big) \tag{5.7}$$

where $\odot$ denotes the Hadamard (element-wise) product, and $Q_1 \in \mathbb{R}^{S \times I_1}, Q_2 \in \mathbb{R}^{S \times I_2}$ are learnable parameter matrices. In general, the input dimensions I_1, I_2 can differ from the output dimension S. However, for simplicity, we use $I_1 = I_2 = S$ and refer to this shared value as the number of units K.

To fully specify a vectorized probabilistic circuit, we must also define how two vectors of distributions over the same scope $\mathbf{p}(\mathbf{x}) = (p_i(\mathbf{x}))_{i=1}^{I_1}$ and $\mathbf{q}(\mathbf{x}) = (q_i(\mathbf{x}))_{i=1}^{I_2}$ are combined, i.e., the sum layer operation $\oplus$. We concatenate the two vectors and apply S weighted sums to the resulting vector:

$$\mathbf{p}(\mathbf{x}) \oplus \mathbf{q}(\mathbf{x}) := W \mathbf{r}(\mathbf{x}) \tag{5.8}$$

where $W \in \mathbb{R}^{S \times (I_1 + I_2)}$ is a learnable weight matrix, and $\mathbf{r}(\mathbf{x})$ is the concatenation of the two input distribution vectors.

Finally, we define how a vector of distributions is aggregated into a single distribution through a weighted sum:

$$p(\mathbf{x}) = \mathbf{w} \cdot \mathbf{q}(\mathbf{x}) \tag{5.9}$$

where $\mathbf{w}$ is a learnable weight vector. This is necessary to obtain a single distribution at the root of the circuit.

The parameters of the circuit consist of the parameters of the input units and the weights in the sum and sum-product layers. Overparameterization enables arbitrary scaling of circuit capacity by increasing the number of units K, thereby enhancing expressiveness.

Region Graph. The region graph describes the hierarchical structure of the mixture as a hierarchical partitioning of variables, specifying at what depth different components are combined (Dennis and Ventura, 2012, Vergari et al., 2021). For example, given input variables $\mathbf{x} = \{x_1, x_2, x_3\}$ with distributions $\mathbf{p}(x_1), \mathbf{q}(x_2), \mathbf{r}(x_3)$, we can combine them as either $(\mathbf{p}(x_1) \otimes \mathbf{q}(x_2)) \otimes \mathbf{r}(x_3)$ or $\mathbf{p}(x_1) \otimes (\mathbf{q}(x_2) \otimes \mathbf{r}(x_3))$. Formally, a region graph is a bipartite graph where each node is either a region $\mathcal{R} \subseteq \mathbf{x}$ or a partition describing how a region splits into disjoint parts: $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ with $\mathcal{R}_1 \cap \mathcal{R}_2 = \emptyset$. The root must be the full region $\mathcal{R} = \mathbf{x}$. Figures 5.2 and 5.3 show examples of region graphs.

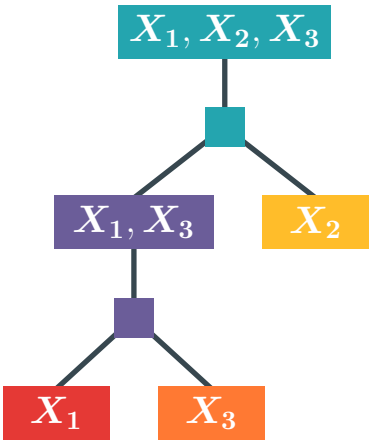


Figure 5.2: Example of a tree region graph partitioning a scope of 3 variables. Given input units $\mathbf{p}(x_1), \mathbf{q}(x_2), \mathbf{r}(x_3)$, the circuit corresponding to this region graph would be $c(x_1, x_2, x_3) = w \cdot \Big[(\mathbf{p}(x_1) \otimes \mathbf{r}(x_3)) \otimes \mathbf{q}(x_2)\Big]$. Image from Loconte et al. (2025a).

Choosing the region graph that will lead to the best performing circuit is a challenging task, and several approaches have been proposed in the literature. While some methods are completely data-independent, others exploit the training data to guide the construction of the region graph. In particular, we resort to a data-dependent method based on the Chow-Liu algorithm (Choi et al., 2011, Rahman et al., 2014, Vergari et al., 2015). The Chow-Liu algorithm (Chow and Liu, 1968) is an efficient method for constructing a tree-shaped Bayesian network approximating a joint distribution, from which a dataset of samples is available. Once the Chow-Liu tree is built, where nodes are the variables, a node is chosen as the root of the tree (in order to obtain a balanced tree), then a region graph is built by progressively merging scopes starting from the leaf nodes toward the root. In such a region graph, highly dependent variables are mixed together earlier. We show in Figure 5.5 and 5.4 an example of a region graph built from a Chow-Liu tree learned from the Magic dataset (Bock, 2004). For more details about the construction of region graphs, we refer the reader to Loconte et al. (2025a).

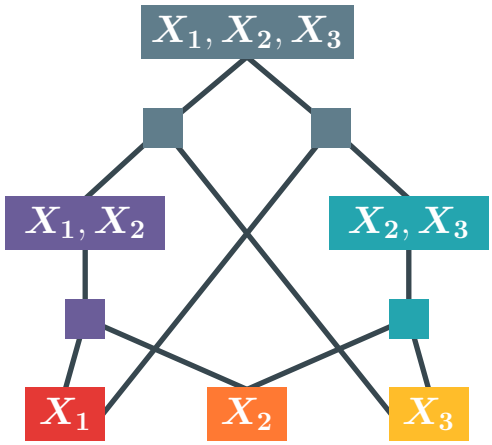


Figure 5.3: Example of a region graph partitioning a scope of 3 variables, where some region nodes are shared between partitionings. Note how the region node X_1, X_2, X_3 is partitioned in two different ways: $(X_1, X_2) \times X_3$ and $X_1 \times (X_2, X_3)$. Starting from the leaf nodes and moving up the DAG, edges entering the partition nodes (the empty squares) represent products of disjoint sets of variables, while edges entering the region nodes represent weighted sums of distributions defined over the same scope. Given input units $\mathbf{p}(x_1), \mathbf{q}(x_2), \mathbf{r}(x_3)$, the circuit corresponding to this region graph would be $c(x_1, x_2, x_3) = w \cdot \Big[\big(\mathbf{p}(x_1) \otimes \mathbf{q}(x_2)\big) \otimes \mathbf{r}(x_3) \oplus \big(\mathbf{q}(x_2) \otimes \mathbf{r}(x_3)\big) \otimes \mathbf{p}(x_1)\Big]$. Image from Loconte et al. (2025a).

As an example of probabilistic circuits for tabular data, we show in Figure 5.6 the probabilistic circuit for the Magic dataset built from the region graph in Figure 5.5 and using CP as the sum-product layer.

Following the naming approach of prior work, we refer to our probabilistic circuits for tabular data as *TabPC* (Tabular-data Probabilistic Circuit). Note how probabilistic circuits can seamlessly handle heterogeneous data types, combining different types of input units within the same model. We next describe how to train TabPC and how to sample from it, which are the two main operations we need in order to use it as a generative model for tabular data.

Training TabPC. Given a dataset of samples $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, we first construct the probabilistic circuit. This consists of two steps: (i) building the region graph based on the Chow-Liu tree described previously, and (ii) populating the region graph with the vectorized input units and computational units described above.

Once the probabilistic circuit $c(\mathbf{x})$ is constructed, we train it by maximizing the log-likelihood of the training data, using the same optimization techniques employed for neural networks. The hyperparameters reduce to the number of units K, which we adapt to each dataset, along with standard optimization hyperparameters (optimizer, learning rate, batch size, number of epochs, etc.). Further details on training and optimization are provided in Appendix C.2.

Sampling from TabPC. A basic sampling algorithm for probabilistic circuits was described in Section 2.3. In our experiments, we use a memory-efficient variant of this algorithm. We define it recursively, starting from the root node of the circuit $c(\mathbf{x})$ and traversing down the DAG until reaching the input units:

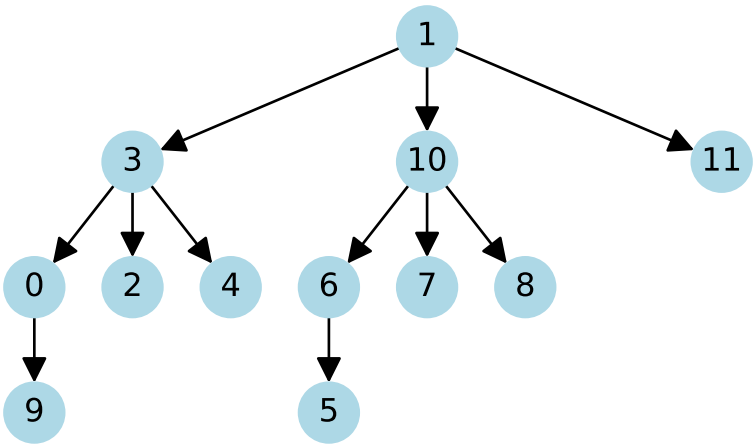


Figure 5.4: Chow-Liu tree learned from the Magic dataset. Numbers from 0 to 10 indicate the different features/columns of the dataset.

- If $c(\mathbf{x})$ is a sum node $c(\mathbf{x}) = \sum_i^S w_i p_i(\mathbf{x})$, sample a component index according to the mixture weights $i \sim \text{Categorical}(i \mid w_1, \ldots, w_S)$, then recursively sample $\mathbf{x} \sim p_i(\mathbf{x})$.
- If $c(\mathbf{x})$ is a product node $c(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_i^n p_i(\mathbf{x}_i)$, recursively sample $\mathbf{x}_i \sim p_i(\mathbf{x}_i)$ for each child p_i, then concatenate the samples $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.
- If $c(x)$ is an input unit, sample directly from it: $x \sim c(x)$.

This algorithm effectively selects one input unit for each variable and samples from them to construct a complete sample. A more general algorithm for conditional sampling with arbitrary evidence is provided in Appendix C.1.

5.3 Benchmarking Tabular Data Generation Methods

In this chapter we aim to compare probabilistic circuits with these state-of-the-art deep generative models for tabular data generation. These models have often been evaluated on a common set of benchmarks, including datasets, pre-processing pipelines and evaluation metrics. In order to compare with open-source state-of-the-art methods, we will follow the same consolidated pipeline and use the same datasets, metrics and baselines as the most recent state of the art methods based on diffusion, including TabDiff (Shi et al., 2025), TabSyn (Zhang et al., 2024) and TabbyFlow (Guzmán-Cordero et al., 2025).

Datasets. The range of datasets used in tabular data generation is quite broad, spanning various domains and sizes. However, there is a common set of datasets from the UCI machine learning repository[1] that have been widely adopted in recent works: Adult (Becker and Kohavi, 1996), Beijing (Chen, 2015), Default (Yeh, 2009), Diabetes (Strack et al., 2014), Magic (Bock, 2004), News (Fernandes et al., 2015) and Shoppers (Sakar and Kastro, 2018). Each dataset contains both categorical and continuous features, and it is associated with a classification or regression task (meaning that one feature is the target variable). More information about the datasets are reported in Table 5.1.

[1] https://archive.ics.uci.edu/datasets

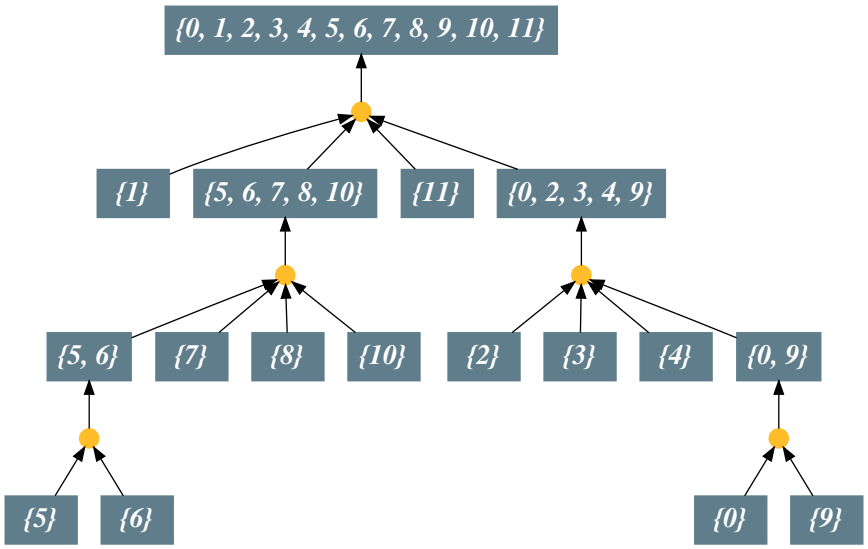


Figure 5.5: Region graph built from the Chow-Liu tree in Figure 5.4. Numbers from 0 to 10 indicate the different features/columns of the dataset.

Moreover, it has become common practice in tabular data generation to pre-process the datasets by replacing missing values with the column mean for numerical variables. For the sake of comparability we will train our model on the same set of datasets and follow the same protocol for removing missing data, although this is not necessary for tractable models, as they can seamlessly handle missing data by marginalization of the missing variables. In particular we follow the same preprocessing steps as Shi et al. (2025).

Metrics. We compare against common metrics used in similar recent works on tabular data generation:

- *Fidelity* metrics: Shape, Trend, Detection based on logistic regression, α-Precision and β-Recall (Alaa et al., 2022);
- *Utility* metric: Machine Learning Efficiency (MLE);
- *Privacy* metric: Distance to Closest Record (DCR) as used in Shi et al. (2025).

Unless specified otherwise, we will follow the same definitions and implementations as in Shi et al. (2025) in order to ensure comparability, many of which are based on the open-source SDMetrics library (SDV Developers, DataCebo, 2024)[2].

Baselines. Many recent works on tabular data generation compare against a more or less common set of baselines. These are chosen to cover most of the main families of deep generative models. Following Shi et al. (2025), we will compare probabilistic circuits with the following open-source models: TVAE (Xu et al., 2019) that is based on VAEs, CTGAN (Xu et al., 2019)

[2] https://docs.sdv.dev/sdmetrics

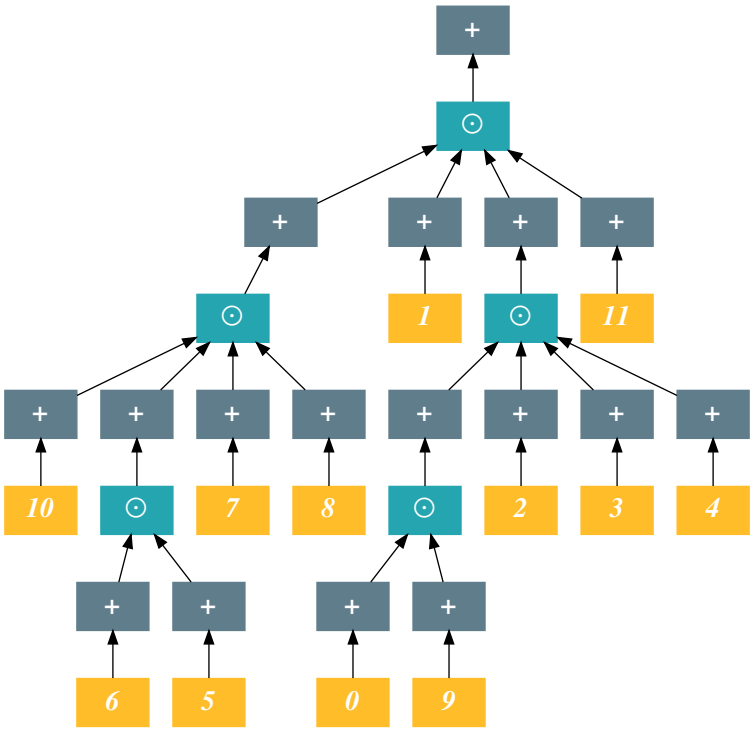


Figure 5.6: Probabilistic circuit for the Magic dataset built from the region graph in Figure 5.5. Note how scopes are merged using a CP layer, which corresponds to two sum layers followed by a Hadamard product layer. Here $\odot$ refers to the Hadamard product, while $+$ refers to multiplication with a weights matrix (the representation makes explicit the use of CP sum-layers).

based on GANs, GReaT (Borisov et al., 2023) that is a language model, and STaSy (Kim et al., 2023), CoDi (Lee et al., 2023), TabDDPM (Kotelnikov et al., 2023), TabSyn (Zhang et al., 2024) and TabDiff (Shi et al., 2025) based on diffusion. In order to evaluate the baselines we trained them using the same codebase and the same hyperparameters as in Shi et al. (2025), although for some models we were not able to reproduce similar results as those reported[3]. For each model and dataset, we generated synthetic data having the same number of records as the training data, following the common practice in tabular data generation. Moreover, for each model and dataset, we generated 20 different synthetic datasets using different random seeds, and we report the average and standard deviation of the metrics across these 20 runs. For GReaT and STaSy, we were not able to produce results for all datasets, due to out-of-memory errors.

5.4 Shortcomings of Common Synthetic Tabular Data Metrics

Although the set of metrics mentioned above is widely adopted by the community, some of them present major flaws, especially the fidelity metrics. In particular we will focus on *Shape*, *Trend*

[3]For TabDDPM, we chose to copy the results from Shi et al. (2025), as we were not able to train the models properly.

Table 5.1: Dataset statistics. # Num = number of numerical columns, # Cat = number of categorical columns, # Max Cat = max categories in any categorical column. # Train and # Test refer to the number of samples in the training and test splits respectively.

Dataset	# Rows	# Num	# Cat	# Max Cat	# Train	# Test	Task
Adult	48,842	6	9	42	32,561	16,281	Classification
Default	30,000	14	11	11	27,000	3,000	Classification
Shoppers	12,330	10	8	20	11,097	1,233	Classification
Magic	19,019	10	1	2	17,117	1,902	Classification
Beijing	43,824	7	5	31	39,441	4,383	Regression
News	39,644	46	2	7	35,679	3,965	Regression
Diabetes	101,766	9	27	716	81,412	20,354	Classification

and *Detection* based on logistic regression, which are commonly reported in the recent literature.

First of all, a major pitfall is that different recent state of the art models have already saturated these metrics (TabSyn, TabDiff, TabbyFlow), such that only very marginal improvements are possible. Moreover, we show how even trivial models with the appropriate pre-processing are able to achieve competitive fidelity scores.

In order to improve the ability to capture complex marginals, we include two pre-processing steps before fitting our models:

- **Quantile normalization of numerical features**: We apply a monotonic scalar transformation mapping samples from any distribution into samples from a standard Gaussian, for each feature separately. This is done in most recent diffusion models for tabular data. Even though the quantile normalization is often only marginally discussed, it is a fundamental component of state-of-the-art models as TabDiff or TabSyn. For a more detailed explanation of this transformation, we refer the reader to Section 2.5.3 of Chapter 2.

- **Handling of inflated values**: For each numerical feature affected by inflated values, i.e., specific values that are oversampled (often the minimum or the maximum possible values), we add a new categorical feature indicating if the value is inflated. These inflated values are subsequently "removed" (i.e., considered missing) from the original columns. Normally, circuits can handle this by simply marginalizing over missing values. In the case of fully factorized distributions, these values are simply ignored when fitting each factor. One can think of these features as having a mixed-type (Zhao et al., 2021) since they can assume either a numerical value or specific numbers treated as categorical values.

We still report the performance of the simple fully factorized model without any pre-processing, to highlight the shortcomings of the common metrics used in tabular data generation.

We show the performance in terms of Shape, Trend, and logistic-regression-based Detection (LR-C2ST) of the synthetic data generated by the fully factorized model (with and without pre-processing) compared with the baselines in Table 5.2, Table 5.3 and Table 5.4, and we provide an overview in Figure 5.7. From these results, we can already gain some insights about the inadequacy of these metrics:

- First of all, it seems that fidelity metrics are already saturated by many recent models,

including TabDiff and TabSyn[4]. This means that only marginal improvements are possible on these metrics, making them ineffective to meaningfully distinguish between different models.

- The fully factorized model without any pre-processing is already capable of achieving perfect or near-perfect scores for the Detection metric based on logistic regression (LR-C2ST). This means that this metric is not able to meaningfully capture the quality of the generated data, as this simple model is clearly not capable of either modelling marginals more complex than a simple Gaussian or categorical, or capturing dependencies between features.
- The fully factorized model with the appropriate pre-processing is also able to achieve very high scores for the Trend metric, despite the fact that all features are generated independently. Moreover, it seems that the preprocessing, that operates independently for each feature, positively affects the Trend metric, that is not supposed to measure how well marginals are modelled. This highlights how this metric is not able to meaningfully capture the quality of correlations and dependencies between pairs of features.
- The fully factorized model with the appropriate pre-processing is able to achieve near-perfect scores for the Shape metric, meaning that preprocessing can be alone really effective in capturing complex marginals.

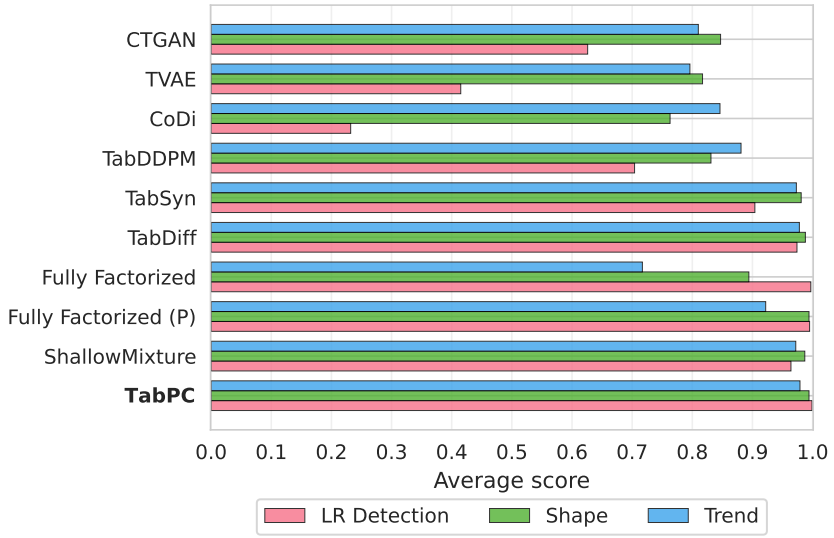


Figure 5.7: Overview of common fidelity metrics and their saturation in tabular data generation. We report average scores across all datasets and runs for each model. Near-perfect scores across multiple methods indicate metric saturation. GReaT and STaSy are excluded due to incomplete results across datasets (this exclusion applies to subsequent overview figures).

On the other hand, Machine Learning Efficiency (MLE) is not saturated, and our simplistic fully factorized model is not able to achieve competitive results. We report the MLE scores for all models and datasets in Table 5.5 and we show an overview with aggregated scores in Figure 5.8.

This provides empirical evidence that Trend and LR-C2ST metrics are problematic. However, the saturation of these metrics may be specific to this commonly used dataset collection. While these

[4]performances for TabbyFlow (Guzmán-Cordero et al., 2025) are not reported, but in their article they also report saturated fidelity metrics

Table 5.2: **Shape** scores (higher is better). Higher scores reflect superior performance. When the value is not reported, it means that the method was not able to generate valid samples on that dataset, usually because of out-of-memory errors.

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	$0.838_{\pm 0.001}$	$0.809_{\pm 0.001}$	$0.835_{\pm 0.001}$	$0.895_{\pm 0.000}$	$0.951_{\pm 0.001}$	$0.862_{\pm 0.000}$	$0.738_{\pm 0.001}$	0.847
TVAE	$0.756_{\pm 0.000}$	$0.736_{\pm 0.001}$	$0.911_{\pm 0.000}$	$0.774_{\pm 0.000}$	$0.951_{\pm 0.001}$	$0.814_{\pm 0.000}$	$0.778_{\pm 0.001}$	0.817
GReaT	$0.425_{\pm 0.000}$	$0.934_{\pm 0.001}$	$0.804_{\pm 0.001}$	–	$0.85_{\pm 0.001}$	–	$0.858_{\pm 0.001}$	–
STaSy	$0.915_{\pm 0.001}$	$0.898_{\pm 0.001}$	$0.922_{\pm 0.001}$	–	$0.89_{\pm 0.002}$	$0.92_{\pm 0.000}$	$0.844_{\pm 0.002}$	–
CoDi	$0.789_{\pm 0.001}$	$0.761_{\pm 0.000}$	$0.749_{\pm 0.001}$	$0.787_{\pm 0.000}$	$0.896_{\pm 0.002}$	$0.699_{\pm 0.000}$	$0.662_{\pm 0.001}$	0.763
TabDDPM	$0.982_{\pm 0.000}$	$0.987_{\pm 0.000}$	$0.984_{\pm 0.001}$	$0.686_{\pm 0.000}$	$0.99_{\pm 0.001}$	$0.212_{\pm 0.000}$	$0.973_{\pm 0.001}$	0.831
TabSyn	$0.992_{\pm 0.000}$	$0.969_{\pm 0.001}$	$0.99_{\pm 0.000}$	$0.982_{\pm 0.000}$	$0.991_{\pm 0.001}$	$0.956_{\pm 0.000}$	$0.986_{\pm 0.001}$	0.981
TabDiff	$0.994_{\pm 0.000}$	$0.99_{\pm 0.001}$	$0.988_{\pm 0.001}$	$0.989_{\pm 0.000}$	$0.992_{\pm 0.001}$	$0.975_{\pm 0.000}$	$0.986_{\pm 0.001}$	0.988
Fully Factorized	$0.905_{\pm 0.000}$	$0.938_{\pm 0.000}$	$0.835_{\pm 0.001}$	$0.994_{\pm 0.000}$	$0.901_{\pm 0.001}$	$0.809_{\pm 0.000}$	$0.875_{\pm 0.001}$	0.894
Fully Factorized (P)	$0.996_{\pm 0.000}$	$0.995_{\pm 0.000}$	$0.995_{\pm 0.000}$	$0.997_{\pm 0.000}$	$0.993_{\pm 0.001}$	$0.991_{\pm 0.000}$	$0.993_{\pm 0.000}$	0.994
ShallowMixture	$0.982_{\pm 0.001}$	$0.985_{\pm 0.001}$	$0.991_{\pm 0.001}$	$0.995_{\pm 0.000}$	$0.985_{\pm 0.002}$	$0.985_{\pm 0.000}$	$0.985_{\pm 0.001}$	0.987
TabPC	$0.995_{\pm 0.001}$	$0.995_{\pm 0.000}$	$0.993_{\pm 0.001}$	$0.995_{\pm 0.000}$	$0.991_{\pm 0.001}$	$0.994_{\pm 0.000}$	$0.993_{\pm 0.001}$	0.994

Table 5.3: **Trend** scores (higher is better).

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	$0.77_{\pm 0.001}$	$0.704_{\pm 0.001}$	$0.719_{\pm 0.015}$	$0.811_{\pm 0.002}$	$0.939_{\pm 0.004}$	$0.95_{\pm 0.001}$	$0.775_{\pm 0.004}$	0.81
TVAE	$0.652_{\pm 0.004}$	$0.711_{\pm 0.011}$	$0.818_{\pm 0.016}$	$0.659_{\pm 0.002}$	$0.959_{\pm 0.006}$	$0.942_{\pm 0.001}$	$0.832_{\pm 0.002}$	0.796
GReaT	$0.191_{\pm 0.002}$	$0.907_{\pm 0.032}$	$0.776_{\pm 0.022}$	–	$0.904_{\pm 0.006}$	–	$0.889_{\pm 0.006}$	–
STaSy	$0.887_{\pm 0.003}$	$0.902_{\pm 0.001}$	$0.912_{\pm 0.002}$	–	$0.955_{\pm 0.004}$	$0.973_{\pm 0.001}$	$0.866_{\pm 0.001}$	–
CoDi	$0.78_{\pm 0.002}$	$0.923_{\pm 0.002}$	$0.824_{\pm 0.003}$	$0.686_{\pm 0.000}$	$0.945_{\pm 0.004}$	$0.958_{\pm 0.001}$	$0.808_{\pm 0.001}$	0.846
TabDDPM	$0.97_{\pm 0.002}$	$0.973_{\pm 0.001}$	$0.951_{\pm 0.001}$	$0.485_{\pm 0.000}$	$0.983_{\pm 0.002}$	$0.868_{\pm 0.001}$	$0.934_{\pm 0.002}$	0.881
TabSyn	$0.982_{\pm 0.002}$	$0.95_{\pm 0.001}$	$0.976_{\pm 0.007}$	$0.959_{\pm 0.001}$	$0.992_{\pm 0.002}$	$0.976_{\pm 0.000}$	$0.978_{\pm 0.002}$	0.973
TabDiff	$0.985_{\pm 0.001}$	$0.974_{\pm 0.002}$	$0.96_{\pm 0.013}$	$0.971_{\pm 0.001}$	$0.991_{\pm 0.002}$	$0.982_{\pm 0.000}$	$0.982_{\pm 0.001}$	0.978
Fully Factorized	$0.611_{\pm 0.007}$	$0.73_{\pm 0.003}$	$0.463_{\pm 0.003}$	$0.918_{\pm 0.002}$	$0.784_{\pm 0.003}$	$0.907_{\pm 0.001}$	$0.605_{\pm 0.002}$	0.717
Fully Factorized (P)	$0.925_{\pm 0.000}$	$0.91_{\pm 0.001}$	$0.888_{\pm 0.001}$	$0.968_{\pm 0.001}$	$0.87_{\pm 0.001}$	$0.955_{\pm 0.001}$	$0.937_{\pm 0.001}$	0.922
ShallowMixture	$0.968_{\pm 0.001}$	$0.968_{\pm 0.003}$	$0.976_{\pm 0.001}$	$0.976_{\pm 0.001}$	$0.982_{\pm 0.001}$	$0.969_{\pm 0.000}$	$0.965_{\pm 0.001}$	0.972
TabPC	$0.986_{\pm 0.001}$	$0.976_{\pm 0.003}$	$0.962_{\pm 0.013}$	$0.982_{\pm 0.002}$	$0.982_{\pm 0.004}$	$0.98_{\pm 0.002}$	$0.984_{\pm 0.002}$	0.979

Table 5.4: **Detection** score (C2ST) using logistic regression classifier (higher is better). Higher scores reflect superior performance.

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	$0.62_{\pm 0.004}$	$0.641_{\pm 0.006}$	$0.488_{\pm 0.004}$	$0.475_{\pm 0.005}$	$0.841_{\pm 0.006}$	$0.793_{\pm 0.006}$	$0.527_{\pm 0.006}$	0.626
TVAE	$0.21_{\pm 0.019}$	$0.331_{\pm 0.005}$	$0.666_{\pm 0.003}$	$0.054_{\pm 0.006}$	$0.841_{\pm 0.006}$	$0.442_{\pm 0.012}$	$0.359_{\pm 0.011}$	0.415
GReaT	–	$0.768_{\pm 0.004}$	$0.48_{\pm 0.004}$	–	$0.463_{\pm 0.003}$	–	$0.441_{\pm 0.003}$	–
STaSy	$0.48_{\pm 0.004}$	$0.675_{\pm 0.003}$	$0.64_{\pm 0.003}$	–	$0.579_{\pm 0.004}$	$0.465_{\pm 0.002}$	$0.223_{\pm 0.003}$	–
CoDi	$0.209_{\pm 0.006}$	$0.12_{\pm 0.002}$	$0.328_{\pm 0.003}$	$0.002_{\pm 0.000}$	$0.731_{\pm 0.003}$	$0.026_{\pm 0.001}$	$0.208_{\pm 0.003}$	0.232
TabDDPM	$0.976_{\pm 0.000}$	$0.951_{\pm 0.000}$	$0.971_{\pm 0.000}$	$0.198_{\pm 0.000}$	$1.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.835_{\pm 0.000}$	0.704
TabSyn	$0.984_{\pm 0.004}$	$0.897_{\pm 0.003}$	$0.988_{\pm 0.007}$	$0.648_{\pm 0.003}$	$0.999_{\pm 0.002}$	$0.823_{\pm 0.004}$	$0.986_{\pm 0.008}$	0.904
TabDiff	$0.997_{\pm 0.002}$	$0.978_{\pm 0.004}$	$0.967_{\pm 0.004}$	$0.959_{\pm 0.005}$	$0.998_{\pm 0.002}$	$0.94_{\pm 0.005}$	$0.979_{\pm 0.008}$	0.974
Fully Factorized	$1.0_{\pm 0.002}$	$0.998_{\pm 0.004}$	$0.996_{\pm 0.004}$	$1.0_{\pm 0.000}$	$0.993_{\pm 0.009}$	$0.994_{\pm 0.007}$	$1.0_{\pm 0.001}$	0.997
Fully Factorized (P)	$0.999_{\pm 0.002}$	$0.998_{\pm 0.002}$	$0.995_{\pm 0.005}$	$1.0_{\pm 0.000}$	$0.995_{\pm 0.005}$	$0.978_{\pm 0.013}$	$0.998_{\pm 0.002}$	0.995
ShallowMixture	$0.955_{\pm 0.003}$	$0.941_{\pm 0.005}$	$0.988_{\pm 0.005}$	$1.0_{\pm 0.000}$	$0.952_{\pm 0.005}$	$0.939_{\pm 0.014}$	$0.971_{\pm 0.009}$	0.964
TabPC	$0.999_{\pm 0.002}$	$0.999_{\pm 0.001}$	$0.999_{\pm 0.001}$	$1.0_{\pm 0.000}$	$0.997_{\pm 0.004}$	$0.998_{\pm 0.003}$	$1.0_{\pm 0.001}$	0.999

metrics might not saturate on more complex datasets, we observe saturation even on relatively high-dimensional tabular datasets such as News. We provide below a more in-depth analysis of the shortcomings of these metrics, and we propose and evaluate alternatives.

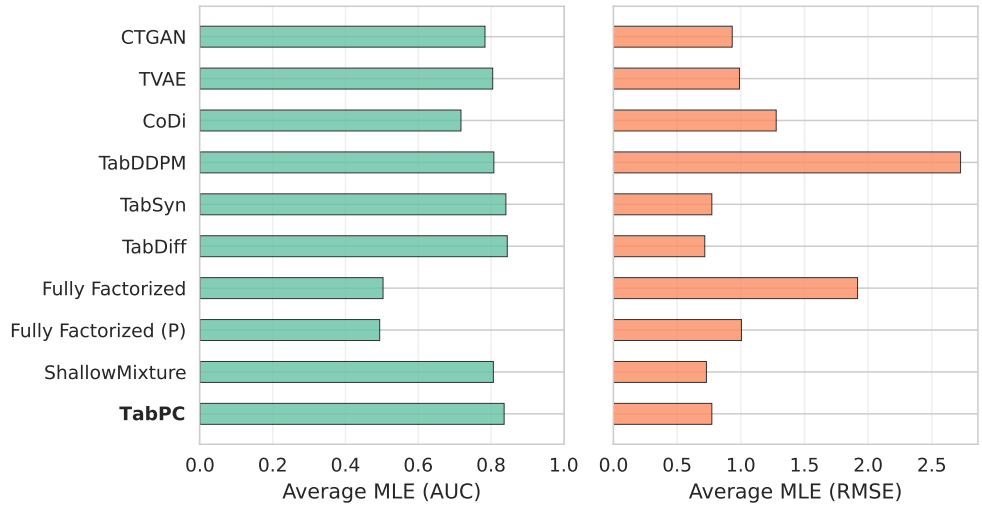


Figure 5.8: Machine Learning Efficiency (MLE) scores for different methods and datasets. The left plot shows the average scores across the classification datasets (Adult, Default, Diabetes, Shoppers), that is computed as the AUCROC. The right plot shows the average scores across the regression datasets (Beijing, News), that is computed as the RMSE.

Trend. The Trend metric is computed as the average of scores for each pair of features: if the two features are both numerical, the Pearson Score is used, while if they are both categorical the Contingency Score is used. For mixed pairs, the numerical feature is discretized and the Contingency Score is used. Since the contingency score is based on the total variation distance between joint distributions, it is heavily affected by the marginal distributions of each feature, meaning that if the marginals are not well modelled, the joint distribution will also be affected. Hence, improving the marginals of numerical features positively affects the Trend score, since the Contingency Score of mixed pairs will benefit from the better marginals of numerical features, that will get discretized in order to compute the score. Accordingly, we argue that the Trend metric cannot meaningfully capture the quality of correlations and dependencies between pairs of features since it is affected by the quality of marginals, that should be measured by the Shape metric. Furthermore, the competitive score achieved by the fully factorized model can be explained by the fact that most pairs of features in the used datasets are characterized by a weak or absent statistical dependence.

In order to better measure how well synthetic data reproduce dependencies between variables, we propose a metric based on mutual information that we call Normalized Mutual Information Error (NMIE). Instead of correlation or contingency similarity, we measure the normalized mutual information (NMI) error between two columns x and y:

$$\text{NMIE}(x, y) := |\text{NMI}_R(x, y) - \text{NMI}_S(x, y)| \tag{5.10}$$

where $\text{NMI}(x, y) := \frac{2I(x,y)}{H(x)+H(y)}$ (Kvalseth, 1987), and I, H refer to empirical mutual information and entropy. This definition guarantees $\text{NMI} \in [0, 1]$, in analogy to the Pearson correlation coefficient. A similar metric based on mutual information has been previously proposed by Yang et al. (2024).

We identify three main advantages in using the NMIE instead of the Trend metric:

1. NMIE provides a unified way of measuring scores across different feature types. The Trend score instead mixes correlation and contingency similarity scores in an inconsistent way.

Table 5.5: **MLE** (Machine Learning Efficiency) Scores. AUC and RMSE are used for classification (c) and regression (r) tasks, respectively. The trained predictive models are based on XGBoost. For more details about the implementation please refer to Shi et al. (2025).

Method	Adult (c)	Beijing (r)	Default (c)	Diabetes (c)	Magic (c)	News (r)	Shoppers (c)
CTGAN	$0.857_{\pm 0.004}$	$0.978_{\pm 0.025}$	$0.726_{\pm 0.012}$	$0.596_{\pm 0.006}$	$0.88_{\pm 0.004}$	$0.887_{\pm 0.024}$	$0.858_{\pm 0.009}$
TVAE	$0.86_{\pm 0.007}$	$1.009_{\pm 0.023}$	$0.746_{\pm 0.005}$	$0.607_{\pm 0.005}$	$0.903_{\pm 0.003}$	$0.97_{\pm 0.016}$	$0.907_{\pm 0.004}$
GReaT	$0.842_{\pm 0.007}$	$0.628_{\pm 0.016}$	$0.757_{\pm 0.005}$	–	$0.914_{\pm 0.003}$	–	$0.904_{\pm 0.004}$
STaSy	$0.906_{\pm 0.001}$	$0.678_{\pm 0.015}$	$0.756_{\pm 0.003}$	–	$0.934_{\pm 0.004}$	$0.87_{\pm 0.023}$	$0.909_{\pm 0.004}$
CoDi	$0.809_{\pm 0.012}$	$0.8_{\pm 0.036}$	$0.506_{\pm 0.011}$	$0.494_{\pm 0.028}$	$0.932_{\pm 0.003}$	$1.756_{\pm 0.142}$	$0.846_{\pm 0.016}$
TabDDPM	$0.907_{\pm 0.001}$	$0.592_{\pm 0.011}$	$0.758_{\pm 0.004}$	$0.521_{\pm 0.008}$	$0.935_{\pm 0.003}$	$4.86_{\pm 3.04}$	$0.918_{\pm 0.005}$
TabSyn	$0.91_{\pm 0.001}$	$0.664_{\pm 0.019}$	$0.761_{\pm 0.004}$	$0.686_{\pm 0.002}$	$0.935_{\pm 0.003}$	$0.881_{\pm 0.027}$	$0.912_{\pm 0.005}$
TabDiff	$0.912_{\pm 0.002}$	$0.57_{\pm 0.014}$	$0.764_{\pm 0.005}$	$0.693_{\pm 0.002}$	$0.937_{\pm 0.003}$	$0.864_{\pm 0.024}$	$0.917_{\pm 0.006}$
Fully Factorized	$0.509_{\pm 0.052}$	$1.196_{\pm 0.103}$	$0.488_{\pm 0.033}$	$0.496_{\pm 0.011}$	$0.496_{\pm 0.043}$	$2.638_{\pm 0.152}$	$0.527_{\pm 0.069}$
Fully Factorized (P)	$0.511_{\pm 0.056}$	$1.077_{\pm 0.039}$	$0.508_{\pm 0.019}$	$0.503_{\pm 0.011}$	$0.486_{\pm 0.039}$	$0.933_{\pm 0.023}$	$0.462_{\pm 0.075}$
ShallowMixture	$0.91_{\pm 0.002}$	$0.572_{\pm 0.008}$	$0.729_{\pm 0.006}$	$0.585_{\pm 0.005}$	$0.925_{\pm 0.004}$	$0.888_{\pm 0.026}$	$0.883_{\pm 0.006}$
TabPC	$0.917_{\pm 0.002}$	$0.664_{\pm 0.023}$	$0.747_{\pm 0.006}$	$0.689_{\pm 0.003}$	$0.932_{\pm 0.003}$	$0.881_{\pm 0.016}$	$0.895_{\pm 0.005}$

2. NMI can also capture non-linear relationships among pairs of numerical variables while, the correlation similarity component of Trend is limited to linear relationships.

3. For pairs of categorical variables, the contingency similarity component of Trend also depends on the quality of the marginals, while the NMIE does not. For example, any fully factorized model will have zero NMI between any pair of features, independently on the marginals, leading to the same overall NMIE score.

Moreover, instead of averaging the score among all pairs of features in a uniform way, we propose to aggregate the errors using weights based on the NMI of each pair of features, leading to the weighted Normalized Mutual Information Error (wNMIE):

$$\text{wNMIE} \coloneqq \sum_{\{x_i, x_j \,:\, i<j\}} \widetilde{w}(x_i, x_j) |\text{NMI}_R(x_i, x_j) - \text{NMI}_S(x_i, x_j)|, \tag{5.11}$$

where x_i, x_j are columns of the dataset, and the weights are defined as:

$$w(x_i, x_j) \coloneqq |\text{NMI}_R(x_i, x_j) + \text{NMI}_S(x_i, x_j)|, \tag{5.12}$$

$$\widetilde{w}(x_i, x_j) \coloneqq \frac{w(x_i, x_j)}{\sum_{i<j} w(x_i, x_j)}. \tag{5.13}$$

In this way, we reduce the impact of column pairs with low NMI on the overall score. As a consequence, the wNMIE score reflects how well synthetic data models relationships between columns only where a statistical dependence exists, as opposed to being increased simply by the presence of largely independent columns.

We show in the right panel of Figure 5.9 and in Table 5.6 the wNMIE for different methods. As expected, the value for the fully factorized model does not change when applying the preprocessing step that results in an improvement of marginals. This is due to the fact that the (empirical) mutual information between any two features in a fully factorized model is null, independently on the factors. Moreover, it achieves a relatively high error, highlighting its inability to model dependencies between features compared to the other methods.

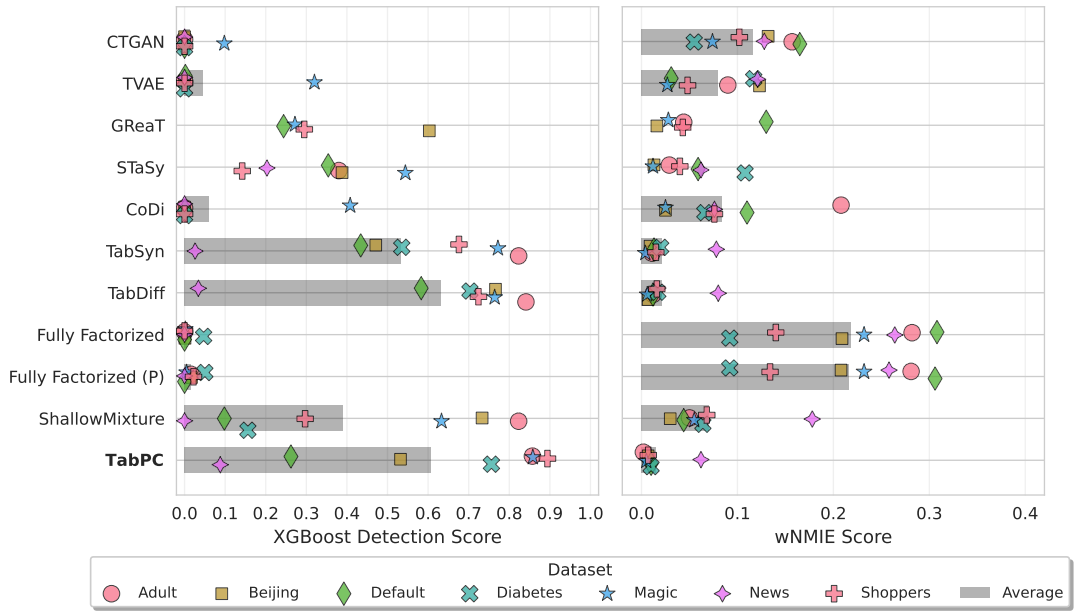


Figure 5.9: XGB C2ST (left panel) and wNMIE (right panel) for different methods and datasets. For each datset, we report the average performace across 20 sampled datasets. Best performing methods are those with the a large XGB C2ST and low wNMIE.

Table 5.6: **wNMIE** (weighted NMI Error) scores (lower is better). The lower the score, the better the performance.

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	$0.157_{\pm 0.001}$	$0.132_{\pm 0.001}$	$0.165_{\pm 0.002}$	$0.055_{\pm 0.000}$	$0.074_{\pm 0.001}$	$0.128_{\pm 0.001}$	$0.102_{\pm 0.002}$	0.116
TVAE	$0.09_{\pm 0.000}$	$0.123_{\pm 0.000}$	$0.031_{\pm 0.001}$	$0.117_{\pm 0.001}$	$0.027_{\pm 0.001}$	$0.121_{\pm 0.001}$	$0.048_{\pm 0.001}$	0.080
GReaT	$0.044_{\pm 0.000}$	$0.016_{\pm 0.001}$	$0.13_{\pm 0.001}$	–	$0.028_{\pm 0.001}$	–	$0.043_{\pm 0.002}$	–
STaSy	$0.029_{\pm 0.001}$	$0.013_{\pm 0.001}$	$0.059_{\pm 0.001}$	–	$0.012_{\pm 0.001}$	$0.062_{\pm 0.001}$	$0.04_{\pm 0.001}$	–
CoDi	$0.208_{\pm 0.000}$	$0.025_{\pm 0.001}$	$0.11_{\pm 0.000}$	$0.066_{\pm 0.000}$	$0.025_{\pm 0.001}$	$0.076_{\pm 0.005}$	$0.076_{\pm 0.002}$	0.084
TabSyn	$0.011_{\pm 0.001}$	$0.009_{\pm 0.000}$	$0.013_{\pm 0.001}$	$0.02_{\pm 0.000}$	$0.004_{\pm 0.001}$	$0.078_{\pm 0.003}$	$0.015_{\pm 0.001}$	0.021
TabDiff	$0.012_{\pm 0.000}$	$0.007_{\pm 0.001}$	$0.012_{\pm 0.001}$	$0.017_{\pm 0.000}$	$0.006_{\pm 0.001}$	$0.08_{\pm 0.002}$	$0.016_{\pm 0.001}$	0.021
Fully Factorized	$0.282_{\pm 0.000}$	$0.209_{\pm 0.000}$	$0.308_{\pm 0.000}$	$0.092_{\pm 0.000}$	$0.232_{\pm 0.000}$	$0.264_{\pm 0.000}$	$0.14_{\pm 0.000}$	0.218
Fully Factorized (P)	$0.281_{\pm 0.000}$	$0.208_{\pm 0.000}$	$0.306_{\pm 0.000}$	$0.092_{\pm 0.000}$	$0.232_{\pm 0.000}$	$0.258_{\pm 0.000}$	$0.134_{\pm 0.000}$	0.216
ShallowMixture	$0.05_{\pm 0.001}$	$0.03_{\pm 0.001}$	$0.044_{\pm 0.001}$	$0.064_{\pm 0.001}$	$0.055_{\pm 0.001}$	$0.178_{\pm 0.001}$	$0.068_{\pm 0.001}$	0.070
TabPC	$0.002_{\pm 0.000}$	$0.007_{\pm 0.000}$	$0.01_{\pm 0.001}$	$0.01_{\pm 0.002}$	$0.005_{\pm 0.001}$	$0.062_{\pm 0.002}$	$0.007_{\pm 0.001}$	0.015

LR-C2ST. Using a logistic classifier as a discriminator for computing C2ST is simple and lightweight, but we provided empirical evidence it is also very easy to fool. In fact, it can be shown that it is possible to achieve a perfect detection score by simply generating synthetic data where each column has the same mean as the real data.

Proposition 5.4.1. Let D_r and D_s denote respectively the real and synthetic datasets, having the same number of records n. Let D be the dataset of samples $(\mathbf{x}, y)$ obtained by combining D_r and D_s, where $\mathbf{x}$ is the record and label $y = 0$ if the record is synthetic otherwise $y = 1$. Let $\log p_{\mathbf{w}}(y \mid \mathbf{x})$ be a logistic regression classifier defined as:

$$p_{\mathbf{w}}(y \mid \mathbf{x}) = \text{Bernoulli}(y \mid \pi = \sigma(\mathbf{w}^{\intercal}\mathbf{x})) \tag{5.14}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function and $\mathbf{w}$ are the learnable parameters.

Then, if the empirical means of each feature in D_r and D_s are equal, i.e., $\frac{1}{n}\sum_{(\mathbf{x},y)\in D_r} \mathbf{x} =$

$\frac{1}{n}\sum_{(\mathbf{x},y)\in D_s}\mathbf{x}$, the data log-likelihood:

$$\mathcal{L}(D,\mathbf{w}) = \sum_{(\boldsymbol{x},y)\in D} \log p_{\mathbf{w}}(y \mid \mathbf{x}) \tag{5.15}$$

is maximized by weights $\boldsymbol{w} = \mathbf{0}$, i.e., when $p_{\mathbf{w}}(y \mid \mathbf{x})$ is the random classifier, which assigns probability 0.5 to any point $\boldsymbol{x}$.

Proof. First, let us rewrite the data log-likelihood by decomposing it into the contributions of real and synthetic data:

$$\begin{aligned}
\mathcal{L}(D,\mathbf{w}) &= \sum_{(\boldsymbol{x},y)\in D} \log p_{\mathbf{w}}(y \mid \mathbf{x}) && (5.16)\\
&= \sum_{\boldsymbol{x}\in D_r} \log p_{\mathbf{w}}(1 \mid \mathbf{x}) + \sum_{\boldsymbol{x}\in D_s} \log p_{\mathbf{w}}(0 \mid \mathbf{x}) && (5.17)\\
&= \sum_{\boldsymbol{x}\in D_r} \log \sigma(\mathbf{w}^\intercal\mathbf{x}) + \sum_{\boldsymbol{x}\in D_s} \log(1-\sigma(\mathbf{w}^\intercal\mathbf{x})) && (5.18)
\end{aligned}$$

Since optimizing the parameters of a logistic regression model is a convex optimization problem, we can find the unique global maximum by setting the gradient of the log-likelihood to zero:

$$\nabla_{\mathbf{w}}\mathcal{L}(D,\mathbf{w}) = \sum_{\boldsymbol{x}\in D_r} (1-\sigma(\mathbf{w}^\intercal\mathbf{x}))\mathbf{x} + \sum_{\boldsymbol{x}\in D_s} -\sigma(\mathbf{w}^\intercal\mathbf{x})\mathbf{x} = 0 \tag{5.19}$$

By setting $\mathbf{w} = \mathbf{0}$, we have $\sigma(\mathbf{w}^\intercal\mathbf{x}) = \frac{1}{2}$ for any $\mathbf{x}$, therefore:

$$\begin{aligned}
\nabla_{\mathbf{w}}\mathcal{L}(D,\mathbf{0}) &= \sum_{\boldsymbol{x}\in D_r} (1-0.5)\mathbf{x} + \sum_{\boldsymbol{x}\in D_s} -0.5\mathbf{x} && (5.20)\\
&= \frac{1}{2}\left(\sum_{\boldsymbol{x}\in D_r}\mathbf{x} - \sum_{\boldsymbol{x}\in D_s}\mathbf{x}\right) = 0 && (5.21)\\
&\iff \frac{1}{n}\sum_{\boldsymbol{x}\in D_r}\mathbf{x} = \frac{1}{n}\sum_{\boldsymbol{x}\in D_s}\mathbf{x} && (5.22)
\end{aligned}$$

This means that when the empirical means of each feature in D_r and D_s are equal, the gradient of the log-likelihood is zero at $\mathbf{w} = \mathbf{0}$. Then, since the log-likelihood is concave, this point is the unique global maximum. □

We can conclude that, when a model is able to generate synthetic data with the same per-feature means as the real data, the logistic regression classifier is not able to distinguish between real and synthetic data, achieving an accuracy of 0.5 that corresponds to a perfect LR-C2ST score of 1.0.

This result highlights the insufficiency of the logistic regression C2ST score, as it reduces to a measure of similarity of per-feature means between real and synthetic data. This metric has been recently used for evaluating the diffusion-based generative models TabSyn (Zhang et al., 2024), TabDiff (Shi et al., 2025), and TabbyFlow Guzmán-Cordero et al. (2025). More progressive

approaches can be found in Borisov et al. (2023), where they adopted a Random Forest classifier (Liaw et al., 2002), and Liu et al. (2023a), where they reported an average of three different classifiers (MLP, XGB, and GMM).

Therefore, we propose replacing the logistic regression classifier with a more powerful discriminator based on XGBoost (Chen and Guestrin, 2016), a gradient-boosted decision tree model (Friedman, 2001) that has been shown to be very effective for tabular data classification tasks (Grinsztajn et al., 2022). We use the implementation of sklearn's XGBClassifier with default hyperparameters (Chen and Guestrin, 2016). Despite being more powerful, XGB-C2ST is relatively fast to evaluate.

From the left panel of Figure 5.9 and Table 5.7 we can see that the XGBoost discriminator is able to correctly discern synthetic data generated by the fully factorized model from real samples with perfect accuracy, leading to an almost null C2ST score. Furthermore, the XGBoost-based C2ST is not saturated by recent state-of-the-art models, and it is able to meaningfully distinguish between different methods. Thus, we recommend using XGBoost as a discriminator for computing C2ST in tabular data generation tasks. Nevertheless, we acknowledge that more powerful discriminators may achieve near perfect discrimination for all available synthetic data generation models.

Table 5.7: **Detection** score (C2ST) using XGBoost classifier (higher is better).

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.098_{\pm 0.003}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	0.014
TVAE	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.002_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.32_{\pm 0.005}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	0.046
GReaT	–	$0.603_{\pm 0.003}$	$0.244_{\pm 0.003}$	–	$0.272_{\pm 0.003}$	–	$0.295_{\pm 0.004}$	–
STaSy	$0.38_{\pm 0.004}$	$0.388_{\pm 0.002}$	$0.354_{\pm 0.003}$	–	$0.544_{\pm 0.005}$	$0.203_{\pm 0.003}$	$0.142_{\pm 0.003}$	–
CoDi	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.001_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.408_{\pm 0.004}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	0.059
TabSyn	$0.823_{\pm 0.008}$	$0.471_{\pm 0.005}$	$0.434_{\pm 0.007}$	$0.535_{\pm 0.004}$	$0.772_{\pm 0.008}$	$0.026_{\pm 0.001}$	$0.676_{\pm 0.008}$	0.534
TabDiff	$0.841_{\pm 0.008}$	$0.766_{\pm 0.006}$	$0.583_{\pm 0.009}$	$0.703_{\pm 0.008}$	$0.764_{\pm 0.009}$	$0.034_{\pm 0.002}$	$0.724_{\pm 0.012}$	0.631
Fully Factorized	$0.0_{\pm 0.000}$	$0.001_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.047_{\pm 0.001}$	$0.002_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	0.007
Fully Factorized (P)	$0.016_{\pm 0.000}$	$0.017_{\pm 0.001}$	$0.0_{\pm 0.000}$	$0.05_{\pm 0.001}$	$0.005_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.022_{\pm 0.001}$	0.016
ShallowMixture	$0.823_{\pm 0.008}$	$0.733_{\pm 0.006}$	$0.098_{\pm 0.004}$	$0.156_{\pm 0.003}$	$0.633_{\pm 0.009}$	$0.0_{\pm 0.000}$	$0.297_{\pm 0.007}$	0.391
TabPC	$0.857_{\pm 0.006}$	$0.532_{\pm 0.006}$	$0.262_{\pm 0.004}$	$0.756_{\pm 0.007}$	$0.858_{\pm 0.008}$	$0.088_{\pm 0.004}$	$0.894_{\pm 0.012}$	0.607

5.5 TabPC: Experimental Results

We have shown that a simple fully factorized model, when combined with appropriate preprocessing steps, can achieve competitive scores on some commonly used tabular data generation metrics, but fails to model dependencies between features, as highlighted by the proposed wNMIE metric, XGB-C2ST, and utility scores (MLE). To go beyond the limitations of fully factorized models, we explore the use of probabilistic circuits (PCs) (Vergari et al., 2021) as generative models for tabular data. As anticipated, we evaluate not only TabPC as a tabular data generator but also compare it to a shallow mixture model in order to gain better understanding of the advantages of hierarchical mixtures over shallow mixtures. We trained our probabilistic circuits on the same datasets and with the same preprocessing steps as the baselines, and we report the results in terms of the metrics described above. These results are shown in Figures 5.7, 5.8, and 5.9, with greater detail provided in Tables 5.2, 5.3, 5.4, 5.5, 5.6, and 5.7.

We begin by analyzing the performance of the Shallow Mixture model. On standard fidelity metrics (Shape, Trend, and LR-C2ST), the shallow mixture achieves competitive performance

comparable to most deep learning baselines, though it remains inferior to state-of-the-art diffusion models such as TabDiff and TabSyn. Similarly, while the wNMIE and XGB-C2ST scores show improvement over the fully factorized baseline, they still lag behind top-performing diffusion methods. The MLE scores, while not matching the best models, remain comparable to other deep learning approaches. Overall, these results demonstrate that it is possible to achieve competitive performance using mixture models.

Turning to our deep hierarchical model, TabPC consistently matches or approaches state-of-the-art performance across multiple metrics. On conventional fidelity metrics and wNMIE, TabPC achieves near-perfect scores comparable to the best methods. MLE scores are typically on par with or slightly below the leading models. The XGB-C2ST metric better highlights the differences between methods. For four of the datasets, the performance is slightly better than the current state of the art; for the News dataset, it is comparable to other models. Interestingly, for the News dataset, the XGB-C2ST score is not useful for effectively distinguishing between different generative models. In this case, the difficulty in generating realistic synthetic data is probably due to the high number of features and the complexity of their inter-dependencies. For the remaining two datasets, our model does not achieve state-of-the-art performance.

However, we were able to train our model in significantly less time, as shown in Table 5.8. The generation time for a number of rows equal to the training set is also lower than that of the best diffusion models available. TabPC achieved average speedups of approximately $10\times$ over TabDiff and $2\times$ over TabSyn. For TabPC memory usage was the bottleneck, requiring us to sample in smaller batches, which increased the overall generation time. Our models based on deep probabilistic circuits, however, required a large number of parameters (on the order of hundreds of millions) to be competitive with the state of the art, as shown in Table 5.10. Specifically, the number of parameters is roughly proportional to both the number of features in each data point and the number of units K, which is the main hyperparameter of our model. We present in Figure 5.10 an overview of model performance in terms of XGB-C2ST score, training time, and number of parameters.

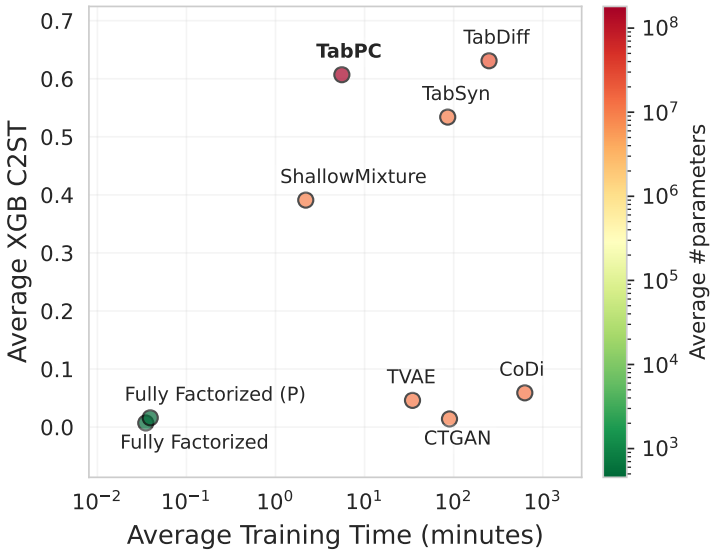


Figure 5.10: Comparison of different models in terms of XGB-C2ST score, training time, and number of parameters. The measures are averaged across all datasets and runs.

Table 5.8: Training times (seconds).

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	3978.0	4214.0	3640.0	13825.0	1881.0	8693.0	1454.0	5383.571
TVAE	1379.0	1573.0	1694.0	4055.0	797.0	4355.0	634.0	2069.571
GReaT	16048.0	13669.0	25673.0	–	6711.0	–	6761.0	–
STaSy	6353.0	6562.0	7687.0	–	4834.0	7162.0	4205.0	–
CoDi	25634.0	27532.0	20341.0	151953.0	9750.0	20607.0	8196.0	37716.143
TabSyn	4431.0	4179.0	5225.0	9618.0	3427.0	5382.0	3737.0	5143.0
TabDiff	11365.0	14064.0	14189.0	30289.0	10515.0	16842.0	7452.0	14959.429
Fully Factorized	1.708	1.238	2.506	1.992	0.611	5.96	0.664	2.097
Fully Factorized (P)	1.32	1.634	2.986	2.323	0.534	7.179	0.494	2.353
ShallowMixture	61.327	115.925	164.478	311.255	18.246	196.379	49.944	131.079
TabPC	391.677	268.667	198.679	566.774	37.706	806.108	62.8	333.202

Table 5.9: Sampling times (seconds). The time refers to generating a dataset of the same size as the training set. For TabPC and Shallow Mixture, the sampling time depends on the batch size, which was adapted with the purpose of maximizing GPU utilization.

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	$1.559_{\pm 0.056}$	$1.975_{\pm 0.056}$	$2.171_{\pm 0.068}$	$6.18_{\pm 0.263}$	$0.917_{\pm 0.03}$	$4.03_{\pm 0.437}$	$0.855_{\pm 0.032}$	2.527
TVAE	$0.948_{\pm 0.028}$	$1.36_{\pm 0.039}$	$1.589_{\pm 0.03}$	$4.259_{\pm 0.152}$	$0.577_{\pm 0.027}$	$2.973_{\pm 0.073}$	$0.572_{\pm 0.015}$	1.754
GReaT	$343.95_{\pm 4.762}$	$230.25_{\pm 1.713}$	$529.9_{\pm 1.774}$	–	$101.25_{\pm 0.444}$	–	$104.5_{\pm 0.607}$	–
STaSy	$14.765_{\pm 10.594}$	$13.43_{\pm 1.6}$	$11.43_{\pm 2.772}$	–	$5.583_{\pm 2.108}$	$13.9_{\pm 3.275}$	$15.325_{\pm 37.41}$	–
CoDi	$9.85_{\pm 0.592}$	$10.356_{\pm 0.477}$	$8.013_{\pm 0.527}$	$652.005_{\pm 7.312}$	$4.086_{\pm 0.233}$	$9.442_{\pm 0.58}$	$3.628_{\pm 0.268}$	99.626
TabDDPM	$72.08_{\pm 2.903}$	$67.241_{\pm 1.734}$	$54.917_{\pm 2.154}$	$278.74_{\pm 5.711}$	$31.821_{\pm 0.76}$	$64.059_{\pm 1.371}$	$35.003_{\pm 1.354}$	86.266
TabSyn	$5.474_{\pm 0.365}$	$5.835_{\pm 0.31}$	$5.751_{\pm 0.532}$	$16.9_{\pm 1.726}$	$2.861_{\pm 0.193}$	$12.272_{\pm 1.43}$	$2.233_{\pm 0.185}$	7.332
TabDiff	$11.141_{\pm 0.442}$	$8.791_{\pm 0.315}$	$9.346_{\pm 0.231}$	$195.646_{\pm 9.661}$	$3.83_{\pm 0.119}$	$15.724_{\pm 0.178}$	$5.577_{\pm 0.16}$	35.722
Fully Factorized	$0.623_{\pm 0.018}$	$0.668_{\pm 0.004}$	$1.135_{\pm 0.005}$	$1.455_{\pm 0.016}$	$0.293_{\pm 0.002}$	$3.512_{\pm 0.095}$	$0.302_{\pm 0.002}$	1.141
Fully Factorized (P)	$0.618_{\pm 0.008}$	$0.647_{\pm 0.01}$	$1.187_{\pm 0.021}$	$1.407_{\pm 0.01}$	$0.296_{\pm 0.002}$	$3.787_{\pm 0.014}$	$0.293_{\pm 0.002}$	1.176
ShallowMixture	$4.245_{\pm 0.14}$	$8.881_{\pm 0.041}$	$5.902_{\pm 0.957}$	$17.672_{\pm 0.144}$	$1.684_{\pm 0.742}$	$10.94_{\pm 0.179}$	$2.937_{\pm 0.718}$	7.466
TabPC	$3.159_{\pm 0.038}$	$1.483_{\pm 0.004}$	$2.839_{\pm 0.014}$	$7.875_{\pm 0.128}$	$0.605_{\pm 0.003}$	$6.568_{\pm 0.012}$	$0.869_{\pm 0.01}$	3.342

Table 5.10: Number of model parameters.

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	9.60×10^6	9.58×10^6	9.66×10^6	1.18×10^7	9.56×10^6	9.89×10^6	9.60×10^6	9.96×10^6
TVAE	1.07×10^7	1.06×10^7	1.07×10^7	1.28×10^7	1.06×10^7	1.09×10^7	1.07×10^7	1.10×10^7
GReaT	1.21×10^8	1.21×10^8	1.21×10^8	–	1.21×10^8	–	1.21×10^8	–
STaSy	4.26×10^7	4.15×10^7	4.19×10^7	–	3.85×10^7	4.05×10^7	4.12×10^7	–
CoDi	1.19×10^7	1.19×10^7	1.19×10^7	1.19×10^7	1.19×10^7	1.19×10^7	1.19×10^7	1.19×10^7
TabDDPM	1.18×10^7	1.17×10^7	1.17×10^7	1.61×10^7	1.16×10^7	1.17×10^7	1.17×10^7	1.23×10^7
TabSyn	1.06×10^7	1.06×10^7	1.07×10^7	1.08×10^7	1.06×10^7	1.09×10^7	1.07×10^7	1.07×10^7
TabDiff	2.12×10^7	2.12×10^7	2.14×10^7	2.16×10^7	2.12×10^7	2.18×10^7	2.13×10^7	2.14×10^7
Fully Factorized	1.30×10^2	1.51×10^2	1.07×10^2	2.35×10^3	2.20×10^1	3.00×10^2	1.31×10^2	4.55×10^2
Fully Factorized (P)	1.34×10^2	1.53×10^2	1.19×10^2	2.35×10^3	2.40×10^1	3.60×10^2	1.45×10^2	4.69×10^2
ShallowMixture	2.70×10^6	7.70×10^6	2.40×10^6	4.69×10^7	2.50×10^5	3.61×10^6	2.92×10^6	9.50×10^6
TabPC	3.31×10^8	7.63×10^7	1.96×10^8	3.43×10^8	6.00×10^7	1.16×10^8	1.44×10^8	1.81×10^8

Finally, we check for privacy leaks using a metric based on the Distance to Closest Record (DCR). Let T_{train} be the training set, S be the synthetic dataset generated by a model, and T_{test} be the test set, and let $\text{DCR}(\mathbf{x}, D)$ be the distance of a record $\mathbf{x}$ to its closest record in the dataset D. In many recent works, the evaluation protocol consists in computing the fraction of times that the DCR of a synthetic record to the training set is lower than the DCR to the test set:

$$\text{DCR Score} = \frac{1}{|S|} \sum_{\mathbf{s} \in S} \mathbb{I}\left[\text{DCR}(\mathbf{s}, T_{train}) < \text{DCR}(\mathbf{s}, T_{test})\right] \tag{5.23}$$

Table 5.11: DCR scores (%). We computed the DCR scores according to the method described in Section 2.5. In this table we report the percentage of times the DCR of a synthetic data sample with respect to the training set was lower than the 2% quantile of the empirical distribution of DCRs of the test set. Values significantly higher than 2% suggest potential privacy leaks. Conversely, values lower than 2% indicate that synthetic samples are farther from training records than test records, which may signal underfitting.

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	$0.01_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.03_{\pm 0.01}$	$0.0_{\pm 0.000}$	$0.09_{\pm 0.02}$	$0.0_{\pm 0.000}$	0.0186
TVAE	$6.41_{\pm 0.11}$	$0.0_{\pm 0.000}$	$0.06_{\pm 0.01}$	$19.55_{\pm 0.17}$	$0.01_{\pm 0.01}$	$0.22_{\pm 0.02}$	$0.02_{\pm 0.01}$	3.7529
GReaT	$0.0_{\pm 0.000}$	$1.49_{\pm 0.06}$	$12.54_{\pm 0.17}$	–	$0.35_{\pm 0.04}$	–	$10.8_{\pm 0.35}$	–
STaSy	$1.1_{\pm 0.04}$	$0.44_{\pm 0.04}$	$0.56_{\pm 0.06}$	–	$0.28_{\pm 0.04}$	$0.25_{\pm 0.03}$	$0.27_{\pm 0.05}$	–
CoDi	$0.06_{\pm 0.01}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.03_{\pm 0.01}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	0.0129
TabSyn	$1.72_{\pm 0.07}$	$0.51_{\pm 0.05}$	$1.12_{\pm 0.05}$	$0.65_{\pm 0.02}$	$0.07_{\pm 0.02}$	$0.81_{\pm 0.05}$	$1.76_{\pm 0.12}$	0.9486
TabDiff	$1.85_{\pm 0.07}$	$0.61_{\pm 0.05}$	$1.4_{\pm 0.06}$	$1.35_{\pm 0.05}$	$0.05_{\pm 0.02}$	$0.89_{\pm 0.05}$	$2.24_{\pm 0.09}$	1.1986
Fully Factorized	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.01_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	0.0014
Fully Factorized (P)	$0.04_{\pm 0.01}$	$0.08_{\pm 0.01}$	$0.0_{\pm 0.000}$	$0.01_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	$0.0_{\pm 0.000}$	0.0186
ShallowMixture	$2.217_{\pm 0.064}$	$0.923_{\pm 0.06}$	$0.708_{\pm 0.055}$	$0.089_{\pm 0.009}$	$0.505_{\pm 0.068}$	$0.07_{\pm 0.01}$	$1.311_{\pm 0.1}$	0.832
TabPC	$1.84_{\pm 0.09}$	$0.51_{\pm 0.04}$	$0.92_{\pm 0.06}$	$1.17_{\pm 0.05}$	$0.14_{\pm 0.03}$	$0.68_{\pm 0.04}$	$2.52_{\pm 0.15}$	1.1114

The idea is that if synthetic records are closer to training records more than to test records, then it means that the model is overfitting the training data, and thus there is a risk of privacy leaks. Then a lower DCR score is better, with a value of 0.5 indicating that there is no overfitting. DCR has already been criticized in Yao et al. (2025), however, alternatives are often more complex to compute. Another relevant limitation is that it requires the training set T_{train} to be of the same size as the test set T_{test}. Indeed, in recent works this led to the choice of training a separate model specifically for evaluating DCR, which we consider not ideal. We propose instead to use the evaluation protocol described in Section 2.5, which can be followed using the same model assessed by the other metrics. Results are reported in Table 5.11, showing that we do not observe any privacy leak.

We also provide in the appendix results relative to the α-Precision and β-Recall (Alaa et al., 2022) (Table C.2 and Table C.3) metrics, which are also widely used in tabular data generation literature.

Overall, results show that deep probabilistic circuits can achieve performances in line with the state of the art for tabular data generation, according to several metrics. Moreover, they seem to have an advantage in terms of efficiency of training and sampling time compared to diffusion models. This is relevant as circuits are tractable models, as opposed to all the other baselines and in particular the state-of-the-art diffusion models.

Note that tractability is not compromised by the preprocessing steps, as quantile normalization is an invertible and differentiable function applied per-feature. This means that for density evaluation, it is sufficient to evaluate the density of the preprocessed data and multiply it by the Jacobian determinant of the quantile transformation, which is straightforward as it is a diagonal matrix. Similarly, sampling can be done by first sampling from the model and then applying the inverse transformation.

5.6 The Advantages of Tractable Models for Tabular Data

Probabilistic circuits are tractable, in the sense that they allow tractable exact computation of several probabilistic queries. In particular, they allow exact sampling, density computation, and marginalization by computing integrals of the density with respect to the input variables. In contrast, most expressive generative models that can sample often do not allow for the exact computation of any other probabilistic query. For example, VAEs and GANs only allow for sampling, and in diffusion-based models, the density can only be approximated using the instantaneous change of variables formula. Autoregressive and discrete normalizing flow models instead allow for both sampling and exact density computation.

In the context of tabular data, exact probabilistic queries have many use cases:

- **Handling missing values**: The computation of density when a data point shows missing values can be accomplished by marginalizing over the missing variables. This allows training in the presence of missing data. Moreover, we exploited marginalization to handle inflated values as previously described.
- **Conditional sampling**: Sampling from the distribution of a subset of variables given evidence on other variables (often referred to as inpainting) can be done exactly with probabilistic circuits. In contrast, inpainting requires specific training for most deep generative models, or can only be approximated (Song et al., 2021).
- **Exact Density Computation and Marginalization**: Probabilistic circuits allow for the exact computation of densities and marginals, which can be useful for comparing models, anomaly detection (Liu et al., 2022a), fairness evaluation (Varley and Belle, 2021), and other probabilistic queries.

5.7 Why do Circuits Perform Well on Tabular Data?

The experimental results prove that deep probabilistic circuits reach state-of-the-art for tabular data generation. This is different with respect to images and text, where deep probabilistic circuits still struggle in obtaining acceptable sample quality. One possible explanation is that circuits cannot scale to significantly larger dimensionalities. Moreover, they probably struggle in modeling large sets of highly correlated variables such as the pixels of an image. This is also suggested by the low performance in terms of XGB-C2ST achieved in the Default and News datasets, which are either characterized by highly correlated variables (Default) or a large number of numerical features (News). Tabular data instead is usually characterized by limited dimensionality, and often the number of strongly correlated variables is limited. The hierarchical structure of the region graph based on the Chow-Liu tree plays a fundamental role in exploiting independencies among variables, and focusing on the sparse but relevant connections between features. Finally, mixtures can handle the highly discontinuous and non-linear relationships that characterize tabular data. The reasons for the effectiveness on tabular data are similar to those that hold for decision trees, which are commonly considered the best models for predictive tasks on tabular data. In fact, deep probabilistic circuits can be seen as the generative version of decision trees (Correia et al., 2020, Ventola et al., 2021).

5.8 Limitations

Metric saturation and interpretability. While wNMIE better targets dependency modeling than Trend, it can still saturate (in this case, reaching an almost null value) when strong models reproduce mutual information well. In general we observe that simple metrics are therefore prone to saturation, and only more metrics based on expressive machine learning models consistently differentiate methods. Conversely, XGB-C2ST can fail to distinguish between models on the lower end of performances, offering limited interpretability.

Benchmark coverage. Our conclusions are based on a commonly used but limited set of tabular datasets. It remains unclear how these metrics and models behave on even larger, and more complex datasets. Expanding benchmarks to verify the failure modes of current models is an important next step.

Model expressiveness and parameter efficiency. TabPC struggles on datasets with many features or strong correlations, suggesting that mixture-based structures and current region graphs may be insufficient for certain dependency patterns. Understanding the motivation is challenging given the limited interpretability of discriminator-based scores. Moreover, TabPC requires a larger number of parameters to be competitive with the state of the art, leading to high memory usage and challenges when scaling to feature-rich datasets. Improving parameter efficiency remains an open problem, likely connected to the challange of improving the expressiveness of the model.

5.9 Discussion

In this chapter, we presented a critical reassessment of tabular data generation through the lens of probabilistic circuits. We exposed critical limitations in commonly used evaluation metrics: standard benchmarks easily saturate on simple models that fail to capture inter-feature dependencies, masking poor distributional fidelity. To address this, we proposed more reliable alternatives: weighted normalized mutual information estimate (wNMIE) to better detect dependency modeling, and XGBoost-based detection (XGB-C2ST) for a more powerful evaluation of data realism. We demonstrated that deep probabilistic circuits are competitive with state-of-the-art diffusion-based methods on tabular benchmarks, while offering distinct advantages: tractable exact inference, efficient sampling, and support for principled probabilistic queries (marginalization, conditioning) that most black-box deep generative models cannot provide. These properties make PCs valuable for domains requiring interpretability, missing-data handling, or formal probabilistic reasoning. However, our approach has limitations. Achieving competitive performance requires substantial overparameterization, quantile preprocessing of numerical features remains necessary, and performance degrades on highly correlated variables and high-dimensional numerical spaces, highlighting the limits of current PC structures for complex dependencies.

Future directions include exploring richer PC architectures (Probabilistic Integral Circuits (Gala et al., 2024), Sum-of-Squares Circuits (Loconte et al., 2025b), or using multivariate Gaussian input units (Pevnỳ et al., 2020)), moving beyond Chow-Liu tree region graphs toward more general directed acyclic graphs (DAGs) or using Expectation Maximization in the learning process instead of maximum likelihood. These advances may extend PC applicability beyond the moderate-dimensional, sparse-correlation regime where they currently excel.

Chapter 6

Conclusion

Generative models for tabular data have significantly advanced during the last years, benefiting from the developments in deep generative models techniques. In particular, state-of-the-art techniques for tabular data generation are often based on diffusion-like models, adapted to handle heterogeneous data types (numerical and categorical) and complex, multimodal marginals. These models can guarantee high fidelity of the synthesized data in terms of quality of marginals and reproduced correlations, while offering protection against privacy leaks. On the other hand, they have shown limited flexibility and capabilities outside unconditional generation. Conditional sampling under user-defined logical constraints is non-trivial, and often specific training is needed. Diffusion models as well as other deep generative models are not tractable, in the sense that they do not allow for exact computation of some relevant probabilistic queries, as exact density computation, marginalization and sampling conditionally on any evidence. Finally, the generation of multiple related tables, also known as relational data generation or multi-table generation, is still not mature as a field, as state-of-the-art techniques are either not flexible enough to handle any kind of relational structure or are based on implicit assumptions limiting the expressiveness of the model.

In this dissertation we discussed these major limitations in tabular data generation, and studied solutions to overcome these and their limits.

In Chapter 2 we give the necessary background on generative models that are the foundation of the following discussions.

In Chapter 3, we introduce a method for performing conditional sampling under user-defined logical constraints from pretrained score-based models. Combining a neuro-symbolic language to encode differentiable logical constraints to a technique for approximating the conditional score, we show we are able to better approximate conditional distributions on a variety of tasks and types of data (tabular, images, time series), compared to a state-of-the-art approach. However, our approach shows limits when applied to very complex constraints and data is highly dimensional. We discuss then when it is preferable to apply other methods that may not guarantee unbiased approximation of conditional distributions but optimize the quality of samples.

In Chapter 4 we elaborate a generative model based on flow matching for generating relational data, conditioning on the graph representing the structure of the relational database. Our approach does not assume any particular structure of the graph, or independence between any two records that are directly or indirectly connected in a relational database, making it more expressive and flexible than previous methods. Empirical results show that we are able to generate better data in terms of fidelity, with some evidence on privacy guarantees. Future works will fo-

cus on the combination of more sophisticated methods for graph generation, better engineering of the neural network backbone, and improved scalability of the training scheme.

Finally, in Chapter 5, we explore the use of overparametrized probabilistic circuits as generative models for tabular data. Using widely known metrics, datasets and recent techniques as benchmarks, we achieve competitive results with the state-of-the-art while retaining tractability, that allows better handling of missing values, exact likelihood computation and exact conditional sampling. Our models are also characterized by a significantly faster training time and a faster sampling with respect to diffusion models. Finally, we show the limits of some of commonly used metrics and propose better alternatives. Future work will involve the exploration of more expressive variants of probabilistic circuits, as these show some scalability issues and limits when data is characterized by many highly correlated features.

Bibliography

Alekh Agarwal, Sham Kakade, and Lin F. Yang. Model-based reinforcement learning with a generative model is minimax optimal. In Jacob Abernethy and Shivani Agarwal, editors, *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 67–83. PMLR, 09–12 Jul 2020. URL https://proceedings.mlr.press/v125/agarwal20b.html.

Kareem Ahmed, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck, and Antonio Vergari. Semantic probabilistic layers for neuro-symbolic learning. *Advances in Neural Information Processing Systems*, 35:29944–29959, 2022.

Ahmed Alaa, Boris Van Breugel, Evgeny S Saveliev, and Mihaela van der Schaar. How faithful is your synthetic data? sample-level metrics for evaluating and auditing generative models. In *International Conference on Machine Learning*, pages 290–306. PMLR, 2022.

Brian D O Anderson. Reverse-time diffusion equation models. *Stochastic Process. Appl.*, 12(3): 313–326, May 1982.

Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

Samuel A. Assefa, Danial Dervovic, Mahmoud Mahfouz, Robert E. Tillman, Prashant Reddy, and Manuela Veloso. Generating synthetic data in finance: Opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance*, ICAIF '20. Association for Computing Machinery, 2021. ISBN 9781450375849.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 17981–17993. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/958c530554f78bcd8e97125b70e6973d-Paper.pdf.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Samy Badreddine, Artur S. d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *CoRR*, abs/2012.13635, 2020. URL https://arxiv.org/abs/2012.13635.

Samy Badreddine, Luciano Serafini, and Michael Spranger. logltn: Differentiable fuzzy logic in the logarithm space. *ArXiv*, abs/2306.14546, 2023. URL https://api.semanticscholar.org/CorpusID:259251594.

Arpit Bansal, Hong-Min Chu, Avi Schwarzschild, Soumyadip Sengupta, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Universal guidance for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 843–852, 2023.

Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018. URL https://arxiv.org/pdf/1806.01261.pdf.

Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

Jens Behrmann, Will Grathwohl, Ricky T.Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, 2019.

Hendrik Blockeel, Sašo Džeroski, Boris Kompare, Stefan Kramer, Bernhard Pfahringer, and WIM VAN LAER. Experiments in predicting biodegradability. *Applied Artificial Intelligence*, 18(2):157–181, 2004.

R. Bock. MAGIC Gamma Telescope. UCI Machine Learning Repository, 2004. DOI: https://doi.org/10.24432/C52C8B.

Vadim Borisov, Kathrin Sessler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. Language models are realistic tabular data generators. In *The Eleventh International Conference on Learning Representations*, 2023.

Asma Azizi Boroojeni, Jeremy Dewar, Tong Wu, and James M Hyman. Generating bipartite networks with a prescribed joint degree distribution. *Journal of complex networks*, 5(6):839–857, 2017.

Luca Bortolussi, Francesca Cairoli, Francesco Giacomarra, and Davide Scassola. Model abstraction and conditional sampling with score-based diffusion models. In *International Conference on Quantitative Evaluation of Systems*, pages 307–310. Springer, 2023.

Alexander Boudewijn, Andrea Filippo Ferraris, Daniele Panfilo, Vanessa Cocca, Sabrina Zinutti, Karel De Schepper, and Carlo Rossi Chauvenet. Privacy measurement in tabular synthetic data: State of the art and future research directions. *arXiv preprint arXiv:2311.17453*, 2023.

Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*, pages 10–21, 2016.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Tom Brown et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020.

Clement Chadebec, Onur Tasar, Eyal Benaroche, and Benjamin Aubin. Flash diffusion: Accelerating any conditional diffusion model for few steps image generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(15):15686–15695, 2025.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

RJ Chen et al. Synthetic data in healthcare: A review. *arXiv preprint arXiv:2107.06499*, 2021.

Song Chen. Beijing PM2.5. UCI Machine Learning Repository, 2015. DOI: https://doi.org/10.24432/C5JS49.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *arXiv preprint arXiv:1611.02731*, 2016.

Myung Jin Choi, Vincent YF Tan, Animashree Anandkumar, and Alan S Willsky. Learning latent tree graphical models. *The Journal of Machine Learning Research*, 12:1771–1812, 2011.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. Technical report, University of California, Los Angeles (UCLA), 2020.

CKCN Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

Hyungjin Chung and Jong Chul Ye. Score-based diffusion models for accelerated mri. *Medical image analysis*, 80:102479, 2022.

Federico Cornalba, Constantin Disselkamp, Davide Scassola, and Christopher Helf. Multi-objective reward generalization: improving performance of deep reinforcement learning for applications in single-asset trading. *Neural Computing and Applications*, 36(2):619–637, 2024.

Alvaro Correia, Robert Peharz, and Cassio P de Campos. Joints in random forests. *Advances in neural information processing systems*, 33:11404–11415, 2020.

Lingxi Cui, Huan Li, Ke Chen, Lidan Shou, and Gang Chen. Tabular data augmentation for machine learning: Progress and prospects of embracing generative ai. *arXiv preprint arXiv:2407.21523*, 2024.

Bin Dai and David Wipf. Diagnosing and enhancing vae models. *arXiv preprint arXiv:1903.05789*, 2019.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

Aaron Dennis and Dan Ventura. Learning the architecture of sum-product networks using clustering on variables. *Advances in Neural Information Processing Systems*, 25, 2012.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *International Conference on Learning Representations*, 2017.

Felix Draxler, Peter Sorrenson, Lea Zimmermann, Armand Rousselot, and Ullrich Köthe. Free-form flows: Make any architecture a normalizing flow. In *International Conference on Artificial Intelligence and Statistics*, pages 2197–2205. PMLR, 2024.

Yilun Du and Igor Mordatch. Implicit generation and modeling with energy-based models. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://papers.nips.cc/paper/2019.

Yilun Du et al. Diffusion policy: Visuomotor policy learning via diffusion models. *arXiv preprint arXiv:2303.04137*, 2023.

Cynthia Dwork. Differential privacy. In *International colloquium on automata, languages, and programming*, pages 1–12. Springer, 2006.

Floor Eijkelboom, Grigory Bartosh, Christian Andersson Naesseth, Max Welling, and Jan-Willem van de Meent. Variational flow matching for graph generation. *Advances in Neural Information Processing Systems*, 37:11735–11764, 2024.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *arXiv preprint arXiv:1702.03118*, 2017.

Jesse Engel, Matthew Hoffman, and Adam Roberts. Latent constraints: Learning to generate conditionally from unconditional generative models. *arXiv preprint arXiv:1711.05772*, 2017.

Kelwin Fernandes, Pedro Vinagre, Paulo Cortez, and Pedro Sernadela. Online News Popularity. UCI Machine Learning Repository, 2015. DOI: https://doi.org/10.24432/C5NS3V.

Joao Fonseca and Fernando Bacao. Tabular and latent space synthetic data generation: a literature review. *Journal of Big Data*, 10(1):115, 2023.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

Gennaro Gala, Cassio de Campos, Robert Peharz, Antonio Vergari, and Erik Quaeghebeur. Probabilistic integral circuits. In *International Conference on Artificial Intelligence and Statistics*, pages 2143–2151. PMLR, 2024.

Kuzman Ganchev, Joao Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *The Journal of Machine Learning Research*, 11:2001–2049, 2010.

PyTorch Geometric. Heterogeneous graph learning, 2024. URL https://pytorch-geometric.readthedocs.io/en/2.6.0/notes/heterogeneous.html. PyG Documentation, version 2.6.0.

Mauro Giuffrè and Dennis L Shung. Harnessing the power of synthetic data in healthcare: innovation, application, and privacy. *NPJ digital medicine*, 6(1):186, 2023.

Aldren Gonzales, Guruprabha Guruswamy, and Scott R Smith. Synthetic data in health care: A narrative review. *PLOS Digital Health*, 2(1):e0000082, 2023.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, pages 2672–2680, 2014.

Alexandros Graikos, Nikolay Malkin, Nebojsa Jojic, and Dimitris Samaras. Diffusion models as plug-and-play priors. *Advances in Neural Information Processing Systems*, 35:14715–14728, 2022.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in Neural Information Processing Systems*, 35:507–520, 2022.

Andreas Grivas, Lorenzo Loconte, Emile van Krieken, Piotr Nawrot, Yu Zhao, Euan Wielewski, Pasquale Minervini, Edoardo Ponti, and Antonio Vergari. Fast and expressive multi-token prediction with probabilistic circuits. *arXiv preprint arXiv:2511.11346*, 2025.

Jiatao Gu, Ying Shen, Tianrong Chen, Laurent Dinh, Yuyang Wang, Miguel Angel Bautista, David Berthelot, Josh Susskind, and Shuangfei Zhai. Starflow-v: End-to-end video generative modeling with normalizing flow. *arXiv preprint arXiv:2511.20462*, 2025.

Mohamed Gueye, Yazid Attabi, and Maxime Dumas. Row conditional-tgan for generating synthetic relational databases. *IEEE ICASSP 2023*, 2022.

Manbir Gulati and Paul Roysdon. Tabmt: Generating tabular data with masked transformers. *Advances in Neural Information Processing Systems*, 36:46245–46254, 2023.

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*, 2016.

Andrés Guzmán-Cordero, Floor Eijkelboom, and Jan-Willem van de Meent. Exponential family variational flow matching for tabular data generation. In *Forty-second International Conference on Machine Learning*, 2025.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Peter J Haas, Paul P Maglio, Patricia G Selinger, and Wang-Chiew Tan. Data is dead... without what-if models. *Proceedings of the VLDB Endowment*, 4(12):1486–1489, 2011.

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

Baran Hashemi and Claudius Krause. Deep generative models for detector signature simulation: A taxonomic review. *Reviews in Physics*, 12:100092, 2024. ISSN 2405-4283. doi: https://doi.org/10.1016/j.revip.2024.100092. URL https://www.sciencedirect.com/science/article/pii/S2405428324000029.

Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. Logan: Membership inference attacks against generative models. *arXiv preprint arXiv:1705.07663*, 2017.

Yutong He, Naoki Murata, Chieh-Hsin Lai, Yuhta Takida, Toshimitsu Uesaka, Dongjun Kim, Wei-Hsiang Liao, Yuki Mitsufuji, J Zico Kolter, Ruslan Salakhutdinov, et al. Manifold preserving guided diffusion. *arXiv preprint arXiv:2311.16424*, 2023.

Mikel Hernandez, Gorka Epelde, Ane Alberdi, Rodrigo Cilla, and Debbie Rankin. Synthetic data generation for tabular health records: A systematic review. *Neurocomputing*, 493:28–45, 2022.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

Zhiting Hu, Zichao Yang, Russ R Salakhutdinov, LIANHUI Qin, Xiaodan Liang, Haoye Dong, and Eric P Xing. Deep generative models with learnable knowledge constraints. *Advances in Neural Information Processing Systems*, 31, 2018.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Simon, and Calvin Hawthorne. Music transformer: Generating music with long-term structure. In *International Conference on Learning Representations (ICLR) Workshop / arXiv*, 2019. arXiv:1809.04281.

Valter Hudovernik. Relational data generation with graph neural networks and latent diffusion models. In *NeurIPS 2024 Third Table Representation Learning Workshop*, 2024.

Valter Hudovernik, Martin Jurkovič, and Erik Štrumbelj. Benchmarking the fidelity and utility of synthetic relational data. *arXiv preprint arXiv:2410.03411*, 2024.

Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.

Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

Fengwei Jia, Hongli Zhu, Fengyuan Jia, Xinyue Ren, Siqi Chen, Hongming Tan, and Wai Kin Victor Chan. A tabular data generation framework guided by downstream tasks optimization. *Scientific Reports*, 14(1):15267, 2024.

William Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning (ICML)*, pages 2323–2332, 2018.

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

Zahra Kadkhodaie and Eero Simoncelli. Stochastic solutions for linear inverse problems using the prior implicit in a denoiser. *Advances in Neural Information Processing Systems*, 34:13242–13254, 2021.

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

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

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

Tero Karras et al. A style-based generator architecture for generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

William Ogilvy Kermack, A. G. McKendrick, and Gilbert Thomas Walker. A contribution to the mathematical theory of epidemics, 1927.

Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. *Computer graphics forum*, 38(2):59–70, 2019.

Jayoung Kim, Chaejeong Lee, and Noseong Park. Stasy: Score-based tabular data synthesis. In *The Eleventh International Conference on Learning Representations*, 2023.

Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, 2018.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Peter E Kloeden and Eckhard Platen. *Numerical solution of stochastic differential equations*, volume 23. Springer Science & Business Media, 2013.

Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. Tabddpm: Modelling tabular data with diffusion models. In *International Conference on Machine Learning*, pages 17564–17579. PMLR, 2023.

T. O. Kvalseth. Entropy and correlation: Some comments. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(3):517–519, 1987.

The APRIL Lab. cirkit, October 2024. URL https://github.com/april-tools/cirkit.

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu-Jie Huang. A tutorial on energy-based learning. Technical report, NYU Courant, 2006. URL: https://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf.

Chaejeong Lee, Jayoung Kim, and Noseong Park. Codi: Co-evolving contrastive diffusion models for mixed-type tabular synthesis. In *International Conference on Machine Learning*, pages 18940–18956. PMLR, 2023.

Jiayu Li and YC Tay. Irg: generating synthetic relational databases using gans. *arXiv preprint arXiv:2312.15187*, 2023.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *AAAI*, 2018.

Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3): 18–22, 2002.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *11th International Conference on Learning Representations, ICLR 2023*, 2023.

Anji Liu, Zilei Shao, and Guy Van den Broeck. Rethinking probabilistic circuit parameter learning. *arXiv preprint arXiv:2505.19982*, 2025.

Boyang Liu, Pang-Ning Tan, and Jiayu Zhou. Unsupervised anomaly detection by robust density estimation. *Proceedings of the AAAI conference on artificial intelligence*, 36(4):4101–4108, 2022a.

Fan Liu, Zhiyong Cheng, Huilin Chen, Yinwei Wei, Liqiang Nie, and Mohan Kankanhalli. Privacy-preserving synthetic data generation for recommendation systems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1379–1389, 2022b.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

Tennison Liu, Zhaozhi Qian, Jeroen Berrevoets, and Mihaela van der Schaar. Goggle: Generative modelling for tabular data by learning relational structure. In *The Eleventh International Conference on Learning Representations*, 2023a.

Xingchao Liu, Lemeng Wu, Shujian Zhang, Chengyue Gong, Wei Ping, and Qiang Liu. Flowgrad: Controlling the output of generative odes with gradients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24335–24344, 2023b.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

Lorenzo Loconte, Antonio Mari, Gennaro Gala, Robert Peharz, Cassio de Campos, Erik Quaeghebeur, Gennaro Vessio, and Antonio Vergari. What is the relationship between tensor factorizations and circuits (and how can we exploit it)? *Transactions on Machine Learning Research (TMLR)*, 2025a. ISSN 2835-8856. Featured Certification.

Lorenzo Loconte, Stefan Mengel, and Antonio Vergari. Sum of squares circuits. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(18):19077–19085, 2025b.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning*, pages 32819–32848, 2024.

Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000.

Ryan McKenna, Daniel Sheldon, and Gerome Miklau. Graphical-model based estimation and inference for differential privacy. In *International Conference on Machine Learning*, pages 4435–4444. PMLR, 2019.

Chenlin Meng, Kristy Choi, Jiaming Song, and Stefano Ermon. Concrete score matching: Generalized score matching for discrete data. *Advances in Neural Information Processing Systems*, 35:34532–34545, 2022.

Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14297–14306, 2023.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. *Interspeech*, 2(3):1045–1048, 2010.

Benjamin Nachman and David Shih. Anomaly detection with density estimation. *Physical Review D*, 101(7):075042, 2020.

Saeid Naderiparizi, Xiaoxuan Liang, Berend Zwartsenberg, and Frank Wood. Constrained generative modeling with manually bridged diffusion models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(18):19607–19615, 2025.

Nithin Gopalakrishnan Nair, Anoop Cherian, Suhas Lohit, Ye Wang, Toshiaki Koike-Akino, Vishal M Patel, and Tim K Marks. Steered diffusion: A generalized framework for plug-and-play conditional image synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 20850–20860, 2023.

Beata Nowok, Gillian M Raab, and Chris Dibben. synthpop: Bespoke creation of synthetic data in r. *Journal of statistical software*, 74:1–26, 2016.

Bernt Øksendal. Stochastic differential equations. In *Stochastic differential equations: an introduction with applications*, pages 38–50. Springer, 2003.

Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. *ICLR*, 2020.

Milton Nicolás Plasencia Palacios, Sebastiano Saccani, Gabriele Sgroi, Alexander Boudewijn, and Luca Bortolussi. Contrastive learning-based privacy metrics in tabular synthetic datasets. *arXiv preprint arXiv:2502.13833*, 2025.

Tianyu Pang, Kun Xu, Chongxuan Li, Yang Song, Stefano Ermon, and Jun Zhu. Efficient learning of generative models via finite-difference score matching. *arXiv preprint arXiv:2007.03317*, 2020.

Wei Pang, Masoumeh Shafieinejad, Lucy Liu, Stephanie Hazlewood, and Xi He. Clavaddpm: Multi-relational data synthesis with cluster-guided diffusion models. *Advances in Neural Information Processing Systems*, 37:83521–83547, 2024.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, 2017.

George Papamakarios, Eric Nalisnick, Danilo J. Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019. URL https://arxiv.org/abs/1912.02762.

Giorgio Parisi. Correlation functions and computer simulations. *Nuclear Physics B*, 180(3):378–384, 1981.

Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. Data synthesis based on generative adversarial networks. *Proceedings of the VLDB Endowment*, 11(10), 2018.

Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In *2016 IEEE international conference on data science and advanced analytics (DSAA)*, pages 399–410. IEEE, 2016.

Cortez Paulo, Cerdeira A., Almeida F., Matos T., and Reis J. Wine Quality. UCI Machine Learning Repository, 2009. DOI: https://doi.org/10.24432/C56S3T.

Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*, pages 744–752. PMLR, 2015.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020.

Yura Perugachi-Diaz, Jakub Tomczak, and Sandjai Bhulai. Invertible densenets with concatenated lipswish. *Advances in Neural Information Processing Systems*, 34:17246–17257, 2021.

Tomáš Pevnỳ, Václav Smídl, Martin Trapp, Ondřej Poláček, and Tomáš Oberhuber. Sum-product-transform networks: Exploiting symmetries using invertible transformations. In *International Conference on Probabilistic Graphical Models*, pages 341–352. PMLR, 2020.

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

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989. doi: 10.1109/5.18626. URL https://ieeexplore.ieee.org/document/18626.

Tahrima Rahman, Prasanna Kothalkar, and Vibhav Gogate. Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 630–645. Springer, 2014.

Danilo J. Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1530–1538, 2015. URL http://proceedings.mlr.press/v37/rezende15.html.

Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. An introduction to exponential random graph (p*) models for social networks. *Social networks*, 29(2):173–191, 2007.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

C. Sakar and Yomi Kastro. Online Shoppers Purchasing Intention Dataset. UCI Machine Learning Repository, 2018. DOI: https://doi.org/10.24432/C5F88Q.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

Benjamin Sanchez-Lengeling and Alán Aspuru-Guzik. Inverse design of materials with deep generative models. *Science*, 361:360–365, 2018.

Simo Särkkä and Arno Solin. *Applied stochastic differential equations*, volume 10. Cambridge University Press, 2019.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Davide Scassola, Sebastiano Saccani, and Luca Bortolussi. Graph-conditional flow matching for relational data generation. *arXiv preprint arXiv:2505.15668*, 2025a.

Davide Scassola, Sebastiano Saccani, Ginevra Carbone, and Luca Bortolussi. Zero-shot conditioning of score-based diffusion models by neuro-symbolic constraints. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(19):20302–20309, 2025b.

SDV Developers, DataCebo. SDMetrics: A python library for evaluating synthetic tabular data. https://github.com/sdv-dev/SDMetrics, 2024. Accessed: 2025.

Chen Shen, Sajjadur Rahman, and Estevam Hruschka. Towards probabilistic question answering over tabular data. *arXiv preprint arXiv:2506.20747*, 2025.

Jiaxin Shi, Kehang Han, Zhe Wang, Arnaud Doucet, and Michalis Titsias. Simplified and generalized masked diffusion for discrete data. *Advances in neural information processing systems*, 37:103131–103167, 2024.

Juntong Shi, Minkai Xu, Harper Hua, Hengrui Zhang, Stefano Ermon, and Jure Leskovec. Tabdiff: a mixed-type diffusion model for tabular data generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

Samarth Sinha, Sayna Ebrahimi, and Trevor Darrell. Variational adversarial active learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5972–5981, 2019.

Alexander J Smola, A Gretton, and K Borgwardt. Maximum mean discrepancy. In *13th international conference, ICONIP*, volume 6, 2006.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

Aivin V. Solatorio and Olivier Dupriez. Realtabformer: Generating realistic relational and tabular data using transformers, 2023. URL https://arxiv.org/abs/2302.02041.

Jiaming Song, Qinsheng Zhang, Hongxu Yin, Morteza Mardani, Ming-Yu Liu, Jan Kautz, Yongxin Chen, and Arash Vahdat. Loss-guided diffusion models for plug-and-play controllable generation. In *International Conference on Machine Learning*, pages 32483–32498. PMLR, 2023.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.

Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in Neural Information Processing Systems*, 33:12438–12448, 2020.

Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation. In *Uncertainty in Artificial Intelligence*, pages 574–584. PMLR, 2020.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=PxTIG12RRHS.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, et al. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv:2508.02193*, 2025.

Amy Steier, Lipika Ramaswamy, Andre Manoel, and Alexa Haushalter. Synthetic data privacy metrics. *arXiv preprint arXiv:2501.03941*, 2025.

Mihaela CÄ Stoian, Eleonora Giunchiglia, and Thomas Lukasiewicz. A survey on tabular data generation: Utility, alignment, fidelity, privacy, and beyond. *arXiv preprint arXiv:2503.05954*, 2025.

Mihaela Cătălina Stoian, Salijona Dyrmishi, Maxime Cordy, Thomas Lukasiewicz, and Eleonora Giunchiglia. How realistic is your synthetic data? constraining deep generative models for tabular data. *arXiv preprint arXiv:2402.04823*, 2024.

Beata Strack, Jon DeShazo, Krzysztof Cios, and John Clore. Diabetes 130-US Hospitals for Years 1999-2008. UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5230J.

Haoran Sun, Lijun Yu, Bo Dai, Dale Schuurmans, and Hanjun Dai. Score-based continuous-time discrete diffusion models. *arXiv preprint arXiv:2211.16750*, 2022.

Yi Ming Tay et al. Anomaly detection in tabular data using deep generative models. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2021.

Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. *Advances in neural information processing systems*, 28, 2015.

Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *Advances in neural information processing systems*, 37:84839–84865, 2024.

Jakub M. Tomczak. *Deep Generative Modeling*. Springer International Publishing, Cham, Switzerland, 2 edition, 2024. ISBN 978-3-031-64086-5. doi: 10.1007/978-3-031-64087-2.

Jakub M Tomczak and Max Welling. Improving variational auto-encoders using convex combination linear inverse autoregressive flow. *arXiv preprint arXiv:1706.02326*, 2017.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a.

Aåron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, and Alex Graves. Pixel recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2016b. URL https://arxiv.org/abs/1601.06759.

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. *Advances in neural information processing systems*, 29, 2016.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence*, 302:103602, 2022.

Michael Varley and Vaishak Belle. Fairness in machine learning with tractable models. *Knowledge-Based Systems*, 215:106715, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5998–6008, 2017.

Fabrizio Ventola, Devendra Singh Dhami, and Kristian Kersting. Generative clausal networks: Relational decision trees as probabilistic circuits. In *International Conference on Inductive Logic Programming*, pages 251–265. Springer, 2021.

Gabriele Venturato, Vincent Derkinderen, Pedro Zuidberg Dos Martires, and Luc De Raedt. Towards tractable dynamic decision making with circuits. In *The 5th Workshop on Tractable Probabilistic Modeling*, 2022.

Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 343–358. Springer, 2015.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, pages 13189–13201. Curran Associates, Inc., 2021.

Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

Marvin Wiese et al. Deep generative models for financial data. *Quantitative Finance*, 20(9):1419–1440, 2020.

Kai Xu, Georgi Ganev, Emile Joubert, Rees Davison, Olivier Van Acker, and Luke Robinson. Synthetic data generation of many-to-many datasets via random graph generation. In *The Eleventh International Conference on Learning Representations*, 2022.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Lei Xu and Kalyan Veeramachaneni. Synthesizing tabular data using generative adversarial networks. *arXiv preprint arXiv:1811.11264*, 2018.

Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. In *Advances in Neural Information Processing Systems*, 2019.

Minkai Xu, Tomas Geffner, Karsten Kreis, Weili Nie, Yilun Xu, Jure Leskovec, Stefano Ermon, and Arash Vahdat. Energy-based diffusion language models for text generation. *arXiv preprint arXiv:2410.21357*, 2024.

Scott Cheng-Hsin Yang, Baxter Eaves, Michael Schmidt, Ken Swanson, and Patrick Shafto. Structured evaluation of synthetic tabular data. *arXiv preprint arXiv:2403.10424*, 2024.

Zexi Yao, Nataša Krčo, Georgi Ganev, and Yves-Alexandre de Montjoye. The dcr delusion: Measuring the privacy risk of synthetic data. In *European Symposium on Research in Computer Security*, pages 469–487. Springer, 2025.

Haotian Ye, Haowei Lin, Jiaqi Han, Minkai Xu, Sheng Liu, Yitao Liang, Jianzhu Ma, James Y Zou, and Stefano Ermon. Tfg: Unified training-free guidance for diffusion models. *Advances in Neural Information Processing Systems*, 37:22370–22417, 2024.

I-Cheng Yeh. Default of Credit Card Clients. UCI Machine Learning Repository, 2009. DOI: https://doi.org/10.24432/C55S3H.

Jiwen Yu, Yinhuai Wang, Chen Zhao, Bernard Ghanem, and Jian Zhang. Freedom: Training-free energy-guided conditional diffusion model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 23174–23184, 2023.

Hengrui Zhang, Jiani Zhang, Balasubramaniam Srinivasan, Zhengyuan Shen, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. In *The twelfth International Conference on Learning Representations*, 2024.

Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):1–41, 2017.

Han Zhao, Pascal Poupart, and Geoffrey J Gordon. A unified approach for learning the parameters of sum-product networks. *Advances in neural information processing systems*, 29, 2016.

Zilong Zhao, Aditya Kunar, Robert Birke, and Lydia Y Chen. Ctab-gan: Effective table data synthesizing. In *Asian conference on machine learning*, pages 97–112. PMLR, 2021.

Yanqiao Zhu, Yuanqi Du, Yinkai Wang, Yichen Xu, Jieyu Zhang, Qiang Liu, and Shu Wu. A survey on deep graph generation: Methods and applications. In *Learning on Graphs Conference*, pages 47–1. PMLR, 2022.

Wojciech Ziarko. Decision making with probabilistic decision tables. In *International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing*, pages 463–471. Springer, 1999.

Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=BJJLHbb0-.

Appendix A

Supplementary Material for Chapter 3

A.1 Comparison with Logic Tensor Networks

Deep learning allows us to learn complex statistical properties at the cost of data inefficiency, struggles with generalization, lack of comprehensibility, and ignorance of prior knowledge. In contrast, Symbolic AI has proven useful for several reasoning tasks (such as theorem proving, logical inference, and verification) but at the cost of high computational complexity and struggles with large and inaccurate data. Neurosymbolic AI aims at combining the strengths of both paradigms. Recent research has focused on developing differentiable approaches for knowledge representation and reasoning. This includes relaxing logical operators into continuous operations using fuzzy semantics. Logic Tensor Networks (Badreddine et al., 2020) is an example of such a neurosymbolic framework that supports learning and reasoning about data with a knowledge base. In LTN, one can represent and perform several tasks of deep learning (such as clustering, classification, and embedding learning) with a fully differentiable first-order logic language, which they call Real Logic, assuming truth values in the interval [0,1]. Real Logic employs fuzzy semantics, where logical connectives ($\wedge$, $\vee$, ...) and quantifiers ($\forall$, $\exists$) are mapped to fuzzy operators like t-norms and aggregators. These operators enable formulas to have degrees of truth in the interval [0, 1], which are optimized during training by maximizing the satisfaction of a knowledge base of logical rules. In LTN, the preferred fuzzy logic is the Stable Product Real Logic. LogLTN (Badreddine et al., 2023) is a recent extension aiming to improve numerical stability in fuzzy semantics by working in logarithmic space, as maximizing the log satisfaction of a formula is equivalent to maximizing its satisfaction.

Our Log-Probabilistic logic has many similarities with the fuzzy logic used in LogLTN. We developed it independently, mainly motivated by probabilistic principles, as our formulas can be seen as describing unnormalized distributions encoding prior knowledge. In practice, the difference from their approach lies in the mapping of the logical disjunction. While we used the probabilistic disjunction as shown in Table 3.1, they used a surrogate of the maximum function, the LogMeanExp operator, as the maximum function is single-passing (the gradient is passed through only one of the arguments):

$$\mathrm{LME}(\mathbf{x} \mid \alpha, C) = \frac{1}{\alpha}\left(C + \log\left(\frac{\sum_{i=1}^{n} e^{\alpha x_i - C}}{n}\right)\right) \tag{A.1}$$

where $C = \max_i(\alpha x_i)$ and α is a hyperparameter. The authors claim that the LME operator is numerically stable, well-bounded, and suitable for differentiation. Our numerically stable implementation of the logical disjunction, which we discuss in Appendix A.6, performed well in our experiments while being well grounded in probabilistic principles.

A.2 Unconditional models

For all datasets except CelebA, we trained unconditional generative models from scratch. We used score-based generative models based on SDEs (Song et al., 2021), also following many of the implementation details from the repository of the original paper. We used two types of SDEs in our experiments: the variance exploding SDE (VE) and a variant of the variance preserving SDE, the sub-VP SDE. As the score-matching algorithm, we mostly used denoising score-matching and also experimented with sliced score matching (Song et al., 2020). As the numerical solver of the reverse SDE, we used Euler-Maruyama, the simplest method. For most of our experiments with SDEs, we used predictor-corrector sampling, i.e., adding steps of Langevin dynamics after each step of reverse SDE solving. As discussed previously, we often performed additional Langevin dynamics steps at $t = 0$. For the experiments with CelebA, we used a pre-trained model made available in the repository related to Song and Ermon (2020), which is based on annealed Langevin dynamics. For tabular data and time series, the neural network architectures we used were simple multi-layer perceptrons. Regarding the model trained on the Adult dataset, we dealt with categorical variables by embedding them in continuous space using one-hot encodings. At generation time, one-hot encodings were transformed back to categorical variables by taking the argmax. Similarly, we generated integer variables by rounding to the closest integer. More complete information can be found in the available code repository.

A.3 Training a score model for tabular data

In Song and Ermon (2020), they suggest scaling the neural network approximating the noise-dependent score in the following way: $\mathbf{s}_\theta(\mathbf{x}, t) = \frac{\mathbf{f}_\theta(\mathbf{x})}{\sigma_t}$ where $\mathbf{f}_\theta(\mathbf{x})$ is the neural network and σ_t is the standard deviation of the diffusion kernel $q(\tilde{\mathbf{x}} \mid \mathbf{x})$. The issue is that when $t \approx 0$, σ_t will approach zero and the score will explode. Apparently, this is not an issue for image datasets, where modeling the exact distribution is less important than the quality of samples. This is not the case for time series and tabular data, where we are interested in correctly modeling the noisy distribution that generated the data. Moreover, this is particularly important when we perform Langevin dynamics at $t \approx 0$ in order to exploit the knowledge of the exact conditional score. We then developed some practices to improve the estimate in these domains:

- We capped the scaling factor σ_t^{-1} by a certain limited value. This value has to be a hyperparameter of the training.
- We used large batch sizes and occasionally accumulated the gradient across several batches before performing the weight update. This helps reduce the noise of the score-matching loss for low values of σ_t.
- We occasionally used sliced score matching to learn the noise-dependent score on data corrupted by the diffusion process. The reason is that we found the sliced score matching loss to be less noisy than the denoising score matching loss, especially at $t \approx 0$.

A.4 Normalization

Data in experiments were normalized. Since the constraint is defined in the original data space, one has to take the normalization into account when applying the constraint to transformed data. Given a data point $\mathbf{x}$, a differentiable and invertible transformation $\mathbf{f}(\mathbf{x})$, and the constraint $c_x(\mathbf{x})$, the actual constraint that is applied to transformed data points $\mathbf{y} = \mathbf{f}(\mathbf{x})$ is $c_y(\mathbf{y}) = c_x(\mathbf{f}^{-1}(\mathbf{y}))$.

A.5 Constrained generation settings

In Table A.1, we show hyperparameters that are used in the constrained generation experiments. In experiments that involved equality constraints, we specify which function we used: LPL (log-probabilistic logic) refers to the first definition we gave in Table 3.1; otherwise, we used the squared error. We also indicate the values of k and λ we used to regulate the constraints' intensity. The step size for Langevin dynamics was chosen dynamically depending on the norm of the score; we used the same implementation as in Song et al. (2021). Constraint intensity was chosen manually, monitoring the trade-off between constraint satisfaction and sample quality across multiple experiments. The score approximation strategy $g(t)$ was less influential than the constraint intensity, although we generally observed better performance when using SNR.

A.6 Numerically stable implementation of logical disjunction

Given the values of two soft constraints $x, y \in (-\infty, 0]$ according to the Log-probabilistic logic defined above, we defined the logical disjunction as:

$$\vee(x, y) := \ln(e^x + e^y - e^{x+y}) \tag{A.2}$$

From a practical point of view, when both x and y are small, the argument of the logarithm can become 0, thus causing a numerical issue. We tackle this by rewriting the formula in the following analytically equivalent way:

$$\ln(1 + e^{y-x} - e^y) + x \quad \text{when } x > y \tag{A.3}$$

$$\ln(e^{x-y} + 1 - e^x) + y \quad \text{when } y > x \tag{A.4}$$

More concisely:

$$\vee(x, y) := \ln(1 + e^{\min(x,y)-\max(x,y)} - e^{\min(x,y)}) + \max(x, y) \tag{A.5}$$

This is analogous to the log-sum-exp trick that is commonly used to improve numerical stability of logarithms of sums of exponentials.

A.7 Hardware specifications

- Processor: AMD Ryzen Threadripper 2950X 16-Core Processor
- Memory: 125 GiB
- GPU: NVIDIA RTX A5000

Table A.1: Constrained generation details

Experiment	**Constraint**	**g(t)**	**LD steps**
White wine: complex constraint (SSM)	LPL($k = 50$)	Linear	4000
White wine: complex constraint (DSM)	LPL($k = 30$)	SNR	2000
Adult: logical formula (DSM)	LPL($k = 30$)	SNR	10
White wine: multivariate constraint (SSM)	LPL($k = 50$)	SNR	4000
eSIRS bridging	LPL, equality: $k = 7$ consistency: $k = 1$	SNR	2000
eSIRS inequality	LPL, inequality: $k = 25$ consistency: $k = 1$	SNR	2500
MNIST digits sum	LPL($k = 30$)	SNR	10
MNIST relative filling	LPL($k = 80$)	SNR	20
MNIST symmetry	LPL($k = 5$)	SNR	0
CelebA symmetry	Squared error ($\lambda = 0.19$)	SNR	0
CelebA color conditioning	Squared error, light: $\lambda = 1000$ dark: $\lambda = 800$	SNR	0
CelebA restoration	Squared error, deblurring: $\lambda = 100$ upsampling: $\lambda = 600$	SNR	0
Toy example	LPL($k = 50$)	SNR	200

Appendix B

Supplementary Material for Chapter 4

B.1 Computational Resources

The models we trained are characterized by relatively fast training and sampling phases. On a single GPU, generation took at most a few seconds per model, and training took no more than 15 minutes. Table B.1 reports the maximum runtime across repetitions for each dataset, including both training and generation.

Table B.1: Maximum runtime across repetitions for each dataset during experimentation.

Dataset Name	Running Time
AirBnB	10m 3s
Biodegradability	1m 6s
CORA	3m 10s
IMDB MovieLens	14m 25s
Rossmann	2m 57s
Walmart	1m 48s

Computing the DDA metric using the `SyntheRela` library took less than one minute for each dataset. Considering three runs for every experiment, including training, generation, and metric evaluation, the total runtime was approximately two hours. The overall research project required more computation time due to the experiments involved in developing and empirically validating the method.

The following hardware was used to train the models and evaluate the results:

- **Processor:** AMD Ryzen Threadripper 2950X (16-Core)
- **Memory:** 125 GiB
- **GPU:** NVIDIA RTX A5000

B.2 Technical Details on Training, Architecture, and Hyperparameters

We provide here a small overview of the technical details regarding neural network training and architecture. For more details, we refer the reader to the released code.

B.2.1 Training Details

As discussed earlier, training corresponds to maximum likelihood training, where the target is the original relational dataset, in the form of a graph, and the input is the relational dataset after noise is added according to the conditional probability path. In each epoch, we performed n loss evaluations and optimization steps. Every loss evaluation in an epoch is characterized by a different noise level t. In particular, we used n equally spaced values in the interval $[0, 1]$, with $n = 100$ or 200, depending on the experiment.

We use RAdam as the optimizer, together with an exponentially decaying learning rate scheduler, starting from approximately 10^{-3} and decaying to approximately 10^{-5}.

Models are trained for 10 to 40 epochs, with early stopping based on validation performance.

B.2.2 GNN Architectures

We use graph neural networks (GNNs) to process relational data and compute node embeddings $\boldsymbol{\varepsilon}^{i}$[1] for each node i in the graph.

Assume the dataset consists of nodes $\mathbf{x}^i$, each one belonging to one of the K tables. We refer to the table $\mathbf{x}^i$ belongs to as k_i. Each GNN layer computes a new representation $\mathbf{h}^i$ for node i as a function of its own features and those of its neighbors N_i:

$$\mathbf{h}^i = \text{GNNConv}(\mathbf{x}^i, N_i) \tag{B.1}$$

Since the graphs are heterogeneous, with multiple node types and edge types, standard GNN layers must be extended to handle this structure. Following the design pattern in PyTorch Geometric for heterogeneous message passing (Geometric, 2024), we define the modified layer as:

$$\mathbf{h}^i = \sum_{k=1}^{K} \text{GNNConv}_{k \to k_i}(\mathbf{x}^i, N_i^{(k)}) \tag{B.2}$$

where $N_i^{(k)}$ is the set of neighbors of node i that belong to node type k, and each $\text{GNNConv}_{k \to k_i}$ is an edge-type-specific instance of the base convolution layer. For brevity, we omit this heterogeneity adaptation in the main text but assume it in all GNN layers.

GATv2-based Architecture. Our primary GNN architecture is based on the GATv2 convolution layer (Brody et al., 2021). Each GNN block contains a GATv2 convolution followed by a residual linear transformation and ReLU activation:

$$\mathbf{h}^i = \text{ReLU}\left(\text{GATv2Conv}(\mathbf{x}^i, N_i) + \text{Linear}_{k_i}(\mathbf{x}^i)\right) \tag{B.3}$$

[1] In this section, we omit the time dependency t to keep the notation uncluttered.

$$\boldsymbol{\varepsilon}^i = \text{GATv2Conv}(\mathbf{h}^i, N_i) + \text{Linear}(\mathbf{h}^i) \tag{B.4}$$

The hidden dimensionality is shared across all node types, while the input linear layers are type-specific. The only architecture-specific hyperparameter is the dimensionality of hidden layers $\mathbf{h}^i$, which we set to 100 in our experiments.

GIN-based Architecture. We also experiment with a variant based on the Graph Isomorphism Network (GIN) (Xu et al., 2018). Each node is first projected into a shared latent space using a type-specific linear layer:

$$\mathbf{z}^i = \text{Linear}_{k_i}(\mathbf{x}^i) \tag{B.5}$$

We then apply multiple GINConv layers adapted to heterogeneous graphs as described above. The architecture-specific hyperparameters in this case are the size of embeddings $\mathbf{z}^i$, the number of GIN layers, and the width of the MLPs used in the GINConv modules. In our experiments, we used three GIN layers with an MLP width of 100. The size of the linear embedding was set to 20 or 50, depending on the experiment.

We employed the GATv2-based GNN for the AirBnB, Rossmann and Walmart datasets, while we used the GIN for the CORA, IMDB MovieLens, and Biodegradability datasets. This choice was motivated by the fact that the GIN-based architecture was necessary to model large and complex datasets (characterized by many tables and different connections), as it is more parameter-heavy and expressive. On the other hand, for smaller and simpler datasets it was prone to overfitting, so we used the GATv2 architecture, which is simpler and lighter.

B.2.3 Table-specific Denoisers

Once node embeddings $\boldsymbol{\varepsilon}_t^i$ are computed for each node $\mathbf{x}_t^i$, we use table-specific denoisers f^k to parametrize the variational distributions. In our case, this corresponds to computing the expected value of the predictive distribution:

$$\hat{\mathbf{x}}_1^i = f^{k_i}(\mathbf{x}_t^i, t, \boldsymbol{\varepsilon}_t^i) \tag{B.6}$$

Each denoiser f^k is implemented as a multi-layer perceptron (MLP), where the input is the concatenation of three components: the current noisy value $\mathbf{x}_t^i$, the time t (embedded following Dhariwal and Nichol (2021)), and the node embedding $\boldsymbol{\varepsilon}_t^i$.

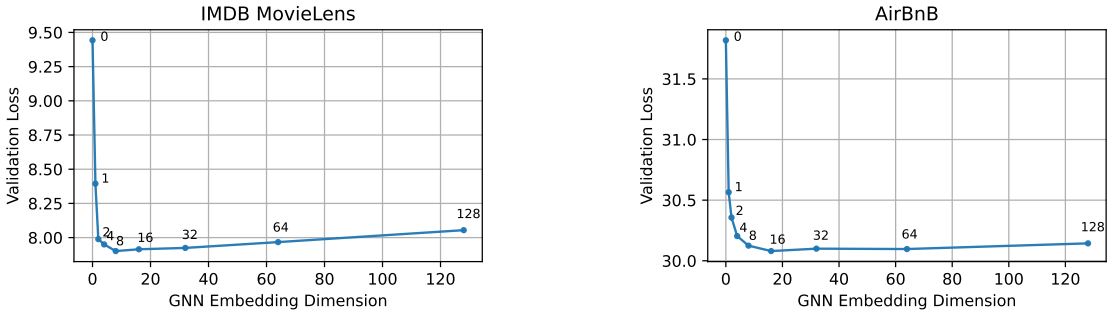
We apply Layer Normalization (Ba et al., 2016) after each hidden activation (SiLU (Elfwing et al., 2017)), except in the final layer. The output layer is linear and restores the original dimensionality of $\mathbf{x}_t^i$. For one-hot-encoded categorical features, a final softmax activation is applied.

For each table k, the hyperparameters are the number of layers and the width (i.e., number of hidden units) of each layer in the MLP. In our experiments, we used 2 or 3 hidden layers, with the number of hidden units ranging from 10 to 1000 depending on the experiment.

B.2.4 Tuning Node Embedding Size

The size of the embedding produced by the GNN is an important hyperparameter. A large embedding size can cause the neural network to memorize structures, while an overly small one will limit expressiveness by restricting information flow. For our experiments, we chose embedding sizes in the range of 2–10, a conservative choice to avoid overfitting and privacy leaks. Figure B.1

shows how the minimum validation loss varies with the embedding size for two datasets. We observe that for the IMDB-MovieLens dataset, performance degrades if the embedding size is arbitrarily increased. For the Airbnb experiments, the effect is less pronounced, likely due to the different GNN architecture used.



(a) IMDB MovieLens

(b) Airbnb

Figure B.1: Best validation loss as a function of the GNN embedding size. A value of zero means the GNN is not used.

B.3 Datasets

We summarize the datasets used in our experiments in Table B.2. To allow comparisons with prior work, the AirBnB, Rossmann, and Walmart datasets were downsampled. AirBnB and Rossmann are relatively simple, each containing a single foreign key relationship, while the Walmart dataset features a table with two child tables. The IMDB, Biodegradability, and CORA datasets have more complex structures, including multiple foreign keys and tables referencing multiple parent tables. In particular, both Biodegradability and CORA include tables with two distinct foreign keys pointing to the same parent table. For example, the `cites` table in CORA contains both the ID of the citing paper and that of the cited paper. Additional details about the datasets are provided in Hudovernik et al. (2024).

Finally, the sampling procedure of the foreign-key graph depended on the dataset. For the IMDB-MovieLens and the CORA datasets, where the largest connected component includes respectively 99% and 93% of the data, the original foreign-key graph was retained. For the other datasets, the graph was built by sampling with replacement an equal number of connected components from the original graph.

Table B.2: Overview of the datasets used in the experiments, showing for each table of each dataset the number of rows, the number of features (feature columns) and tables referred by foreign keys (the parent tables).

Dataset	Table	# Rows	# Features	Foreign Keys
AirBnB	users	10,000	15	–
	sessions	47,217	5	users
Biodegradability	molecule	328	3	–
	group	1,736	1	–
	atom	6,568	1	molecule
	gmember	6,647	–	atom, group
	bond	6,616	1	atom1, atom2
CORA	paper	2,708	1	–
	content	49,216	1	paper
	cites	5,429	–	paper1, paper2
IMDB MovieLens	users	6,039	3	–
	movies	3,832	4	–
	actors	98,690	2	–
	directors	2,201	2	–
	ratings	996,159	1	movie, user
	movies2actors	138,349	1	movie, actor
	movies2directors	4,141	1	movie, director
Rossmann	store	1,115	9	–
	historical	57,970	7	store
Walmart	stores	45	2	–
	features	225	11	store
	depts	15,047	4	store

Appendix C

Supplementary Material for Chapter 5

C.1 Conditional Sampling with Probabilistic Circuits

Probabilistic circuits (PCs) support efficient conditional sampling when they are smooth and decomposable (Choi et al., 2020), and when input units, if multivariate, support efficient conditional sampling. We now briefly describe how to perform conditional sampling with PCs. Let $\mathbf{X} = \{\mathbf{X}_s, \mathbf{X}_c\}$ be the set of random variables modeled by a PC $p(\mathbf{X})$, where $\mathbf{X}_s$ are the variables to be sampled and $\mathbf{X}_c$ are the given variables to condition on. Since a probabilistic circuit is either a simple tractable distribution, a mixture of PCs defined over the same scope or a product of PCs over disjoint scopes, sampling can be performed recursively:

Input units. If the PC is an input unit defined over variables $\mathbf{X}$, then we can directly sample from the conditional distribution $p(\mathbf{X}_s \mid \mathbf{X}_c)$ using the input unit's conditional sampling procedure.

Sum units. If the PC is a mixture of n child PCs $p_i(\mathbf{X})$ with weights w_i, i.e.:

$$p(\mathbf{X}) = \sum_{i=1}^{n} w_i q_i(\mathbf{X}) \tag{C.1}$$

then, according to the latent variable interpretation of mixtures, we have:

$$p(\mathbf{X}) = \sum_{z=1}^{n} p(\mathbf{X}, z) = \sum_{z=1}^{n} p(z) p(\mathbf{X} \mid z) \tag{C.2}$$

where $p(z) = w_i$ and $p(\mathbf{X} \mid z) = q_i(\mathbf{X})$. Thus, we can write the conditional distribution as:

$$p(\mathbf{X}_s \mid \mathbf{X}_c) = \sum_{z=1}^{n} p(\mathbf{X}_s, z \mid \mathbf{X}_c) = \sum_{z=1}^{n} p(z \mid \mathbf{X}_c) p(\mathbf{X}_s \mid \mathbf{X}_c, z) \tag{C.3}$$

This means that to sample from $p(\mathbf{X}_s \mid \mathbf{X}_c)$, we can first sample z from $p(z \mid \mathbf{X}_c)$

$$p(z \mid \mathbf{X}_c) = \frac{p(z) p(\mathbf{X}_c \mid z)}{p(\mathbf{X}_c)} = \frac{w_i q_i(\mathbf{X}_c)}{\sum_{j=1}^{n} w_j q_j(\mathbf{X}_c)} \tag{C.4}$$

where marginals $q_i(\mathbf{X}_c)$ can be computed efficiently with a forward pass, and then recursively sample $\mathbf{X}_s$ from the conditional distribution of the selected child PC:

$$p(\mathbf{X}_s \mid \mathbf{X}_c, z = i) = q_i(\mathbf{X}_s \mid \mathbf{X}_c) \tag{C.5}$$

Product units. Without loss of generality, we consider a PC defined as a product of two child PCs defined over disjoint sets of variables $\mathbf{X}^1$ and $\mathbf{X}^2$:

$$p(\mathbf{X}) = p(\mathbf{X}^1, \mathbf{X}^2) = p_1(\mathbf{X}^1)p_2(\mathbf{X}^2) \tag{C.6}$$

Then $\mathbf{X}_s$ and $\mathbf{X}_c$ can be partitioned as $\mathbf{X}_s = \{\mathbf{X}_s^1, \mathbf{X}_s^2\}$ and $\mathbf{X}_c = \{\mathbf{X}_c^1, \mathbf{X}_c^2\}$, where $\mathbf{X}_s^i$ and $\mathbf{X}_c^i$ are the variables to sample and to condition on respectively. It can be shown that:

$$\begin{aligned}
p(\mathbf{X}_s \mid \mathbf{X}_c) &= \frac{p(\mathbf{X}_s, \mathbf{X}_c)}{p(\mathbf{X}_c)} && \text{(C.7)}\\
&= \frac{p_1(\mathbf{X}_s^1, \mathbf{X}_c^1)p_2(\mathbf{X}_s^2, \mathbf{X}_c^2)}{\int p(\mathbf{X}_c, \mathbf{X}_s)\,\mathrm{d}\mathbf{X}_s} && \text{(C.8)}\\
&= \frac{p_1(\mathbf{X}_s^1, \mathbf{X}_c^1)p_2(\mathbf{X}_s^2, \mathbf{X}_c^2)}{\int\int p(\mathbf{X}_c^1, \mathbf{X}_c^2, \mathbf{X}_s^1, \mathbf{X}_s^2)\,\mathrm{d}\mathbf{X}_s^1\,\mathrm{d}\mathbf{X}_s^2} && \text{(C.9)}\\
&= \frac{p_1(\mathbf{X}_s^1, \mathbf{X}_c^1)p_2(\mathbf{X}_s^2, \mathbf{X}_c^2)}{\int\int p(\mathbf{X}_c^1, \mathbf{X}_s^1)p(\mathbf{X}_c^2, \mathbf{X}_s^2)\,\mathrm{d}\mathbf{X}_s^1\,\mathrm{d}\mathbf{X}_s^2} && \text{(C.10)}\\
&= \frac{p_1(\mathbf{X}_s^1, \mathbf{X}_c^1)p_2(\mathbf{X}_s^2, \mathbf{X}_c^2)}{p_1(\mathbf{X}_c^1)p_2(\mathbf{X}_c^2)} && \text{(C.11)}\\
&= p_1(\mathbf{X}_s^1 \mid \mathbf{X}_c^1)p_2(\mathbf{X}_s^2 \mid \mathbf{X}_c^2) && \text{(C.12)}
\end{aligned}$$

Therefore, to sample from $p(\mathbf{X}_s \mid \mathbf{X}_c)$, we can recursively sample $\mathbf{X}_s^1$ from $p_1(\mathbf{X}_s^1 \mid \mathbf{X}_c^1)$ and $\mathbf{X}_s^2$ from $p_2(\mathbf{X}_s^2 \mid \mathbf{X}_c^2)$, and combine the two samples to obtain $\mathbf{X}_s$.

Notice that this procedure requires a forward pass to evaluate the marginals $p(\mathbf{X}_c)$ at sum units, and a backward pass to perform the recursive sampling.

C.2 Training Details

We trained our models by maximizing the log-likelihood of the training data using the RAdam optimizer (Liu et al., 2019) and batch size 256 for probabilistic circuits and 1024 for shallow mixture models. We used a learning rate scheduler that reduces the learning rate by a factor of 0.85 if the validation loss has not improved in the last epoch, starting from an initial learning rate of 2×10^{-2}. We used 10% of the training data as a validation set to perform early stopping with a patience of 10 epochs. Columns with less than 50 unique values were treated as categorical, while the remaining ones as continuous. Quantized numerical columns were de-quantized by adding uniform noise in the range $[-\frac{q}{2}, \frac{q}{2}]$, where q is the quantization step. These were re-quantized after sampling. The only hyperparameter that depends on the dataset is the number of units (we used the same number for both sum and input units), this is reported in Table C.1.

C.3 Other Metrics

We report in Table C.2 and Table C.3 the results relative to the α-Precision and β-Recall metrics (Alaa et al., 2022) for all the considered baselines. The idea behind these metrics is to separately

Table C.1: Number of sum and input units used in TabPC for each dataset and shallow mixture components for each dataset.

Dataset	# TabPC Units	# Shallow Mixture Components
Adult	$3,500$	$20,000$
Beijing	$2,000$	$50,000$
Default	$2,000$	$20,000$
Diabetes	$2,500$	$20,000$
Magic	$2,000$	$10,000$
News	$1,000$	$10,000$
Shoppers	$2,000$	$20,000$

Table C.2: Alpha-precision (higher is better).

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	$0.761_{\pm 0.003}$	$0.974_{\pm 0.001}$	$0.677_{\pm 0.002}$	$0.827_{\pm 0.003}$	$0.898_{\pm 0.003}$	$0.98_{\pm 0.001}$	$0.722_{\pm 0.005}$	0.834
TVAE	$0.618_{\pm 0.002}$	$0.802_{\pm 0.002}$	$0.861_{\pm 0.003}$	$0.207_{\pm 0.001}$	$0.975_{\pm 0.003}$	$0.956_{\pm 0.003}$	$0.664_{\pm 0.006}$	0.726
GReaT	$0.594_{\pm 0.003}$	$0.983_{\pm 0.001}$	-	-	$0.868_{\pm 0.003}$	-	-	-
STaSy	$0.914_{\pm 0.003}$	$0.935_{\pm 0.003}$	$0.903_{\pm 0.003}$	—	$0.978_{\pm 0.003}$	$0.977_{\pm 0.002}$	$0.946_{\pm 0.006}$	-
CoDi	$0.783_{\pm 0.004}$	$0.964_{\pm 0.003}$	$0.825_{\pm 0.002}$	$0.435_{\pm 0.002}$	$0.859_{\pm 0.004}$	$0.925_{\pm 0.003}$	$0.888_{\pm 0.005}$	0.811
TabSyn	$0.987_{\pm 0.003}$	$0.987_{\pm 0.002}$	$0.992_{\pm 0.002}$	$0.981_{\pm 0.002}$	$0.994_{\pm 0.002}$	$0.939_{\pm 0.003}$	$0.989_{\pm 0.005}$	0.981
TabDiff	$0.991_{\pm 0.002}$	$0.981_{\pm 0.003}$	$0.986_{\pm 0.004}$	$0.943_{\pm 0.002}$	$0.994_{\pm 0.002}$	$0.914_{\pm 0.002}$	$0.991_{\pm 0.003}$	0.971
Fully Factorized	$0.958_{\pm 0.002}$	$0.976_{\pm 0.002}$	$0.696_{\pm 0.001}$	$0.981_{\pm 0.002}$	$0.868_{\pm 0.003}$	$0.928_{\pm 0.003}$	$0.966_{\pm 0.003}$	0.91
Fully Factorized (P)	$0.958_{\pm 0.002}$	$0.954_{\pm 0.002}$	$0.696_{\pm 0.001}$	$0.981_{\pm 0.001}$	$0.862_{\pm 0.002}$	$0.837_{\pm 0.002}$	$0.971_{\pm 0.002}$	0.894
ShallowMixture	$0.979_{\pm 0.004}$	$0.955_{\pm 0.004}$	$0.973_{\pm 0.002}$	$0.99_{\pm 0.001}$	$0.972_{\pm 0.005}$	$0.971_{\pm 0.003}$	$0.985_{\pm 0.003}$	0.975
TabPC	$0.996_{\pm 0.002}$	$0.996_{\pm 0.001}$	$0.995_{\pm 0.001}$	$0.996_{\pm 0.001}$	$0.994_{\pm 0.003}$	$0.988_{\pm 0.005}$	$0.994_{\pm 0.003}$	0.994

Table C.3: Beta-recall (higher is better).

Method	Adult	Beijing	Default	Diabetes	Magic	News	Shoppers	Average
CTGAN	$0.119_{\pm 0.002}$	$0.372_{\pm 0.002}$	$0.124_{\pm 0.002}$	$0.1_{\pm 0.001}$	$0.155_{\pm 0.003}$	$0.252_{\pm 0.002}$	$0.22_{\pm 0.004}$	0.192
TVAE	$0.107_{\pm 0.001}$	$0.052_{\pm 0.001}$	$0.314_{\pm 0.002}$	0.025	$0.37_{\pm 0.004}$	$0.29_{\pm 0.002}$	$0.26_{\pm 0.004}$	0.203
GReaT	$0.484_{\pm 0.002}$	$0.604_{\pm 0.005}$	-	-	$0.398_{\pm 0.004}$	-	-	-
STaSy	$0.359_{\pm 0.002}$	$0.496_{\pm 0.002}$	$0.383_{\pm 0.002}$	—	$0.445_{\pm 0.003}$	$0.397_{\pm 0.003}$	$0.336_{\pm 0.005}$	-
CoDi	$0.09_{\pm 0.001}$	$0.54_{\pm 0.003}$	$0.186_{\pm 0.002}$	$0.014_{\pm 0.001}$	$0.506_{\pm 0.003}$	$0.366_{\pm 0.002}$	$0.186_{\pm 0.003}$	0.27
TabSyn	$0.485_{\pm 0.002}$	$0.494_{\pm 0.002}$	$0.466_{\pm 0.003}$	$0.353_{\pm 0.002}$	$0.477_{\pm 0.003}$	$0.38_{\pm 0.003}$	$0.477_{\pm 0.006}$	0.448
TabDiff	$0.514_{\pm 0.003}$	$0.591_{\pm 0.003}$	$0.517_{\pm 0.003}$	$0.434_{\pm 0.002}$	$0.476_{\pm 0.003}$	$0.362_{\pm 0.002}$	$0.5_{\pm 0.006}$	0.485
Fully Factorized	$0.05_{\pm 0.001}$	$0.434_{\pm 0.002}$	$0.051_{\pm 0.001}$	$0.094_{\pm 0.001}$	$0.01_{\pm 0.001}$	0.009	$0.128_{\pm 0.003}$	0.111
Fully Factorized (P)	$0.076_{\pm 0.002}$	$0.439_{\pm 0.003}$	$0.066_{\pm 0.001}$	$0.096_{\pm 0.001}$	$0.019_{\pm 0.001}$	$0.017_{\pm 0.001}$	$0.233_{\pm 0.002}$	0.135
ShallowMixture	$0.504_{\pm 0.002}$	$0.634_{\pm 0.002}$	$0.35_{\pm 0.003}$	$0.182_{\pm 0.001}$	$0.503_{\pm 0.004}$	$0.05_{\pm 0.003}$	$0.373_{\pm 0.006}$	0.371
TabPC	$0.486_{\pm 0.003}$	$0.538_{\pm 0.002}$	$0.458_{\pm 0.003}$	$0.424_{\pm 0.003}$	$0.505_{\pm 0.004}$	$0.34_{\pm 0.006}$	$0.497_{\pm 0.004}$	0.464

measure the fidelity and the diversity of the generated data with respect to the real one. Alpha-precision measures the quality of the generated data, in the sense of how much of the generated data lies within the support of the real data, while beta-recall measures its coverage of the real data distribution.

C.4 Computational Resources

The following hardware was used to train the models and evaluate the results:

- **GPU:** NVIDIA RTX A6000 (49 GiB)

- **Processor:** AMD EPYC 7452 32-Core Processor
- **Memory:** 512 GiB

UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

La borsa di dottorato è cofinanziata con risorse dell'Unione europea, NextGeneration EU - Piano Nazionale di Ripresa e Resilienza, Missione 4 – Componente 2 – Investimento 3.3 CUP J92B22001000007