# Knowledge-Refined Deep Neural Networks for Real-World Applications


Mihaela Cătălina Stoian
St Hilda's College
University of Oxford

A thesis submitted for the degree of
*Doctor of Philosophy*

Trinity 2025

To my family
*Familiei mele*

# Acknowledgements

I am most grateful to my supervisor, Thomas Lukasiewicz, for his exceptional and consistent guidance and support throughout my DPhil studies. His mentorship helped me build the confidence to overcome challenges and grow as a researcher, and I am especially grateful for his insightful advice and for his invaluable direction during moments of indecision. I am also thankful for our collaboration.

I am also deeply thankful to Eleonora Giunchiglia for her exceptional guidance and mentorship. Her steadfast support, especially in difficult times, was invaluable, and she has significantly contributed to my growth as a researcher. I am also grateful for our collaboration.

My sincere thanks also go to my other collaborators: Salijona Dyrmishi, Maxime Cordy, Salman Khan, Alex Tatomir, Izzeddin Teeti, Andrew Bradley, Fabio Cuzzolin, Reza Javanmard Alitappeh, Gurkirt Singh.

I am grateful to my examiners, Ani Calinescu and Pasquale Minervini, for their thorough review and helpful comments on my DPhil thesis.

I also acknowledge the excellent administrative support from Sarah Retz-Jones, Jodie Mattioli and the IT team of the Department of Computer Science.

I would also like to thank Nicky Charles and St Hilda's College for creating a supportive and vibrant environment, and to thank the wonderful friends I made in Oxford. Further, I am especially thankful to my friends from before this chapter began for their constant encouragement from afar.

This work has been supported by the Engineering and Physical Sciences Research Council Scholarship for Doctoral Studies EP/T517811/1.

Finally, I want to offer my deepest gratitude to my parents, Florentina and Ion, and to Kyriakos, for their unconditional support.

# Abstract

Deep neural networks have demonstrated remarkable success in finding patterns in raw data, yet their purely data-driven nature often results in predictions that violate fundamental background knowledge. In this thesis, I address this critical problem by proposing novel neuro-symbolic methods, which integrate background knowledge constraints into neural networks for real-world applications. While *background knowledge* broadly refers to established logical rules or physical constraints that govern a specific domain, in the context of this work, *background knowledge* (or simply *knowledge*) refers to: (i) a set of propositional logic formulae written over a set of labels, (ii) a set of linear inequalities, or (iii) a set of disjunctions over linear inequalities, capturing the relations between data features. Consequently, a knowledge-refined neural network is one that either augments standard data-driven learning by strictly adhering to this background knowledge or guides the learning using the background knowledge. I focus on two main real-world applications for my proposed methods: the synthesis of tabular data across domains such as healthcare and finance, as well as safety-critical applications like autonomous driving. Importantly, the datasets used to evaluate the methods for data synthesis cover a diverse range of characteristics to address real-world challenges, including: binary and multi-class classification datasets, regression datasets, data-scarce settings, large-scale data, and background knowledge constraints of various complexities. Similarly, in the context of autonomous driving, I evaluate my approach on a complex dataset annotated with a large number of constraints.

Firstly, I focus on integrating background knowledge into deep generative models for tabular data synthesis. While powerful, these models often fail to comply with inherent relationships between the tabular features. To solve this, I propose the first framework capable of injecting constraints, expressed as linear inequalities, into deep generative models. Based on a

novel layer added directly into the model's topology, this approach ensures compliance with the constraints during both training and inference, obviating the need for inefficient rejection sampling. Additionally, I investigate how different heuristics for constructing this layer affect model performance.

Secondly, I advance this work to a more complex scenario involving background knowledge constraints expressed as disjunctions over linear inequalities. These constraints are significantly more expressive, capable of modelling non-convex and even disconnected output spaces. To this end, I introduce a new layer that corrects the generated data to adhere to these complex relationships. This constitutes the first framework to integrate such constraints into deep generative models while guaranteeing their satisfaction. Crucially, the layer not only ensures compliance but also significantly improves the machine learning efficacy of the models without hindering sample generation times.

Thirdly, to complement the layer-based approaches above, I investigate an alternative neuro-symbolic paradigm for the task of multi-label classification. I demonstrate that popular loss-based neuro-symbolic methods, which use constraints to guide learning, become impractical when dealing with a large number of propositional logic constraints. To overcome this, I design a memory-efficient, logic-based loss function that effectively penalises non-compliant outputs, significantly improving predictive accuracy, particularly in scenarios where little annotated data is available.

Finally, to bridge the gap between research and practice, I introduce PiShield, a package that encapsulates the methods developed in this thesis. By providing accessible implementations of both layer-based mechanisms as well as of the memory-efficient loss-based functions, PiShield is a versatile tool for applying neuro-symbolic approaches to practical problems, allowing users to build more accurate and reliable models for real-world challenges.

# Publications

This thesis is based on the following publications (in chronological order):

Stoian et al. (2025): **Mihaela C. Stoian**, Salijona Dyrmishi, Maxime Cordy, Thomas Lukasiewicz, and Eleonora Giunchiglia. Generalised Linearly-Constrained Layers for Deep Generative Modelling. *Manuscript in preparation for journal publication*, 2026.

Stoian et al. (2026): **Mihaela C. Stoian**, Eleonora Giunchiglia and Thomas Lukasiewicz. A Survey on Deep Learning Approaches for Tabular Data Generation: Utility, Alignment, Fidelity, Privacy, Diversity, and Beyond. *Transactions on Machine Learning Research*, 2026.

Stoian and Giunchiglia (2025): **Mihaela C. Stoian** and Eleonora Giunchiglia. Beyond the Convexity Assumption: Realistic Tabular Data Generation under Quantifier-Free Real Linear Constraints. In *Proceedings of the International Conference on Learning Representations*, 2025.

Stoian (2024): **Mihaela C. Stoian**. Deep Learning with Requirements in the Real World. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Doctoral Consortium, 2024.

Stoian et al. (2024a): **Mihaela C. Stoian**[1], Salijona Dyrmishi[1], Maxime Cordy, Thomas Lukasiewicz, and Eleonora Giunchiglia. How Realistic Is Your Synthetic Data? Constraining Deep Generative Models for Tabular Data. In *Proceedings of the International Conference on Learning Representations*, 2024.

Stoian et al. (2024b): **Mihaela C. Stoian**, Alex Tatomir, Thomas Lukasiewicz, and Eleonora Giunchiglia. PiShield: A PyTorch Package for Learning with Requirements. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2024.

[1]Equal first author.

Stoian et al. (2023): **Mihaela C. Stoian**, Eleonora Giunchiglia, and Thomas Lukasiewicz. Exploiting T-norms for deep learning in autonomous driving. In *Proceedings of the International Workshop on Neural-Symbolic Learning and Reasoning*, 2023.

During my DPhil studies, I have also co-authored the following publications, which are not part of my thesis:

Khan et al. (2024): Salman Khan, Izzeddin Teeti, Reza Javanmard Alitappeh, **Mihaela C. Stoian**, Eleonora Giunchiglia, Gurkirt Singh, Andrew Bradley, Fabio Cuzzolin. ROAD-Waymo: Action Awareness at Scale for Autonomous Driving. *Submitted for journal publication*, 2025.

Giunchiglia et al. (2024): Eleonora Giunchiglia, Alex Tatomir, **Mihaela C. Stoian**, and Thomas Lukasiewicz. CCN+: A neuro-symbolic framework for deep learning with requirements. *International Journal of Approximate Reasoning*, 171, 2024.

Dyrmishi et al. (2024): Salijona Dyrmishi, **Mihaela C. Stoian**, Eleonora Giunchiglia, Maxime Cordy. Deep generative models as an adversarial attack strategy for tabular machine learning. In *Proceedings of the International Conference on Machine Learning and Cybernetics*, 2024.

Giunchiglia et al. (2023b): Eleonora Giunchiglia, **Mihaela C. Stoian**, Salman Khan, Fabio Cuzzolin, and Thomas Lukasiewicz. ROAD-R: The autonomous driving dataset for learning with requirements. *Machine Learning*, 112, 2023.

Giunchiglia et al. (2022): Eleonora Giunchiglia, **Mihaela C. Stoian**, and Thomas Lukasiewicz. Deep learning with logical constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2022.

Finally, during my DPhil, I have had the pleasure to co-supervise two Master's and one Bachelor's student theses:

1. *Adaptive Label Ordering in Deep Learning with Logical Constraints.* Thesis written by Alina Godun for the Master's degree, Faculty of

Informatics, Vienna University of Technology, (In progress, expected 2025).

2. *Investigating and Mitigating the Overpopulation Problem Typical of Constrained Deep Generative Models in the Tabular Data Generation Field.* Thesis written by Shinde Lee for the degree of Master of Engineering in Electronic and Information Engineering, Department of Electrical and Electronic Engineering, Imperial College London, 2025.

3. *Investigating the Statistical Properties of Constrained Deep Generative Models.* Thesis written by Luka Pejic for the degree of Bachelor of Science in Software and Information Engineering, Faculty of Informatics, Vienna University of Technology, 2025.

# Contents

# List of Figures

# List of Tables

# Glossary of Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| ANN | Artificial Neural Network |
| AP | Average Precision |
| DGM | Deep Generative Model |
| DL | Deep Learning |
| DNN | Deep Neural Network |
| DRL | Disjunctive Refinement Layer (a differentiable layer capable of constraining DGM output spaces according to QFLRA formulae) |
| f-mAP | frame-wise mean Average Precision |
| FOL | First-Order Logic |
| GAN | Generative Adversarial Network |
| GPU | Graphics Processing Unit |
| IoU | Intersection over Union |
| KDE | Kernel Density Estimation |
| LL | Linearly-constrained Layer (a differentiable layer capable of constraining DGM output spaces according to linear inequality constraints) |
| mAP | mean Average Precision |

| | |
|---|---|
| ML | Machine Learning |
| NeSy | Neuro-Symbolic AI |
| PiShield | A PyTorch-based package encapsulating Shield Layers and Shield Losses for integrating background knowledge into neural networks |
| QFLRA | Quantifier-Free Linear Real Arithmetic (formulae capturing relationships via linear inequalities, conjunctions, disjunctions, and negations) |
| RCGRU | Random Connectivity Gated Recurrent Unit |
| resp. | respectively |
| ROAD-R | The Road Event Awareness Dataset with Logical Requirements |
| s.t. | such that |
| TSTR | Train on Synthetic, Test on Real (a protocol to evaluate machine learning efficacy by training a machine learning model on synthetic data and testing it on real data) |
| w.l.o.g. | without loss of generality |
| w.r.t. | with respect to |

# Chapter 1

# Introduction

Deep neural networks (DNNs) have demonstrated a remarkable ability to find patterns in raw data, leading to their prevalence. This success, however, has been met with growing concerns about the safety and reliability of purely data-driven approaches, as such approaches do not account for any available background knowledge and, thus, can lead to contradictory outputs. Neuro-symbolic AI, which aims to combine neural networks with symbolic reasoning capabilities, has been proposed to tackle these concerns (d'Avila Garcez and Lamb, 2023). In this thesis, I address this critical problem by proposing novel neuro-symbolic methods while also ensuring their feasibility for real-world applications.

To provide context for my work, *background knowledge* (or simply *knowledge*) specifically refers to a user-defined set of: (i) propositional logic formulae written over a set of labels (e.g., in autonomous driving, expressing that a pedestrian cannot be on both the left and right pavement simultaneously), (ii) linear inequalities (e.g., in healthcare, ensuring a patient's minimum haemoglobin level is less than or equal to their maximum), or (iii) disjunctions over linear inequalities, which capture the inherent relations between data features. Consequently, I design *knowledge-refined* neural networks, defined as models that either augment standard data-driven learning by strictly adhering to this predefined background knowledge or guide the learning process using this knowledge.

To ensure my proposed knowledge-refined methods can be deployed in practice, I focus on two main real-world applications: the synthesis of tabular data across different domains such as healthcare and finance, and safety-critical applications like autonomous driving. Importantly, the datasets used to evaluate these methods span a wide range of complexities to reflect real-world challenges. For tabular data, this includes binary and multi-class classification datasets, regression datasets, data-scarce

settings, large-scale data, and background knowledge constraints of various complexities (from linear inequalities to expressive disjunctions of linear inequalities capturing relations among the data features). Similarly, in the context of autonomous driving, I evaluate my approach on a highly complex dataset annotated with a large set of propositional logic constraints capturing relations between the output labels in a multi-label classification scenario.

Although neuro-symbolic AI is not new, with a history of works spanning over 20 years (see, e.g., (Towell and Shavlik, 1994) and (d'Avila Garcez and Zaverucha, 1999), which are early methods that encode symbolic knowledge into the neural networks' weights), it has gained significant new excitement and interest from the research community in the recent years. This resurgence has been driven, in large part, by the breakthrough paper on deep learning by Hinton et al. (2006), often considered the turning point for the practical success of neural networks. Neuro-symbolic AI encompasses approaches that use pre-defined constraints (given by a user or domain knowledge) and approaches that discover constraints from data (e.g., Daniele et al. (2023a); Debot et al. (2024)). In this thesis, I focus only on the former group of methods, among which two main categories of methods have emerged: (i) approaches that embed the constraints into the loss function to guide the learning process and penalise neural networks when they violate the constraints (e.g., (Diligenti et al., 2012; Donadello et al., 2017; Xu et al., 2018; Fischer et al., 2019; Badreddine et al., 2022; Li et al., 2023)), and (ii) methods that integrate the constraints directly into the architecture of the neural networks to guarantee that the predictions satisfy the constraints (e.g., (Giunchiglia and Lukasiewicz, 2021; Hoernle et al., 2022; Ahmed et al., 2022b)).

As prior work has shown, integrating background knowledge into neural networks offers several key advantages, as the constraints (i) are often easier to obtain than large-scale annotated datasets, e.g., creating autonomous driving datasets requires the expensive and time-consuming process of annotating millions of bounding boxes (Giunchiglia et al., 2023b), (ii) are often interpretable, as they capture domain knowledge, (iii) offer a formal way to verify whether the predictions are correct (with respect to the background knowledge), (iv) have the potential to improve prediction quality, and (v) have the potential to allow models to learn from fewer examples.

Beyond a formal language, neuro-symbolic systems also require that they are *modular*, a fundamentally important property for any computing system, as argued by d'Avila Garcez and Lamb (2023). Modularity creates a separation between the system's neural network and reasoning mechanism modules. This separation, in turn,

allows for the easy integration of new advancements from a fast-paced field like deep learning, enabling neuro-symbolic systems to co-evolve. As I demonstrate in this thesis, this property not only facilitates adding constraint-enforcing mechanisms to multiple standard, off-the-shelf deep learning architectures but also allows for swapping out the constraint-enforcing mechanisms themselves (for instance, replacing a constraints-aware layer designed for one formal language with another designed for more expressive constraints).

Despite the growing development of neuro-symbolic approaches, such methods have largely been designed for and applied to small datasets that fail to capture real-world challenges. A notable exception is the work in (Giunchiglia et al., 2023b), which introduced the first autonomous driving dataset annotated with logical constraints. In this thesis, I emphasise that, in addition to modularity, the neuro-symbolic field would benefit from focusing on developing methods in the context of real-world applications. This focus is essential to derive practical benefits, ensure the utility of proposed methods that account for the background knowledge, and ultimately advance the state of the art.

Furthermore, especially in these real-world contexts, it is critical to determine whether systems that account for and enforce constraints are truly useful in practice compared to the standard models. To this end, evaluating the prediction quality of the neuro-symbolic approaches offers a more principled way to compare these new methods against standard deep learning. This raises a key research question: *is there a trade-off between enforcing constraints and the quality of the neural network's predictions when considering real-world applications?* This is a central question that I address in my thesis, specifically in scenarios involving deep generative modelling for tabular data and autonomous driving.

Naturally, designing neuro-symbolic methods for such complex scenarios comes with unique challenges, which I describe and address in the relevant parts of my thesis. Furthermore, in an effort to bridge theory with practice, I package these methods into practical modules capable of enforcing constraints, ready for use also in other real-world scenarios. In what follows, I provide an overview of this thesis and a summary of my contributions.

First, in Chapter 2, I present the concepts needed to navigate the later content and related work relevant to the methods I introduce in this thesis. Specifically, I provide the necessary background into neuro-symbolic methods, as well as the real-world scenarios where such methods have been applied or could be applied as a potential future direction. Further, I focus on a specific deep learning task, namely

deep generative modelling, and review the approaches used for tabular data synthesis, which is one of the main real-world applications considered in in this thesis, while also offering a new perspective on what makes synthetic data practically useful, and outlining key steps toward achieving such a goal.

The motivation for focusing on synthetic data is that modern deep learning relies on massive datasets, yet real-world tabular data is often restricted by privacy regulations or limited by class imbalances and scarcity of rare data examples. Synthetic data generation addresses these bottlenecks by providing realistic, artificial datasets that democratise data access (Jordon et al., 2022; Savage, 2023) and improve model robustness through data augmentation. However, for such data to be realistic, they must adhere to background knowledge that captures domain-specific constraints. This is the primary motivation for the constrained generative models I propose in Chapters 3-5.

To this end, in Chapter 3, I first introduce a framework for deep generative modelling using background knowledge to synthesise realistic tabular data. To this end, I formalise the problem of constrained generative modelling with background knowledge expressed as linear inequalities. Then, I propose a novel approach which automatically parses user-defined linear inequality constraints and builds a differentiable Linearly-constrained Layer (LL) that can be seamlessly integrated with a given Deep Generative Model (DGM). This, in turn, results in a novel model, namely the deep generative model equipped with the Linearly-constrained Layer (DGM+LL), whose sample space is guaranteed to be compliant with the constraints. Importantly, this framework distinguishes itself from the prior works which rely on rejection sampling, being the first work to directly incorporate background knowledge into DGMs and guarantee its satisfaction. Further, through an in-depth experimental analysis, I show that integrating constraints via LL into DGMs provides significant overall improvements w.r.t. two measures commonly used for evaluating the quality of synthetic data, namely: machine learning efficacy and detection, all while not hindering the sample generation runtime during inference. In the context of real-world applications, it is worth noting that LL is easy to use, as it requires only two inputs in order to be applied to any DGM, namely: the user-defined constraints and an ordering of the tabular data features.

The ordering of the features determines the order in which LL corrects the predicted features' values. Thus, the output of LL for a given sample may depend on the chosen ordering (i.e., different feature orderings may lead to different LL outputs). To this end, in Chapter 4, I propose five different methods to determine such feature

orderings, namely: (i) random, (ii) correlation-based, (iii) kernel density estimation-based, (iv) Wasserstein distance-based, and (v) causal-based methods. I analyse and contrast these methods from two primary perspectives: their analytical focus (e.g., whether they prioritise the feature distributions or specific statistical relationships between the features), and their dependency on the unconstrained deep generative model. Finding the best ordering can be a challenge, as different orderings work better in different scenarios. Nevertheless, through an extensive experimental analysis, I show that there are orderings which are reliable in the sense that they provide clear improvements in performance w.r.t. efficacy and detection.

The Linearly-constrained Layer considers the problem of background knowledge alignment only for constraints provided as linear inequalities and restricts the DGMs' output space to coincide with the one defined by such constraints. However, this solution is only available when the knowledge can be captured by linear inequalities, which have very limited expressivity. Since real-world data often involve non-linear relationships among features, e.g., such as those captured using disjoint intervals, it becomes necessary to have a solution for these, more complex, cases. To address this problem and allow for more expressive constraints, in Chapter 5, I propose a novel layer, called the Disjunctive Refinement Layer (DRL), which is the first framework able to constrain any DGM output space according to background knowledge expressed as quantifier-free linear real arithmetic (QFLRA) formulae, which I also refer to as disjunctive inequality constraints. QFLRA formulae can capture any relationship over the features that can be represented as a combination of conjunctions, disjunctions and negations of linear inequalities. Due to their expressivity, QFLRA formulae can define spaces that are not only non-convex but can also be disconnected, unlike linear inequalities, which can only capture convex output spaces. While linear inequalities establish a single lower and upper bound (if existent) for each feature, QFLRA formulae define multiple intervals where the background knowledge holds, each with its own boundaries. This significantly increases the complexity of the problem, especially when modelling the inherent, hidden relations between the features that can be derived from the explicitly-provided constraints. Once built, by definition, DRL (i) guarantees the satisfaction of the constraints, (ii) can be seamlessly added to the topology of any neural model, (iii) allows the backpropagation of the gradients at training time, (iv) performs all the computations in a single forward pass (i.e., no cycles), and (v) given a sample generated by a DGM, it returns a new one that is optimal with respect to the original (intuitively, which minimally differs

from the original sample while taking into account the user preferences on which features should be changed first). Further, I empirically demonstrate that adding DRL to DGMs improves their machine learning efficacy on a range of different scenarios, while also providing inference runtimes comparable to the unconstrained models.

Importantly, through extensive evaluations conducted across Chapters 3 to 5, I show that there is no trade-off between ensuring that the samples generated using my methods satisfy the background knowledge and the sample quality, with the resulting models (i.e., models equipped with LL or DRL) in fact demonstrating significant improvements in quality, according to one of the most important measures used for benchmarking synthesisers, namely machine learning efficacy. Secondly, I show that the standard method, i.e., rejection sampling, for creating sample sets that adhere to the constraints, is sub-optimal as: (i) it often takes longer to be applied at inference time, with some cases proving impossible to apply it to (i.e., when all generated samples violate at least one constraint), (ii) it often results in degraded sample quality according to the machine learning efficacy. Finally, the experiments demonstrate a strong need for methods like LL and DRL. Indeed, standard DGMs generate synthetic datapoints violating the background knowledge more often than expected. In more than half of the evaluated real-world scenarios, the DGMs produced datasets with over 50% datapoints violating the constraints, and in many cases this reached 100%.

Finally, in Chapter 6, I propose PiShield, a PyTorch-based package supporting methods belonging to the two main categories of neuro-symbolic AI: methods able to integrate constraints in the loss function and penalise the neural network models when they violate the constraints, and methods that emerged later and that are able to incorporate constraints directly in the architecture of the neural networks to guarantee their satisfaction. PiShield is the first package designed for seamlessly integrating background knowledge into neural networks via layer-based and loss-based methods. Specifically, these two methods are enforced by two core mechanisms: (i) Shield Layers, which are based on the LL and DRL layers introduced in the previous chapters, guaranteeing the satisfaction of the constraints regardless of the input, and (ii) Shield Losses, which can be added to the training objective of neural networks to penalise outputs that contradict the background knowledge constraints, thus guiding the learning process.

It is important to note that, while the above chapters are dedicated to the latter branch of neuro-symbolic AI methods, the focus of Chapter 6 is on Shield Losses, which belong to the former branch of approaches. First, I show that popular loss-based neuro-symbolic methods, which use constraints to guide learning, become im-

practical when dealing with a large number of propositional logic constraints. To overcome this, I design a memory-efficient, logic-based loss function that effectively penalises non-compliant outputs, significantly improving predictive accuracy, particularly in scenarios where little annotated data is available. In the context of real-world applications, these losses are suitable for complex settings with a large number of constraints. To this end, I demonstrate the capabilities of my proposed approach for the task of multi-label classification in autonomous driving.

By providing accessible implementations of both layer-based mechanisms, for guaranteed constraints' satisfaction, as well as of the memory-efficient loss-based functions, for guiding the learning according to the constraints, PiShield is a versatile tool for applying neuro-symbolic approaches to practical problems. Indeed, a key motivation for creating PiShield was to make my proposed neuro-symbolic methods easily accessible to researchers and practitioners working with real-world applications.

The rest of this thesis is organised as follows:

1. In Chapter 2, in order to place my contributions into context, I provide the background, general notation and concepts relevant to the rest of this thesis. Additionally, I discuss the related work on neuro-symbolic approaches, with an emphasis on deep generative modelling.

2. In Chapter 3, I introduce the Linearly-constrained Layer (LL), the first framework capable of injecting background knowledge expressed as linear inequalities into deep generative models.

3. In Chapter 4, I present different heuristics to construct feature orderings, which determine the order in which LL corrects the features' values.

4. In Chapter 5, I present the Disjunctive Refinement Layer (DRL), which advances the work from the previous two chapters to a more complex scenario involving background knowledge constraints expressed as disjunctions over linear inequalities.

5. In Chapter 6, I introduce a memory-efficient logic-based loss function able to integrate constraints into neural networks, along with PiShield, a package that encapsulates the methods developed in this thesis.

6. In the remaining Chapter 7, I present a summary of this thesis and the findings, along with my perspective on future research directions.

# Chapter 2

# Background

In this Chapter, based on (Stoian et al., 2026), I will present the concepts needed to navigate the later content and related work relevant to the methods I introduce in this thesis. Specifically, Section 2.1 provides the necessary background into neuro-symbolic methods, as well as the real-world scenarios where such methods have been applied or could be applied as a potential future direction. Further, Section 2.2 discusses a specific deep learning task, namely deep generative modelling. In particular, I review the approaches used for tabular data synthesis, which is one of the main real-world applications considered for the methods in this thesis. Finally, in Section 2.3, I conclude with a summary and a discussion on future directions and opportunities for improvement.

## 2.1 Deep Learning with Constraints

In this Section, I review neuro-symbolic methods, which focus on combining deep learning with reasoning based on background knowledge constraints. In particular, I discuss and group such approaches into two broad categories: methods that constrain the architecture of the neural networks, and methods that constrain the loss function. Finally, I discuss why background knowledge is important in real-world applications.

### 2.1.1 Neuro-symbolic Methods

Deep neural networks (DNNs) have shown their strengths in various application domains. However, their data-driven nature restricts their learning capabilities to observed examples, and standard models ignore any available background knowledge. This often leads to them failing to comply with given constraints that define the safe

output space of the model. Addressing this shortcoming, neuro-symbolic AI methods have gained popularity recently by aiming to combine neural networks with the formal reasoning capabilities of symbolic systems (Raedt et al., 2018; d'Avila Garcez et al., 2019; d'Avila Garcez and Lamb, 2023).

Integrating symbolic knowledge improves the learning efficiency and can lead to better performance (e.g., (Giunchiglia et al., 2023b)). Additionally, learning from domain knowledge brings little annotation overhead, as formulating constraints is often less costly than the efforts needed to annotate large datasets. Initially, a variety of neuro-symbolic methods were introduced to integrate such background knowledge constraints into the loss function. Later, a second category of methods emerged, incorporating constraints directly into the network's topology and offering stronger guarantees.

#### 2.1.1.1 Constraining the Neural Network Loss Function

The first major category of neuro-symbolic methods comprises approaches that integrate constraints into the loss function to penalise models when they violate the given constraints (e.g., Diligenti et al. (2012); Minervini et al. (2017); Diligenti et al. (2017b); Donadello et al. (2017); Xu et al. (2018); Fischer et al. (2019); Nandwani et al. (2019); Badreddine et al. (2022); Li et al. (2023); Ahmed et al. (2022c); Stoian et al. (2023)). While these approaches help reduce the incidence of constraint violations, they cannot guarantee their satisfaction. Indeed, the goal becomes an optimisation problem, where the constraints-derived penalty is balanced against the primary task objective. These methods primarily differ in how they translate discrete logical formulae into a differentiable loss term.

A foundational strategy involves using fuzzy or real-valued logic. These methods map logical formulae to a continuous value in $[0, 1]$, which represents a degree of truth. One of the earliest frameworks, Semantic-Based Regularisation (SBR) (Diligenti et al., 2017b), uses fuzzy logic T-norms to convert first-order logic formulae into a differentiable regularisation term that measures the extent of a violation. Similarly, Logic Tensor Networks (LTNs) (Serafini and d'Avila Garcez, 2016) define "real logic" where predicates and functions are mapped to tensors. The satisfaction of a formula is aggregated across the training data to create a loss term that guides the neural network to learn embeddings consistent with the provided knowledge.

Another family of methods approaches the problem from a probabilistic perspective. Instead of measuring a degree of truth, they penalise a model for assigning a low probability to outcomes that satisfy the constraints. DeepProbLog (Manhaeve

et al., 2018) is one such popular method, combining probabilistic logic programming with deep learning. It uses neural networks to provide probabilities for a set of neural predicates, which are then taken as input to a ProbLog (Raedt et al., 2007) program. The gradient from the loss is then backpropagated through the neural predicates, allowing the network to learn directly from the logical constraints. A more direct approach is the Semantic Loss (Xu et al., 2018), which offers a syntax-independent penalty. Specifically, the proposed loss is proportional to the negative logarithm of the probability of generating an output that satisfies the constraints, for any given input. This encourages the network's probabilistic output distribution to assign high likelihood to valid configurations.

There are also methods that use constraints more indirectly to guide the learning process. Rather than adding a penalty to the loss, they use the constraints to actively generate challenging new data. For instance, the method proposed by Minervini and Riedel (2018) improves the logical consistency of natural language inference (NLI) models by first mapping constraints (e.g., that contradiction is a symmetric relation: *if sentence* $s_1$ *contradicts sentence* $s_2$*, then* $s_2$ *contradicts* $s_1$) into a loss function. In turn, this loss function is used to guide a search for adversarial examples that cause the model to violate the constraints. Importantly, when using these examples for training, the method drastically reduces the number of times the model violates the constraints and provides a major increase in predictive accuracy on adversarial datasets.

#### 2.1.1.2 Constraining the Neural Network Architecture

Emerging later, the second category consists of methods that incorporate a given set of constraints directly into the topology of the network (e.g., Giunchiglia and Lukasiewicz (2021); Hoernle et al. (2022); Ahmed et al. (2022b)). These approaches do not compromise on adhering to a strict set of constraints and are thus able to guarantee their satisfaction, which is crucial for safety-critical applications.

Early pioneering work in this area includes the Knowledge-Based Artificial Neural Network (KBANN) (Towell and Shavlik, 1994) and CIL$^2$P (d'Avila Garcez and Zaverucha, 1999). Both frameworks map a set of logical rules into the initial structure of a neural network. As a foundational method in this domain, KBANN directly translates a set of acyclic rules into the initial structure of a neural network. The process involves mapping supporting facts to input neurons, intermediate conclusions to hidden neurons, and final conclusions to output neurons. The connections between these neurons are created to mirror the dependencies in the rule set. For instance,

a rule like $A \wedge B \rightarrow C$, where $A$, $B$, and $C$ are labels, would result in the neurons for $A$ and $B$ having connections to the neuron for $C$. The weights and biases are initialised to closely approximate the behaviour of the AND/OR logical gates. After this rule-based initialisation, standard backpropagation is used to refine the network on training data. One disadvantage is that KBANN only allows for acyclic sets of rules, as cycles would lead to a recurrent and potentially unstable network structure that was not supported by the framework. In order to address this limitation and handle cyclic rules, CIL$^2$P uses a different mapping that creates a recurrent network capable of handling recursion in the rules, and demonstrates that the resulting network can compute the stable model of the corresponding logic program.

More recent approaches build on this principle by designing specialised layers or modules that enforce constraints. An important such work is C-HMCNN (Giunchiglia and Lukasiewicz, 2020), which introduces a new layer that is able to incorporate hierarchical classification constraints into the topology of neural networks. Instead of defining the entire network topology from the constraints, this architecture allows a standard deep learning model (e.g., a convolutional neural network) to perform feature extraction and initial prediction. The unconstrained outputs of this base neural network are then fed into the new layer, placed on top of the model (i.e., as a final layer). This layer is mathematically constructed from the set of logical constraints to act as a projection function, mapping any input prediction vector to the nearest valid vector that satisfies all of the provided constraints. Thus, this method ensures compliance while retaining the powerful representation learning capabilities of modern neural networks. Later, CCN (Giunchiglia and Lukasiewicz, 2021) was introduced to generalise C-HMCNN to explore more complex logical relations between classes in a hierarchical structure.

Yet another work able to guarantee the satisfaction of the constraints is the neuro-symbolic constrained structured predictor (referred to as NESTER) (Dragone et al., 2021), which integrates a constraint programming solver into the architecture in a differentiable manner. NESTER works in two stages: first, the neural network produces an initial, unconstrained prediction; second, this prediction is fed as input to the constraint programming solver, which enforces the constraints to find a compliant final output. As the entire architecture is differentiable end-to-end, the gradients from the constraint satisfaction mechanism (i.e., how much the initial prediction had to be changed) can flow back and train the neural network to produce better, more compliant initial predictions over time. This hybrid approach is versatile, capable of handling both hard and soft constraints over categorical and numerical variables, with

the authors presenting an empirical evaluation on tasks such as handwritten equation recognition.

While most of such works require that the constraints are provided as propositional constraints, there are also recent methods where the constraints can be on numerical outputs. For instance, MultiPlexNet (Hoernle et al., 2022) supports constraints written as quantifier-free linear arithmetic formulae over the rationals. It first transforms the set of constraints into disjunctive normal form (i.e., a disjunction of conjunctions). Then, for each disjunct (a conjunction of constraints), a separate transformation is added to the network's output layer. The final output is then a "multiplexed" combination of these transformations, which is guaranteed to satisfy at least one of the disjuncts, and, thus, the entire original formula.

More recently, an alternative method (van Krieken et al., 2023), also capable of guaranteeing compliance with the constraints, proposed using neural networks for performing approximate inference in polynomial time to address the scalability problem of probabilistic neuro-symbolic learning frameworks such as DeepProbLog (Manhaeve et al., 2018).

For a more in-depth survey of methods combining deep learning with logical constraints, I refer the reader to Giunchiglia et al. (2022), and for a broader overview of neuro-symbolic AI, to d'Avila Garcez et al. (2019).

### 2.1.2 Real-World Applications

Deep learning models are increasingly deployed in high-stakes, real-world applications where errors can have significant safety, financial, or ethical consequences. However, standard deep learning models often struggle with data scarcity and an inability to incorporate explicit background knowledge, leading to unpredictable behaviour that is inconsistent with the available knowledge. As discussed in the previous Section, neuro-symbolic methods have emerged as a promising solution to these challenges. In this thesis, I will explore the critical role of background knowledge constraints in two such domains: autonomous driving and tabular data synthesis, with a primary emphasis on the latter.

Autonomous driving is one of the most representative examples of a safety-critical application. Thanks to its success in finding patterns in raw data and turning them into accurate predictions, deep learning has been central to the development of the autonomous driving systems (Grigorescu et al., 2020; Huang and Chen, 2020). However, the limitations of purely data-driven models are a major obstacle to achieving full autonomy (Hacohen et al., 2022). A self-driving vehicle must not only perceive

its environment but also operate within a strict set of rules: from traffic laws to the laws of physics. Integrating such fundamental background knowledge directly into the model can lead to more robust, reliable, and data-efficient systems, a key motivation for applying neuro-symbolic techniques in this field, especially with the recent developments highlighting a positive impact particularly in scenarios where little annotated data is available (see, e.g., (Li and Srikumar, 2019; Fischer et al., 2019; Marra et al., 2019)).

While autonomous driving highlights the need for constraints in physical systems, tabular data synthesis presents an equally important challenge from the perspective of logical and semantic integrity. The generation of high-quality synthetic tabular data has become increasingly valuable across numerous sectors, including finance, healthcare, and manufacturing. Fuelled by advances in deep generative modelling, synthesisers are now used to augment datasets in low-resource scenarios (Chen et al., 2019; Zhao et al., 2021; Jia et al., 2024), create privacy-preserving datasets for research (Park et al., 2018; Vero et al., 2024), and ultimately improve the performance of downstream machine learning tasks such as disease diagnosis or credit scoring (Kim et al., 2021, 2023).

Traditionally, the success of generative models was often measured by metrics such as downstream machine learning efficacy or statistical fidelity (e.g., (Xu and Veeramachaneni, 2018)). However, these evaluations frequently overlook a fundamental requirement for creating trustworthy data: background knowledge alignment. Real-world tabular data is almost always governed by implicit, domain-specific constraints. When a synthesiser violates these rules, the generated data becomes logically inconsistent, unreliable, and potentially harmful if used for decision-making. For example, in healthcare, synthetic patient data must adhere to physiological constraints (e.g., ensuring the minimum blood pressure value recorded for a patient is not higher than the maximum value). In finance, generated market data must comply with known economic principles, and synthetic loan applications must follow logical rules (e.g., loan term cannot be negative). Similarly, in resource management, synthetic inventory data must not violate logistical constraints, such as production capacity limits or non-negative stock levels.

Clearly, data that violate such background knowledge are inherently flawed. Thus, ensuring that synthetic data aligns with the constraints is not just a desirable property but a necessity for its trustworthy application. In this thesis, I focus on developing methods that explicitly address this challenge, aiming to create generative models

which produce data that are not only useful for real-world downstream applications but also realistic with respect to the background knowledge constraints.

## 2.2 Deep Generative Modelling with Constraints

In recent years, the synthesis of realistic tabular data has emerged as a critical research area, driven by the need for privacy-preserving machine learning, robust AI benchmarking, and data augmentation in high-stakes applications such as finance and healthcare. These diverse needs led to not only a proliferation of different generative models, but also to an explosion of diverse evaluation metrics whose aim is to evaluate whether a model is able to address all the requirements of the given use case. However, the sheer diversity of these metrics has made it increasingly difficult to assess and compare different synthesisers in a standardised manner.

The literature review I present in this Section takes a unique stance and analyses models and evaluation methods alike from the point of view of the user, categorising them on the ground of the requirements they primarily address and measure: machine learning efficacy, background knowledge alignment, statistical fidelity, and privacy. The analysis thus serves as a structured resource for both newcomers and experienced practitioners seeking a formal evaluation framework for tabular data synthesis. To this end, I first give an overview of the tabular data synthesis problem (Section 2.2.1) and its requirements (Section 2.2.2). Then, I examine in detail existing methods along two dimensions (Sections 2.2.2.1-2.2.2.4), grouping them (i) by the requirement they primarily focus on and (ii) by the model architecture type they are based on. For each requirement, I detail the relevant evaluation protocols and metrics. A summary of the surveyed approaches can be found in Table 2.2, which also gives an overview of the characteristics for each model, such as their ability to handle high-dimensional and mixed data types, as well as their sample generation times. Lastly, I review other deep generative models originally developed for other fields (such as natural language processing and computer vision), which can be adapted to tabular data synthesis (Section 2.2.3).

### 2.2.1 Synthesising Tabular Data

First, I provide the preliminaries and notation used in the rest of this Chapter, but also in the later Chapters of this thesis, for facilitating the discussion on synthesising tabular data using deep generative models and on the desiderata for these data.

In the tabular data synthesis field, a tabular dataset $\mathcal{D}$ is a set of $N$ independent and identically distributed (i.i.d.) samples $\mathbf{x}$ drawn from an unknown distribution $p_X$ over $X \in \mathbb{R}^D$, where $D$ is the number of features. Each feature of $\mathcal{D}$ uniquely corresponds to a variable in the finite set $\mathcal{X} = \{x_k \mid k = 1, \ldots, D\}$. Further, the features can be of mixed types: they can be continuous (real-valued), discrete[1], or even a mixture of the previous two types, and their values might be missing for some datapoints. Typically, each $\mathbf{x}$ in $\mathcal{D}$ represents a row in the dataset, while each feature represents a column in $\mathcal{D}$. To ease the notation, I will assume that the $k$-th entry of $\mathbf{x}$ corresponds to the value of $x_k$ at that datapoint of dataset $\mathcal{D}$.

One of the many challenges of this field is given by the fact that the domain $\mathbb{D}$ of each feature can be very different from one another. Common domains are the following: binary domains, i.e., $\mathbb{D} = \{0, 1\}$, categorical domains, e.g., $\mathbb{D} = \{\textit{Red}, \textit{Yellow}, \textit{Blue}\}$, and numerical domains, i.e., $\mathbb{D} \subseteq \mathbb{R}$. Given a dataset $\mathcal{D}$ with $N > 1$, it is always possible to partition its instances in order to obtain a *training set* used to train a machine learning model, and a *test set* used to test a machine learning model.

The goal of standard deep generative modelling is to learn from $\mathcal{D}$ the parameters $\theta$ of a generative model such that the model distribution $p_\theta$ approximates $p_X$. While synthesising tabular data is a long-standing problem (Reiter, 2005), in this thesis, I focus on methods that use deep generative models, i.e., models based on deep neural networks.

### 2.2.2 Requirements over Synthetic Data

Different use cases can lead to additional requirements, which, in turn, can lead to different goals for the generative task. Thus, the above goal for standard generative modelling would become insufficient if the aim is to satisfy these requirements. In what follows, I will review and group the related works based on the requirements listed below:

1. ***Machine learning efficacy.*** *Synthetic data should yield similar predictive performance to real data when used to train machine learning models for the same task, such as classification or regression.* For instance, given a healthcare dataset like WiDS (Matthys et al., 2021), which contains medical records of patients from the first 24 hours of intensive care, and where the target column specifies whether a patient has been diagnosed with diabetes mellitus, training

[1]Note that I refer to discrete features to be both the numerical discrete and the categorical features.

a classifier on synthetic data should yield comparable (or better) performance on the real test set to that obtained by training the classifier on real data of the same size. Hence, given a machine learning model $m$ and a predictive performance metric $r$ (e.g., accuracy), maximising machine learning efficacy entails learning from $\mathcal{D}$ the parameters $\theta$ of a generative model such that training $m$ on the dataset $\mathcal{D}'$ sampled from $p_\theta$ leads to the best score according to $r$.

2. ***Background knowledge alignment.*** *Synthetic data should align with any known (user-provided) domain-specific knowledge.* The knowledge can be expressed in multiple ways: simply as constraints on the range of values that a single feature can assume, or as more complex constraints capturing relations between the features. Continuing the example above, if the *minimum* and the *maximum haemoglobin level* are two of the features recorded for each patient, then clearly the real data do not contain any records for which the value of the *minimum* is higher than the value of the *maximum level*. This type of constraint is easily captured using a linear inequality. Hence, given a set of constraints expressing some background knowledge about the sample space of $p_X$, i.e., stating which samples are admissible and which are not, the goal is to learn the parameters $\theta$ of a generative model such that (i) the model distribution $p_\theta$ approximates $p_X$, and (ii) the sample space of $p_\theta$ is compliant with the constraints.

3. ***Statistical fidelity.*** *Synthetic data should preserve the statistical properties of the real data.* Continuing with the same example, if the aim is to maximise the fidelity on the WiDS dataset, then the distribution of the synthetic values of the *minimum haemoglobin level* feature should resemble the real distribution of the same feature. Hence, in this case, the goal is to learn parameters $\theta$ such that (i) the marginals of $p_\theta$ are as similar as possible to the corresponding marginals of $p_X$, and (ii) the joint distribution $p_\theta$ closely follows $p_X$.

4. ***Privacy.*** *Synthetic data should present minimal risk of disclosing sensitive attributes and of re-identifying individuals from the real data.* In the example above, the WiDS dataset contains sensitive features such as: the patient's *age*, whether the patient had an *elective surgery*, and whether the patient received a *leukaemia* diagnosis. If a rare combination of values for these features is retained in the synthetic data, then a patient might be identified, hence failing to meet privacy requirements. Hence, in this case, the goal is to learn the parameters $\theta$ such that it is impossible to sample a data point from $p_\theta$ that would allow for the re-identification (or for disclosing sensitive attributes) of a real data point.

Table 2.1: Overview of the evaluation metrics typically used for the four synthetic tabular data requirements. The formula and terminology definitions for each metric are provided in Appendix G.

| **Requirement** | **Evaluation Metric** | |
|---|---|---|
| Machine learning efficacy | Accuracy<br>F1-score<br>Weighted F1-score<br>Root mean square error (RMSE)<br>Mean absolute error (MAE)<br>Explained variance | |
| Alignment | Constraint violation rate (CVR)<br>Constraint violation coverage (CVC)<br>Sample-wise constraint violation coverage (sCVC) | |
| Fidelity | Feature-wise | Wasserstein distance<br>Kolmogorov-Smirnov test statistic<br>Jensen-Shannon divergence<br>Total variation distance |
| | Pair-wise | RMSE differences in mutual information<br>MAE differences in mutual information<br>Correlation difference |
| | Joint | Likelihood fitness score |
| Privacy | AUCROC (for membership attacks)<br>Precision (for attribute disclosure attacks)<br>Recall/Sensitivity (for attribute disclosure attacks)<br>Distance to the closest record (DCR)<br>Nearest neighbour distance ratio (NNDR)<br>k-anonymity | |

In what follows, I group existing deep generative modelling methods for synthesising tabular data by the requirement they primarily address. Prior surveys on tabular data synthesis generally compare methods by the architecture-specific properties of the models. For example, both Borisov et al. (2024) and Davila R. et al. (2025) group methods mainly by their architectures, while also describing the necessary data transformation and regularisation techniques. Lautrup et al. (2024) provide an overview of the models proposed in the field and the metrics used to evaluate them, while not taking this requirement-centric perspective which helps both newcomers and seasoned practitioners and researchers in the field to find the perfect model and evaluation protocol for their needs.

For each of the four requirements above, Table 2.1 lists the metrics that are commonly used to evaluate the generated data. Further, in Appendix G, I provide the formula for each of these metrics, along with the terminology definitions.

#### 2.2.2.1 Machine Learning Efficacy

Determining whether synthetic data can replace real training data in downstream tasks has been the most common evaluation approach (e.g., Choi et al. (2017); Park

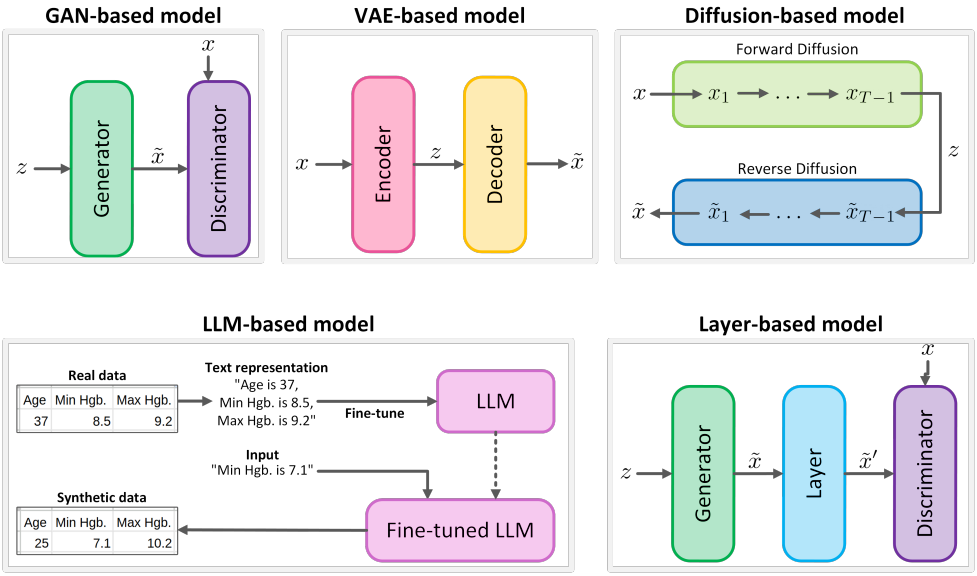



Figure 2.1: Visualisation of common model types used for synthesising tabular data. Here, $\mathbf{x}$, $\mathbf{z}$, and $\tilde{\mathbf{x}}$ denote a real sample, noise sample, and synthetic sample, respectively; $\tilde{\mathbf{x}}'$ is a sample modified by layer-based models; $x_i$ is the sample at step $i$ of forward diffusion (for $i$ in $\{1, \ldots, T\}$, where $T$ is the maximum number of diffusion steps); and $\tilde{x}_i$ is the sample at step $i$ of reverse diffusion.

et al. (2018); Kotelnikov et al. (2023); Borisov et al. (2023)). The main reason for this is that synthetic data are often generated to augment or create datasets for real-world applications, where good predictive performance is crucial, e.g., predicting diabetes diagnoses from intensive care unit data (Matthys et al., 2021). Therefore, machine learning efficacy still remains a key requirement for synthetic data and benchmark for comparing synthesisers.

The most common protocol used to evaluate the machine learning efficacy of generative models is the *Train on Synthetic, Test on Real* (TSTR) Protocol (see, e.g., Esteban et al. (2017)). The procedure is to train a suite of machine learning models (e.g., a support vector machine, XGBoost) on the synthetic data, and then test their performance on a real test set, reporting the performance using different metrics on the ground of the task defined by the dataset of interest: for classification datasets standard metrics include accuracy (Eq. G.1) and F1-score (Eq. G.2), and weighted F1-score (Eq. G.3), while for regression datasets mean absolute error (Eq. G.5), root mean square error (Eq. G.4) and explained variance (Eq. G.6) are commonly used. Note that a dataset can be for either a classification or a regression task, depending on whether its target column (i.e., the feature whose values are to be predicted by the model) is a discrete or a continuous variable, respectively.

In the context of this thesis, machine learning efficacy is formally defined as the predictive performance of a (suite of) machine learning model(s) trained on synthetic data and tested on a real test dataset, benchmarked against the generalisation performance of the same model(s) trained on the original real dataset. To ensure a consistent and robust evaluation throughout this thesis, and to acknowledge the *No Free Lunch Theorem* (Wolpert and Macready, 1997), which states that no single machine learning algorithm is universally superior across all datasets and tasks, I describe here the downstream setting for measuring machine learning efficacy (which is consistent across all of my proposed methods for synthesising tabular data, i.e., in Chapters 3-5). Specifically, rather than relying on a single model, I use the TSTR protocol to measure the machine learning efficacy with a diverse suite of standard classification and regression algorithms (as detailed in Sections 3.4.1 and 5.4.1), namely (i) for binary classification datasets, I use Decision Tree (Wu et al., 2008), AdaBoost (Schapire, 2013), Multi-layer Perceptron (MLP) (Haykin, 1994), Random Forest (Ho, 1995), XGBoost (Chen and Guestrin, 2016), and Logistic Regression (Cox, 1958) classifiers, (ii) for multi-class classification datasets, I use Decision Tree, MLP, Random Forest, and XGBoost classifiers, and (iii) for regression datasets, I use MLP, XGBoost, Random Forest regressors and linear regression. By averaging the predictive performance (which I measure via F1-score, weighted F1-score, and Area Under the ROC Curve for the classification datasets, and the previously mentioned standard error metrics for the regression datasets) across this diverse set of algorithms, I ensure that the reported machine learning efficacy reflects the utility of the synthetic data itself, rather than the suitability of one specific downstream algorithm.

It is important to note that the literature on tabular data synthesis lacks a fully standardised terminology for this metric. Consequently, the related works reviewed here (and summarised in Table 2.2) often use different terminology to describe the same underlying concept, including: downstream utility, machine learning utility, predictive performance, and efficiency, TSTR performance (naming the specific evaluation methodology used to calculate the efficacy). Regardless of the specific terminology used by these works, all these terms align with the rigorous definition of machine learning efficacy established here: measuring the performance of models trained on synthetic data when evaluated on real test sets.

**GAN-based models.** Many of the works surveyed here are based on the original Generative Adversarial Network (GAN) (Goodfellow et al., 2014), which relies on two neural networks: (i) a *generator*, which given a noise vector returns a synthetic datapoint and hence learns the real data distribution, and (ii) a *discriminator*, which

given a data point (either real or synthetic) classifies it as either real or synthetic and hence learns to distinguish between synthetic and real data. The GAN jointly trains these two models in an adversarial framework, where the generator's objective is to produce samples that confuse the discriminator. See the first schema from the left in Figure 2.1 for a simple abstraction of GAN-based models. CWGAN: Building on the seminal Wasserstein GAN (WGAN) (Arjovsky et al., 2017), which used the Wasserstein distance as a loss function for a GAN model, Engelmann and Lessmann (2021) proposed CWGAN as an oversampling framework for class balancing. CWGAN generates synthetic samples for under-represented classes, augmenting the training set. Results indicate that oversampling is beneficial primarily for strongly non-linear datasets. OCT-GAN: Proposed by Kim et al. (2021), it incorporates neural ordinary differential equations (neural ODEs) in the design of its GAN-based generator and discriminator through the introduction of a new layer that allows for the extraction of a sequence of hidden vector representations given an input sample, i.e., the hidden evolution trajectory of the sample. The trajectory is used to help the discriminator decide whether a sample is real or synthetic. While improving machine learning efficacy, the usage of ODEs makes OCT-GAN slower than most GAN-based models.

**GAN and BN-based models.** GANBLR: Zhang et al. (2021) construct the generator and discriminator as Bayesian Networks (BNs), which are able to encode known feature interactions. The background knowledge inclusion leads to better machine learning efficacy than standard GANs.

**Diffusion-based models.** TabDDPM: Motivated by the their success in computer vision, Kotelnikov et al. (2023) introduced a diffusion-based model for synthesising tabular data. Popular for its high machine learning efficacy results, TabDDPM is able to model both discrete and continuous features, but separately, by using multinomial (Hoogeboom et al., 2021) and Gaussian diffusion models (Sohl-Dickstein et al., 2015), respectively. For a simple abstraction on how a diffusion model can be used to generate data, see the top right schema in Figure 2.1. STaSy: Proposed by Kim et al. (2023), STaSy is built on a score-based generative model (SGM), which was shown to be a competitive alternative to diffusion-based models in (Ho et al., 2020). Contrarily to TabDDPM, STaSy estimates the score function (gradient of log-probability) instead of explicitly modelling the forward and reverse Markov processes. As the loss is difficult to train, STaSy uses a self-paced learning strategy, which starts with a reduced set of training data—yielding a stable, low loss—and gradually expands to the full dataset. Evaluated on 15 real-world datasets against 7 baselines (mostly

GAN-based models), STaSy achieves high machine learning efficacy. Due to the forward and backward passes, both STaSy and TabDDPM suffer from slow sampling times. Moreover, the base SGM model is known to be unstable in high-dimensional settings. This can limit its suitability for real-world datasets which may often have more features than 58, the largest number of dimensions in the datasets considered for evaluating STaSy, as we can also see in Table 2.2. CoDi: Motivated by the difficulty of modelling discrete features, Lee et al. (2023) proposed a framework comprising co-evolving diffusion models that separately handle continuous and discrete features as in TabDDPM (Kotelnikov et al., 2023), with the difference that the two models condition on each other during training. At each training step, the discrete (resp., continuous) diffusion model takes the perturbed sample from the continuous (resp., discrete) one as input, and both models are conditioned on both samples from the previous step during the denoising process. Slower than most GAN-based models (excluding OCT-GAN), CoDi is faster than STaSy and TabDDPM, while also obtaining higher machine learning efficacy on mixed types datasets.

**LLM-based models.** GReaT (Borisov et al., 2023) represents an answer to the abundance of works that convert tabular data into numerical representations thus losing the contextual connections between the features, which very often carry semantic meaning. To address this, GReaT takes the following steps, which are also illustrated in Figure 2.1. First, the tabular data undergoes a textual encoding process, applied specifically on the features' naming and values (provided in a given order), to convert them into text. Next, a feature order permutation step is applied. The resulting sentences are then used to fine-tune the pretrained self-attention-based large language models (LLMs) (Vaswani et al., 2017). At sampling time, either a single feature name or arbitrary feature-value pairs are given as input to the LLM, which then completes them (note that this allows for arbitrary conditioning). Using this approach, GReaT outperforms most of the baselines w.r.t. machine learning efficacy, although its data sampling time is notably high. TabuLa: To address this problem, Zhao et al. (2025) introduce a method to reduce the length of the generated token sequences. Unlike GReaT, which relies on large language models pre-trained on natural language processing tasks, TabuLa starts with a randomly initialised model and iteratively fine-tunes it on tabular data synthesis tasks. Another key distinction is that TabuLa does not use GReaT's feature permutation mechanism. Instead, it targets scenarios where arbitrary feature conditioning is not needed and assumes that the feature values appear in a consistent order across all data points. To this end, the authors tokenise the dataset feature-wise, ensuring that each generated token can be mapped to a specific

feature based on its absolute position in the row. Specifically, for each feature, they determine the longest token sequence across the dataset and pad all sequences (corresponding to that feature) to this length during training. Using this approach, TabuLa manages to significantly reduce the training time compared to GReaT. However, its sample generation time is still considerably higher than GAN-based models such as CTGAN or TableGAN.

**Energy-based models.** TabPFGen (Ma et al., 2023) is a generative model that uses a pretrained network as an energy-based model for data augmentation and class balancing. Although its capabilities are limited to low-dimensional inputs (it was tested on datasets with maximum 10 features), TabPFGen demonstrates good machine learning efficacy performance in the data augmentation task by simply using a pretrained model compared to specialised GAN-, VAE-, and diffusion-based models.

**Transformer and GAN-based models.** TabTransGAN (Zhang et al., 2025) achieves performance comparable to GReaT by combining the strengths of Transformer- (Vaswani et al., 2017) and GAN-based models, which allows it to model both contextual and structural relationships across all features. Specifically, TabTransGAN adopts a GAN architecture in which the generator is based on CTGAN, while the the discriminator replaces the standard design with a Transformer architecture. This enables it to better capture both feature-wise distributions and dependencies between the features, leading to more accurate discrimination between synthetic and real data.

#### 2.2.2.2 Background Knowledge Alignment

Brought to the attention of the research community more recently (Chen et al., 2019) than the other requirements, background knowledge alignment is an important condition for synthetic data when evaluating their realism. Specifically, alignment requires that synthetic data should comply with user-provided knowledge, which is an important condition in practice, as many real-world applications have available domain-specific knowledge which must be satisfied. For example, synthetic patient data in healthcare must adhere to physiological constraints, ensuring a recorded minimum value for an indicator (like blood pressure or haemoglobin level) is not higher than the maximum value recorded. Similarly, in resource management, synthetic inventory and demand data must not violate logistical constraints, such as production capacity limits or non-negative stock levels.

As I will present later, in my work (Stoian et al., 2024a), I have proposed three different metrics to evaluate alignment, namely: (i) the Constraint Violation Rate

(CVR) (Eq. G.7), which computes the percentage of samples that do not satisfy at least one of the constraints in the available set of constraints, (ii) Constraint Violation Coverage (CVC) (Eq. G.8), computing the percentage of constraints that have been violated at least once by the sample set, and (iii) the sample-wise Constraint Violation Coverage (sCVC) (Eq. G.9) which determines the average percentage of samples violating each of the constraints. Out of these three metrics, CVR has also been used in (Jia et al., 2024), albeit called feasibility rate there.

**GAN and AE-based models.** ITS-GAN: proposed for tabular data augmentation ITS-GAN (Chen et al., 2019) is an early method advocating for the use of background knowledge. The knowledge it can express is of two types: (i) rules stating that the value of one feature uniquely determines the value of another (e.g., the feature *Position* determines the feature *Salary*), and (ii) rules stating that a specific value for a set of features determines the specific value for another (e.g., if *Position* = *"CEO"* then *Salary* = *"2M"*). To encode the first types of rules, the authors train on the real data an autoencoder (AE) for each rule to output the value of the dependent features given the values of the independent ones. Then, at training time, the GAN generator is trained to minimise the difference between its output and one of the AEs, while also being penalised if it violates any rule of the second type. Additionally, the discriminator receives as input the difference between the AE's outputs and the synthetic samples, thus being able to use this information to assess the sample.

**GAN and BN-based models.** $C^3$-TGAN: Similarly to GANBLR, $C^3$-TGAN (Han et al., 2023) uses Bayesian networks to capture background knowledge. However, while GANBLR models only discrete features, $C^3$-TGAN can handle both discrete and continuous features by following the CTGAN (Xu et al., 2019) approach. To incorporate explicit attribute correlations and property constraints from background knowledge, it represents constraints as control vectors (similar to the conditional vectors in CTGAN) and uses them to guide training, ensuring better alignment.

**VAE and GNN-based models.** GOGGLE: Proposed by Liu et al. (2022), it is a different approach which integrates knowledge about pairwise feature dependencies by encoding them in a graph where each feature is a node and an edge exists if a relation between the two features is known. New relations can be learnt using a message passing mechanism from graph neural networks (GNNs), which is jointly trained with a VAE-based architecture, where only relationships prioritised by the learned graph influence feature generation. Importantly, the background knowledge can take various forms, but it always encodes properties for the graph structure. For example, they allow for encoding sparsity (i.e., requiring that the features depend

on small subsets of other features) and graph connectivity constraints (i.e., allowing for penalising learned graphs with large differences in the number of connections each feature has). Due to its relational learning graph-based component, it might be difficult to scale GOGGLE to datasets of higher dimensionality.

**Diffusion-based models.** TDGGD: The first framework encoding background knowledge about feature relations into diffusion-based models is TDGGD (Jia et al., 2024). The relations can express lower and upper bounds over single features or their sum. Rather than explicitly modelling the relations, TDGGD models whether the features are part of such relations or not, which only gives an indication of hidden relations between the columns to the model.

**My frameworks: Layer-based constrained deep generative models.** In order to put my work in context and offer a comparison against the existing relevant works on synthesising tabular data using deep generative models, I will briefly describe here two of the frameworks that I developed, namely (Stoian et al., 2024a) and (Stoian and Giunchiglia, 2025), and that I will later present in detail as part of this thesis. DGM+LL (Stoian et al., 2024a) is the first method to constrain deep generative models and guarantee the constraints satisfaction. The constraints can capture any set of linear inequalities over the feature space. Differently from the methods surveyed above, my approach relies on a new layer, which I call the Linearly-constrained Layer (LL), that is added immediately after the sample output layer. The layer restricts the output space of the model to coincide with the space defined by the constraints and can be added on top of any deep generative model, with the resulting models called DGM+LL models. As the layer is differentiable and acyclic, it can be added both at inference and training time, while also improving the machine learning efficacy of the model. DGM+DRL (Stoian and Giunchiglia, 2025): I developed this layer-based model in order to capture constraints as expressive as disjunctions over linear inequalities, which allow for expressing relations like "if the sum of two features is greater than 10 then the difference of other two features should be lower than 5". Allowing for modelling non-convex and even disconnected output spaces, I refer to the new layer as the Disjunctive Refinement Layer (DRL). Further, a DGM constructed using DRL is called DGM+DRL. Just as DGM+LL, DGM+DRL is able to guarantee alignment with the background knowledge, while also improving machine learning efficacy across all considered baselines. For a visual representation on how the layer-based methods can be integrated into GAN models, see the bottom right schema in Figure 2.1.

Table 2.2: Comparison of the reviewed methods for synthesising tabular data. For each work, I report (i) the requirement type it primarily focuses on, (ii) the model type, (iii) whether the model allows for mixed data types (marked with "✓" if so, and with "✗" otherwise), (iv) the maximum number of features used for evaluation (indicating whether the model can handle high-dimensionality data), (v) the model's data generation time relative to the other models. In the last four columns, I indicate with "✓" (resp., "✗") the requirements that were (resp., were not) addressed, and with "•" the requirements whose resolution depends on the base DGM model upon which the Layer-based methods are built. Specifically, these requirements are: (vi) machine learning efficacy, (vii) background knowledge alignment, (viii) statistical fidelity, and (ix) privacy. Note that "—" indicates that data are not available, and "✓✓" indicates that the method provides guarantees that the domain-specific constraints are satisfied.

| **Model** | **Primary Requirement** | **Model Type** | **Mixed Data** | **Max.# Features** | **Generation Time** | **Efficacy** | **Alignment** | **Fidelity** | **Privacy** |
|---|---|---|---|---|---|---|---|---|---|
| CWGAN Engelmann and Lessmann (2021) | Efficacy | GAN | ✓ | 36 | — | ✓ | ✗ | ✗ | ✗ |
| OCT-GAN Kim et al. (2021) | | GAN | ✓ | 59 | medium | ✓ | ✗ | ✗ | ✗ |
| GANBLR Zhang et al. (2021) | | GAN & BN | ✗ | 55 | — | ✓ | ✗ | ✗ | ✗ |
| TabDDPM Kotelnikov et al. (2023) | | Diffusion | ✓ | 51 | slow | ✓ | ✗ | ✓ | ✗ |
| STaSy Kim et al. (2023) | | Diffusion | ✓ | 58 | slow | ✓ | ✗ | ✗ | ✗ |
| CoDi Lee et al. (2023) | | Diffusion | ✓ | 31 | medium | ✓ | ✗ | ✗ | ✗ |
| GReaT Borisov et al. (2023) | | LLM | ✓ | 47 | slow | ✓ | ✓ | ✗ | ✗ |
| TabuLa Zhao et al. (2025) | | LLM | ✓ | 55 | slow | ✓ | ✗ | ✗ | ✗ |
| TabPFGen Ma et al. (2023) | | Energy-based model | ✗ | 77 | — | ✓ | ✗ | ✗ | ✗ |
| TabTransGAN Zhang et al. (2025) | | Transformer & GAN | ✓ | 59 | — | ✓ | ✗ | ✗ | ✓ |
| ITS-GAN Chen et al. (2019) | Alignment | GAN & AE | ✓ | 45 | — | ✗ | ✓ | ✓ | ✗ |
| $C^3$-TGAN Han et al. (2023) | | GAN & BN | ✓ | 54 | — | ✓ | ✓ | ✓ | ✗ |
| GOGGLE Liu et al. (2022) | | VAE & GNN | ✓ | 168 | medium | ✓ | ✓ | ✗ | ✗ |
| TDGGD Jia et al. (2024) | | Diffusion | ✗ | 44 | — | ✓ | ✓ | ✗ | ✓ |
| DGM+LL Stoian et al. (2024a) | | Layer-based constrained model | ✓ | 109 | fast | ✓ | ✓✓ | • | • |
| DGM+DRL Stoian and Giunchiglia (2025) | | Layer-based constrained model | ✓ | 64 | fast | ✓ | ✓✓ | • | • |
| TGAN Xu and Veeramachaneni (2018) | Fidelity | GAN | ✓ | 55 | medium | ✓ | ✗ | ✓ | ✗ |
| CTGAN Xu et al. (2019) | | GAN | ✓ | 785 | fast | ✓ | ✗ | ✓ | ✗ |
| CTAB-GAN Zhao et al. (2021) | | GAN | ✓ | 55 | medium | ✓ | ✗ | ✓ | ✗ |
| CTAB-GAN+ Zhao et al. (2022) | | GAN | ✓ | 55 | medium | ✓ | ✗ | ✓ | ✓ |
| TVAE Xu et al. (2019) | | VAE | ✓ | 785 | fast | ✓ | ✗ | ✓ | ✗ |
| Neural Spline Flows Durkan et al. (2019) | | Normalising Flows | ✗ | 50 | — | ✗ | ✗ | ✓ | ✗ |
| FinDiff Sattarov et al. (2023) | | Diffusion | ✓ | 84 | slow | ✓ | ✗ | ✓ | ✓ |
| AutoDiff Suh et al. (2023) | | Diffusion & AE | ✓ | 61 | slow | ✓ | ✗ | ✓ | ✗ |
| TabSyn Zhang et al. (2024) | | Diffusion & VAE | ✓ | 48 | medium | ✓ | ✗ | ✓ | ✗ |
| medGAN Choi et al. (2017) | Privacy | GAN & AE | ✗ | 1071 | fast | ✗ | ✗ | ✗ | ✓ |
| IT-GAN Lee et al. (2021) | | GAN & AE | ✓ | 59 | slow | ✓ | ✗ | ✗ | ✓ |
| TableGAN Park et al. (2018) | | GAN | ✓ | 32 | medium | ✗ | ✗ | ✗ | ✓ |
| PATE-GAN Jordon et al. (2019) | | GAN | ✓ | 617 | fast | ✗ | ✗ | ✗ | ✓ |
| CuTS Vero et al. (2024) | | GAN | ✗ | 20 | — | ✗ | ✓ | ✗ | ✓ |

#### 2.2.2.3 Statistical Fidelity

While statistical fidelity is not a good predictor of downstream task performance (Hansen et al., 2023), it can provide useful observations on how the synthetic data compares to the real data. Fidelity was indeed one of the requirements often measured in early works on synthesising tabular data.

Being the longest-standing requirements, a large number of methods were developed evaluating the statistical fidelity of synthetic data w.r.t. real data. In this Chapter, I will focus only on quantitative evaluations when reviewing the relevant work. These can be categorised into (i) feature-wise, (ii) pair-wise, and (iii) joint evaluations.

**Feature-wise evaluation**: compares synthetic and real data feature by feature, using different metrics based on feature type. For continuous features, Wasserstein distance (Eq. G.10) and the Kolmogorov-Smirnov test statistic (Eq. G.11) are common, while Jensen-Shannon divergence (Eq. G.12) or total variation distance (Eq. G.13) are common metrics for discrete features. The difference in treatment is supported empirically in (Zhao et al., 2021), which found that Jensen-Shannon divergence is unstable for measuring the statistical similarity between continuous features' distributions, especially with no overlap between synthetic and real data.

**Pair-wise evaluation**: investigates how well relationships between pairs of features are preserved in the synthetic data w.r.t. the real data. Common metrics include: (i) root mean squared error (RMSE) (Eq. G.14) and mean absolute error (MAE) (Eq. G.15) differences in mutual information between pairs of features, and (ii) correlation difference (Eq. G.16), using Pearson's coefficient for continuous pairs, Theil's uncertainty coefficient or the total variation distance for discrete pairs, and the correlation ratio between discrete-continuous pairs of features.

**Joint evaluation**: compares the joint distribution of the synthetic samples against the real samples' distribution. It is more difficult to determine how similar are two (potentially high-dimensional) joint distributions. Thus, such an analysis is often conducted in a controlled environment, e.g., on simulated data, rather than real-world data. Relevant evaluation methods include measuring the likelihood fitness score (Eq. G.17), separately for real and synthetic data, as in (Xu et al., 2019).

**GAN-based models.** TGAN (Xu and Veeramachaneni, 2018) is an early GAN-based model that proposes using long short-term memory networks with an attention mechanism to synthesise data sequentially, feature by feature, in order to model tabular data of mixed types. Comparing with conventional statistical models and showing better fidelity performance, TGAN contributed to the popularity of the GAN-based

approaches for tabular data. CTGAN (Xu et al., 2019): focused on better modelling the types of the variables in an effort to increase fidelity. Indeed, the authors propose a new preprocessing method, which is column-type-specific and models discrete features in the continuous space via Gumbel-Softmax transformations, while it employs mode-specific normalisation via a variational Gaussian mixture model applied for each continuous features. Thanks to these transformations, CTGAN is able to avoid mode collapse, which often burdens other GAN-based models, such as medGAN (Choi et al., 2017) and TableGAN (Park et al., 2018). CTAB-GAN (Zhao et al., 2021): Building upon CTGAN, it enhances statistical fidelity by addressing data imbalance and long-tail issues. It introduces a conditional vector to handle mixed-type features and multi-modal continuous variables, extending support beyond discrete and continuous features. Further, it supports mixed-type features, which are features that might have highly-recurring discrete values but also continuous values across the data points (e.g., a feature *Credit Card Transaction USD* might often have value 0, but it can also exceed 500). The benefit of providing support for mixed-type features is reflected in the fidelity performance, with CTAB-GAN outperforming CTGAN. CTAB-GAN+ (Zhao et al., 2022) is an extension of CTAB-GAN, addressing privacy concerns by training a discriminator using differential private-SGD (Abadi et al., 2016). Importantly, CTAB-GAN+ demonstrates high fidelity performance.

**VAE-based models.** TVAE (Xu et al., 2019), proposed along with CTGAN, was designed to check whether CTGAN's generator's lack of access to the real data during training makes it weaker compared to models that do use these data for training their generator, such as models based on Variational Autoencoders (VAEs) (Kingma and Welling, 2014). Indeed, TVAE's architecture showed an improvement over CTGAN in terms of fidelity. However, like CTAB-GAN+, the authors also considered the impact of their proposed models w.r.t. privacy and noted that TVAE's access to the training data might make it unsuitable in downstream tasks where sensitive data are present.

**Normalising flows-based models.** Neural Spline Flows: Rather than using the common transformations found in normalising flows, Durkan et al. (2019) proposed using a fully-differentiable module based on monotonic rational-quadratic piecewise functions. This adaptation, combined with the inherent properties of normalising flows, which model the data as the output of an invertible differentiable transformation of a noisy sample drawn from a known distribution, can help synthesise high-fidelity tabular data. The authors note that such a result is conditional on having enough training data compared to the datasets dimensionality.

**Diffusion-based models.** FinDiff: Motivated by the need to preserve the statistical properties of real data of mixed types in real-world financial contexts, Sattarov et al. (2023) propose FinDiff, which is capable of generating mixed-type tabular financial data using a diffusion model to capture high-dimensional dependencies. Unlike another popular diffusion model, TabDDPM (Kotelnikov et al., 2023), which uses one-hot encoding for categorical features, FinDiff employs an embedding encoding for better handling of mixed data types. This leads to higher fidelity performance compared to the baselines, indicating that FinDiff effectively captures both feature-wise and joint distributions of the data.

**Diffusion and AE-based models.** AutoDiff: Proposed in (Suh et al., 2023), AutoDiff is a diffusion-based model designed to preserve statistical fidelity by modelling the joint distribution via an autoencoder, rather than modelling the distributions of individual features separately, as seen in GAN- and diffusion-based models like CTGAN and TabDDPM. More precisely, AutoDiff uses the autoencoder to learn the joint distribution of the features (which can be of mixed types) in a continuous space, and then passes these continuous representations to the diffusion model. This model, in turn, generates latent representations that are then decoded back into representations in the original feature space. Moreover, like CTAB-GAN, it introduces an approach to handle mixed-type features by adding a new feature to the autoencoder that encodes the frequency of recurring values in a mixed-type feature.

**Diffusion and VAE-based models.** TabSyn: Introduced by Zhang et al. (2024), it trains a score-based diffusion model, like STaSy, but in a joint VAE-learned space of numerical and categorical features, aiming to preserve the correlations between the features. Notably, TabSyn significantly reduces the runtime of diffusion-based models while outperforming baselines w.r.t. fidelity and machine learning efficacy.

#### 2.2.2.4 Privacy

Synthetic data are particularly valuable in privacy-sensitive domains like finance and healthcare. Hence, there has been growing focus on generating data without revealing sensitive information that could identify entities from the original dataset.

Privacy is typically evaluated by assessing the risk of attacks succeeding in targeting sensitive information. The most common attacks are: (i) *membership attacks*, which use machine learning classifiers or black-box attacks (e.g., see (Chen et al., 2020)) to determine whether an entity was part of the set used to train a model, (ii) *attribute disclosure attacks*, which try to gain access to sensitive attributes of an

entity from the real dataset, and (iii) *re-identification attacks*, which try to map a synthetic data point back to the original dataset.

To measure the risk of the attacks being successful, for (i), AUCROC (Eq. G.18) is reported for the trained attack models (typically one for each class of a feature of interest), for (ii), precision (Eq. G.19) and sensitivity (i.e., recall) (Eq. G.20) are measured while varying the number of attributes known to the attacker, and, for (iii), common metrics include: the distance to the closest record (DCR) (Zhao et al., 2021) (computed as the minimum L2 distance from a synthetic data point to each real data point, as in Eq. G.21), the nearest neighbour distance ratio (computed as the ratio between the distances to the closest and second-closest real neighbours for a synthetic record, as in Eq. G.22), but also k-anonymisation (Eq. G.23) as seen in (Jia et al., 2024).

Finally, methods using differential privacy (Dwork, 2006) introduce noise during training to reduce retention of original data features. Privacy is then evaluated using different privacy budget values, which determine the amount of added noise and balance machine learning efficacy and privacy (e.g., see (Park et al., 2018)).

**AE and GAN-based models.** medGAN: Proposed by Choi et al. (2017) for synthesising healthcare datasets, medGAN is one of the pioneering models for synthesising tabular data using deep generative models. It generates distributed representations of health records using the generator of a GAN-based model, then decodes these representations using an autoencoder pretrained on the real data, and passes the decoded representations to the discriminator component of the GAN, which is trained to distinguish the synthetic from the real data. In terms of privacy, medGAN showed low attribute disclosure risks, except when an attacker focused on a small number of data points. IT-GAN: Similar to OCT-GAN, but designed to enhance privacy robustness, IT-GAN (Lee et al., 2021) employs a generator based on neural ordinary differential equations (neural ODEs) to balance machine learning efficacy and privacy protection. Instead of directly synthesising samples in the input space, the generator produces hidden representations, as seen also in medGAN. Notably, it ensures that the hidden representation space matches the input dimensionality, enabling the application of neural ODEs.

**GAN-based models.** TableGAN: Another model that marked an important milestone for deep generative modelling of tabular data is the framework developed by Park et al. (2018), which, unlike medGAN, can model continuous features. While it is based on a standard GAN model, consisting of a generator and a discriminator, TableGAN has an additional component: a classifier that predicts the value of the

target feature for synthetic samples using semantic integrity constraints learned from the real data. As exemplified in the original paper, a sample with values of 60 and 1 for features *Cholesterol* and *Diabetes*, respectively, does not align with the background knowledge, as the cholesterol level is too low for the target feature to indicate a diabetes diagnosis. Focusing on ensuring privacy, the authors argue for the use of synthesiser models, which do not learn bijective relationships between real and fake data, making them less vulnerable to re-identification attacks, unlike prior perturbation or anonymisation methods. Indeed, TableGAN shows that it can synthesise tabular data with a low risk of re-identification, membership and attribute disclosure attacks to succeed. PATE-GAN: Another model that is based on a GAN architecture, but alters it for preserving privacy, is PATE-GAN (Jordon et al., 2019). Rather than using the standard GAN discriminator, it relies on a modified Private Aggregation of Teacher Ensemble (PATE) (Papernot et al., 2017) mechanism to ensure that the discriminator is differentially private. More specifically, a number of teacher-discriminators are created and trained to improve their loss w.r.t. the generator, which adjusts its loss according to the single student-discriminator in the mechanism. In turn, the student-discriminator adjusts its loss according to the teacher-discriminators and allows for backpropagation to the generator, closing the loop. Critically to the privacy requirement, specialising each of the teacher-discriminators on small partitions of the training set ensures that the effect of individual samples is small and, thus, higher differential privacy. CuTS: Vero et al. (2024) proposed CuTS as a customisable framework, which can adapt to user-defined specifications, such as background knowledge constraints, but also to differentially private training. In the private setting, CuTS builds on the DP iterative framework from (McKenna et al., 2022), but uses its own GAN-based generator, which is equipped with a mechanism that matches the generator's output distribution to the real data distribution by comparing marginals. Additionally, CuTS can incorporate background knowledge in the form of propositional logical constraints that each datapoint must satisfy via a new loss term. More specifically, since these constraints are hard, i.e. non-differentiable, this method uses a binary mask that indicates the synthetic samples which do not satisfy a given constraint all while working within the one-hot encoding representation of the feature values, which is differentiable. Finally, rejection sampling is used to ensure that the model's outputs satisfy the constraints. Although the framework works only with discrete features, it takes a step towards synthesisers for tabular data that account for multiple requirements, including alignment and privacy.

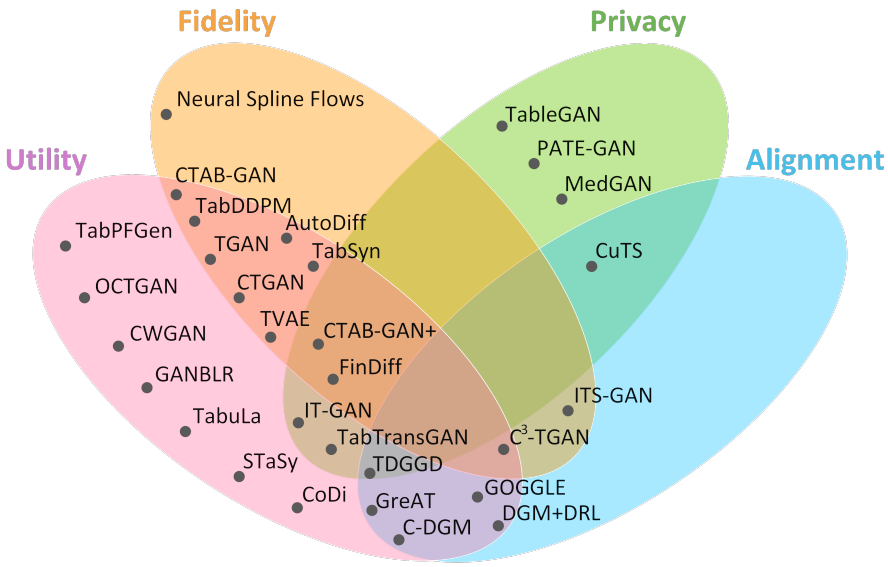



Figure 2.2: Visualisation of the reviewed models based on the requirements they address.

### 2.2.3 Other Metrics and Models Adapted to Tabular Data Synthesis

**Other metrics.** Synthetic data can be evaluated in terms of other considerations. Sampling diversity is another such evaluation measure and it is typically determined by one of the following metrics: (i) coverage, i.e. the proportion of real datapoints that have at least one synthetic datapoint contained within their manifold, as used by STaSy, (ii) the proportion of unique synthetic samples weighted by their efficacy scores, as used by TabGFN to measure the sampling diversity of discrete data, and (iii) authenticity, as used by GOGGLE. A different measure which can be used to evaluate the synthetic data is detection. It measures how easy it is for a machine learning model to predict whether the data are real or synthetic, with results typically reported using AUCROC.

Moreover, tabular data can also be synthesised using deep generative models adapted to this application domain. While in Sections 2.2.2.1-2.2.2.4 I compared deep generative models originally proposed for tabular data synthesis, here I discuss seminal models first developed in other fields and later adapted for tabular data.

**Models adapted to tabular data synthesis.** WGAN (Arjovsky et al., 2017): Originally designed for computer vision tasks, it was introduced in a paper that explores ways to measure the distance between the real and model distributions, arguing that weaker loss functions induce weaker model topologies, thus creating weak mod-

els for the real distribution. The authors compare the Wasserstein distance with traditional probability distance measures, including Kullback-Leibler (KL) divergence, Jensen-Shannon divergence, and total variation distance, showing that the Wasserstein distance has properties that would be better suited for learning distributions in low-dimensional manifolds. They thus propose GAN models that use the Wasserstein distance as loss function. WGAN-GP: A popular extension of WGAN is the model proposed by Gulrajani et al. (2017), which added a new term in the loss that penalised the gradient norm of the discriminator w.r.t. the discriminator's input, providing an alternative to the weight clipping that was leading to the generator retaining the real data points and adding Gaussian noise to fill the gaps. VEEGAN: Srivastava et al. (2017) jointly train a generator and a novel reconstructor network in order to avoid mode collapse. The reconstructor learns to map the real data distribution to Gaussian random noise, approximating the inverse of the generator and encouraging the generator to map the noise distribution back to the real data distribution.

## 2.3 Summary and Discussion

In this Chapter, I reviewed popular neuro-symbolic methods, grouping them into two main branches, namely (i) the methods that integrate background knowledge constraints into the topology of the neural networks, and (ii) those that integrate constraints into the loss function. I then focused on deep generative modelling methods for synthesising tabular data. Through a detailed survey, I identified four key requirements for the effective deployment of synthetic data, along with the relevant evaluation procedures and deep generative models, at the same time categorising the models by their architecture.

Tabular data generation is a rapidly evolving field, and early approaches have often prioritised individual objectives such as machine learning efficacy or privacy. As illustrated in Figure 2.2, this fragmentation highlights a key challenge: the need for models that can balance multiple, often competing, requirements. Bridging this gap will be essential for advancing the field and enabling more reliable applications of tabular data generation.

In this thesis, I propose methods for synthesising tabular data that focus on the background knowledge alignment and belong to the first neuro-symbolic branch mentioned above. A key contribution of this work, as demonstrated later in this thesis, is that my proposed methods do not introduce trade-offs between alignment and other requirements, such as machine learning efficacy. Since my proposed frameworks are

layer-based, they can be applied to any deep generative model, inheriting the requirements handled by the underlying architecture, as indicated in Table 2.2. As the field matures, new requirements will likely emerge, and this kind of hybridisation will be crucial for developing more versatile models.

Finally, in this thesis I also present a method that belongs to the second group of neuro-symbolic methods. Applied to yet another complex real-world scenario, namely autonomous driving, my proposed method demonstrates that it is possible to integrate constraints into the neural networks' loss function efficiently, even in such a demanding scenario. Similarly to the tabular data synthesis methods, I show that this method does not introduce a trade-off between satisfying constraints and achieving high predictive performance. In fact, I show that it often improves the predictive performance, particularly in low-data regimes.

# Chapter 3

# Deep Generative Modelling with Linear Constraints

In this Chapter, which is based on (Stoian et al., 2024a), I introduce a framework for deep generative modelling using background knowledge, expressed as linear inequalities, to synthesise realistic tabular data.

## 3.1 Motivation and Overview

As we have seen in Chapter 2, synthetic data generation has seen dedicated efforts from the machine learning community, its importance growing due to its numerous applications in the real world. Indeed, synthetic data have been increasingly used to augment real data to improve the predictive performance of machine learning models (see, e.g., (Han et al., 2005)) and even to generate new datasets to ensure privacy in sensitive settings (see, e.g., (Jordon et al., 2019; Yoon et al., 2020; Lee et al., 2021)).

One of the most powerful tools for synthesising tabular data are the Deep Generative Models (DGMs), whose capability to capture the complex distributions that characterise the real data has progressively improved (see, e.g., (Kim et al., 2023)). However, generating realistic tabular data requires more than merely learning a good distribution approximation. The synthetic samples must also obey constraints that express background knowledge about known characteristics of the features and/or existing relationships among them. For example, assume a healthcare dataset containing features for the *minimum* and *maximum haemoglobin level* recorded for each patient. Clearly, any synthesised data must satisfy the constraint that the maximum value is greater than or equal to its corresponding minimum value. This relationship can be simply represented by a linear inequality over two variables and yet existing methods, while excellent at capturing complex distributions, are still not able to learn even

from this type of background knowledge, and, thus, do not provide any guarantee of constraint satisfaction. In addition, no prior work incorporates background knowledge in DGMs for tabular data except for GOGGLE (Liu et al., 2022), which, however, is only able to inject very simple background knowledge about existing correlations among features.

To address this limitation, in this Chapter, I show how to integrate any background knowledge that can be expressed as a set of linear inequalities into standard DGMs for tabular data. To this end, in Section 3.2, I first introduce the problem of constrained generative modelling with background knowledge expressed as linear inequalities and then, in Section 3.3, I introduce a novel approach which takes as input user-defined linear inequality constraints and a DGM, and then automatically parses the constraints and generates a differentiable Linearly-constrained Layer (LL) that can be seamlessly integrated with the DGM. This, in turn, results in a novel model, namely the Deep Generative Model equipped with the Linearly-constrained Layer (DGM+LL), whose sample space is guaranteed to be compliant with the constraints. Importantly, this framework distinguishes itself from the prior works which rely on rejection sampling, being the first work to directly incorporate background knowledge into DGMs and guarantee its satisfaction.

To evaluate my approach, I conduct an extensive experimental analysis involving (i) five different DGM architectures: Wasserstein GAN (WGAN) (Arjovsky et al., 2017), CTGAN (Xu et al., 2019), TableGAN (Park et al., 2018), TVAE (Xu et al., 2019), and GOGGLE (Liu et al., 2022), and (ii) six different datasets. Crucially, I select datasets enriched with background knowledge, annotated with up to 31 constraints, each encompassing up to 17 distinct features. To this end, I first show that these models frequently violate the constraints: WGAN generates 100% non-compliant samples for one dataset, while four out of five models produce over 95% non-compliant samples for another dataset (see Table 3.3).

Then, I consider two standard measures used to evaluate the quality of the synthetic tabular data: (i) detection, which determines whether the synthetic data are distinguishable from the real data, and (ii) machine learning efficacy, which determines to what extent the synthetic data can replace the real data to train machine learning models on downstream tasks. I quantitatively demonstrate that the addition of the constraints in the architecture of the network improves the both the detection and the efficacy. Furthermore, I show that LL can also be applied when the user lacks access to the DGM, or the ability to retrain it with the background knowledge, by

simply incorporating it at inference time as a guardrail. Finally, I show that LL does not hinder the sample generation time of the models.

## 3.2 Problem Statement

Recall from Section 2.2.1 that for the problem of standard generative modelling, I assumed to have $p_X$, an unknown distribution over $X \in \mathbb{R}^D$, and $\mathcal{D}$, a training dataset consisting of $N$ independent and identically distributed (i.i.d.) samples drawn from $p_X$. The goal in this case is to learn from $\mathcal{D}$ the parameters $\theta$ of a generative model such that the model distribution $p_\theta$ approximates $p_X$.

In this Chapter, I focus on the problem of *constrained generative modelling*, where I assume to have access to a set of constraints expressing some background knowledge about the sample space of $p_X$. In particular, here I will assume the constraints to be linear inequalities capturing the background knowledge. Based on these constraints, it is possible to decide whether a sample is admissible (i.e., it satisfies the constraints) or not. The goal in this case is to learn the parameters $\theta$ of a generative model such that not only the model distribution $p_\theta$ approximates $p_X$, but also that the sample space of $p_\theta$ is compliant with the constraints.

Linear constraints represent an important class of constraints, a fact underlined by the existence of entire fields, such as linear programming, dedicated to their study. They possess many favourable properties, including that (i) they are logically complete, as inconsistencies can be determined by computing linear combinations of the given constraints (Farkas, 1902), (ii) it is possible to compile them into a backtrack-free representation that allows for computing satisfying samples in linear time proportional to the size of the compiled representation (Dechter, 1999), and (iii) they always define a convex space.

In the rest of the Chapter, I will assume that $\mathcal{X} = \{x_k \mid k = 1, \ldots, D\}$ is a set of variables $x_k$, each variable uniquely corresponding to a feature of the given dataset $\mathcal{D}$. Note that from this point onwards, I will use the variables and the features interchangeably. Further, I will assume $\Pi$ is a finite set of constraints, where each constraint is a linear inequality over $\mathcal{X}$ having the following form:

$$\sum_k w_k x_k + b \geq 0, \tag{3.1}$$

with $w_k \in \mathbb{R}$, $b \in \mathbb{R}$, and $x_k$ representing a continuous feature. Given a constraint of the form (3.1), I will say that a variable $x_k$ *appears positively* (resp., *negatively*) in it if $w_k > 0$ (resp., $w_k < 0$). See Example 3.2.1 for a simple set of constraints.

**Example 3.2.1.** *The set of constraints* $\Pi = \{x_1 - x_2 \geq 0, x_2 - 4.6 \geq 0\}$ *requires that for every generated sample* $\tilde{\mathbf{x}}$*, (i) the first feature (associated with* $x_1$*) must be greater than or equal to the second feature (associated with* $x_2$*), and (ii) the second feature must always be at least equal to* 4.6*. In this example, variable* $x_1$ *appears positively in the first constraint, while variable* $x_2$ *appears in both constraints: negatively in the first, and positively in the second.*

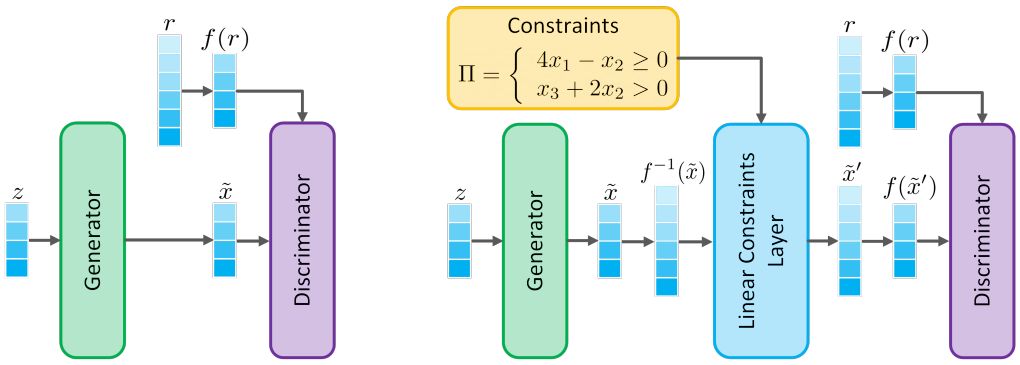
Note that any set of linear inequalities $\sum_k w_k x_k + b \leq 0$, can be rewritten as $\sum_k w'_k x_k + b' \geq 0$, where $w'_k = -w_k$ and $b' = -b$. Additionally, any equality $\sum_k w_k x_k + b = 0$ can be easily expressed as the conjunction of two inequalities, captured by the set of constraints $\{\sum_k w_k x_k + b \geq 0, \sum_k w_k x_k + b \leq 0\}$. Furthermore, strict constraints of the form $\sum_k w_k x_k + b > 0$ can easily be expressed as $\sum_k w_k x_k + b' \geq 0$, where $b' = b - \epsilon$ and $\epsilon$ is the smallest representable positive real number.

I will denote by $\tilde{\mathbf{x}}$ a *sample* generated by a DGM. Any $\tilde{\mathbf{x}}$ is an assignment to the variables in $\mathcal{X}$ such that $\tilde{x}_k$ is the value of the sample corresponding to variable $x_k$. Further, a sample $\tilde{\mathbf{x}}$ *satisfies*: (i) a constraint of the form (3.1), if substituting $\tilde{x}_k$ for $x_k$ yields a valid inequality, and (ii) a set $\Pi$ of constraints of the form (3.1), if it satisfies all the constraints in $\Pi$. On the other hand, if $\tilde{\mathbf{x}}$ does not satisfy a constraint (resp., $\Pi$) then $\tilde{\mathbf{x}}$ *violates* the constraint (resp., $\Pi$). A set $\Pi$ of constraints is *satisfiable* if there exists a sample satisfying $\Pi$. A DGM $m$ is *compliant* with $\Pi$ if all of its generated samples satisfy $\Pi$. Moreover, I will refer to the space determined by the samples that satisfy $\Pi$ as the *valid output space*. Thus, it is possible to rephrase the above as follows: a DGM $m$ is *compliant* with $\Pi$ if its generated samples always fall in the valid output space.

Given the above, I can now formulate the problem I am addressing as follows. Suppose there is a DGM $m$ generating samples that do not necessarily satisfy $\Pi$. The goal is to create a DGM $m'$ such that: (i) $m'$ is compliant with $\Pi$, and (ii) for each sample $\tilde{\mathbf{x}}$ generated by $m$, $m'$ generates a sample $\tilde{\mathbf{x}}'$ such that $\tilde{\mathbf{x}}'$ satisfies $\Pi$ and $\tilde{\mathbf{x}}'$ minimally differs from $\tilde{\mathbf{x}}$ (while taking into account the user preferences on which features should be changed first).

## 3.3 Constrained Deep Generative Models

To achieve the goal above, I propose a framework that builds a differentiable layer that (i) can be seamlessly integrated with DGMs of multiple architectures, (ii) guarantees the satisfaction of the constraints, and (iii) guarantees an output that minimally




(a) A standard GAN-based model. (b) A GAN-based model with LL.

Figure 3.1: Illustrating how LL can be integrated into a GAN-based model.

changes the initial DGM predictions. As the constraints are assumed to be linear inequalities, I will call the layer the Linearly-constrained Layer (LL).

Given a standard DGM, LL can be added immediately after the layer generating the samples. Furthermore, LL allows the gradients to backpropagate through it and, therefore, (i) LL can be used during both training and inference, and (ii) the network can learn to exploit the background knowledge that LL encodes. I will refer to a DGM equipped with LL simply as a DGM+LL model.

To see how LL can be integrated into a DGM, consider Figure 3.1. On the left panel, in Figure 3.1a, we can see an overview of a standard DGM with a GAN-based architecture, where a noise vector $\mathbf{z}$ is passed to the generator, which in turn outputs a sample $\tilde{\mathbf{x}}$. Often, the real datapoint $\mathbf{r} \in \mathcal{D}$ is first processed by a pre-defined bijective mapping $f$ before being passed to the discriminator along with $\tilde{\mathbf{x}}$. This transformation is needed as the generator's output space may differ from the feature space in which the real data reside.

Given this GAN-based architecture, in Figure 3.1b, I show how LL can be integrated into the DGM. Specifically, the sample $\tilde{\mathbf{x}}$ received from the generator needs to be first transformed into the feature space in which LL works. As $f$ is assumed to be a bijective function, it is invertible and, thus, $f^{-1}$ exists. Hence, the LL simply receives as input $f^{-1}(\tilde{\mathbf{x}})$ and returns the corrected $\tilde{\mathbf{x}}'$, which is now compliant with the constraints $\Pi$. Similarly to the real datapoint $\mathbf{r}$, $\tilde{\mathbf{x}}'$ needs to be processed by $f$ before being passed to the discriminator.

In the following, I focus on the construction of LL, and, for ease of presentation and without loss of generality, I assume that the output space of the generator coincides with the feature space of the real dataset. Thus, I assume that the generator generates a sample $\tilde{\mathbf{x}}$, which is directly passed to LL to obtain LL($\tilde{\mathbf{x}}$), the output of

the DGM+LL. Then, given a sample $\tilde{\mathbf{x}}$, the constraint layer changes $\tilde{\mathbf{x}}$ feature by feature w.r.t. an ordering of the features. More precisely, LL minimally changes the value $\tilde{x}_i$ of the $i$-th feature in the sample whenever $\tilde{x}_i$ is not compatible with the values of the already analysed features in the feature ordering and the given constraints. To this end, let $\lambda : \mathcal{X} \mapsto [1, D]$ be a *variable ordering function* injectively mapping each variable in $\mathcal{X}$ to an integer in $[1, D]$. I will assume that this ordering is user-defined and that it determines the order in which LL computes the values of the features. To ease the notation, and without loss of generality, from now on I assume that $\lambda(x_i) = i$ and, thus, I denote the $i$-th variable within the ordering $i$ simply by $x_i$.

Hence, given $\lambda$ and $\Pi$, LL automatically creates a set of constraints $\Pi_i$ for each variable $x_i$ in the reversed $\lambda$ ordering, as follows:

$$\Pi_i = \begin{cases} \Pi, & \text{if } i = D, \\ \Pi_j \setminus (\Pi_j^- \cup \Pi_j^+) \cup \{red_j(\phi^1, \phi^2) \mid \phi^1 \in \Pi_j^-, \phi^2 \in \Pi_j^+\}, & \text{otherwise,} \end{cases} \tag{3.2}$$

where (i) $j = i + 1$, (ii) $\Pi_j^-$ (resp., $\Pi_j^+$) is the subset of $\Pi_j$ in which $x_j$ appears negatively (resp., positively), and (iii) $red_j(\phi^1, \phi^2)$ is the *reduction of $\phi^1$ and $\phi^2$ over* $x_j$ defined, assuming $\phi^1 = \sum_k w_k^1 x_k + b^1 \geq 0$ and $\phi^2 = \sum_k w_k^2 x_k + b^2 \geq 0$, as

$$\sum_{k \neq j} (w_k^1 |w_j^2| + w_k^2 |w_j^1|) x_k + b^1 |w_j^2| + b^2 |w_j^1| \geq 0.$$

**Example 3.3.1.** *Continuing Example 3.2.1, where* $\Pi = \{x_1 - x_2 \geq 0, x_2 - 4.6 \geq 0\}$, *we have that* $\Pi_2 = \Pi$ *and* $\Pi_1 = \{red_2(x_1 - x_2 \geq 0, x_2 - 4.6 \geq 0)\} = \{x_1 - 4.6 \geq 0\}$.

Consider now a generic DGM model $m$ and a sample $\tilde{\mathbf{x}}$ generated by $m$. The constraints $\Pi$ define the valid output space, and the goal is for the outputs of $m$ to fall within this space. LL determines the bounds of this space and uses them to correct $\tilde{\mathbf{x}}$ whenever it does not belong to this region. To define LL($\tilde{\mathbf{x}}$), I will first introduce the bounds which can be derived from the constraints.

**Definition 3.3.2.** *For any $i$ in $\{1, \ldots, D\}$, a constraint $\phi$ of the form (3.1) s.t. $w_i \neq 0$ defines a bound on $x_i$ given by the expression:*

$$\varepsilon_i^\phi = -\sum_{k \neq i} (w_k / w_i) x_k - b / w_i.$$

*Furthermore, if $w_i$ is positive, $\varepsilon_i^\phi$ is a lower bound on $x_i$, i.e., $x_i - \varepsilon_i^\phi \geq 0$. Otherwise, $\varepsilon_i^\phi$ is an upper bound on $x_i$, i.e., $-x_i + \varepsilon_i^\phi \geq 0$.*

Given Definition 3.3.2, the constraints $\Pi_i$, and the computed values $\mathrm{LL}(\tilde{\mathbf{x}})_j$ associated with the features $x_j$ with $j < i$, I can now determine the upper bounds (resp., lower bounds) on $x_i$, for each $i$ in $\{1, \ldots, D\}$. Clearly, each constraint $\phi$ in $\Pi_i^+$ (resp., $\Pi_i^-$) defines a lower (resp., upper) bound on $x_i$. And, as each bound expressed by $\varepsilon_i^\phi$ may contain variables $x_j$ distinct from $x_i$, all $x_j$'s need to be instantiated to get a value for the bound. Note that, instead of substituting $\tilde{x}_j$ for $x_j$, it is crucial $\mathrm{LL}(\tilde{\mathbf{x}})_j$ is substituted for $x_j$. This step will ensure that the changes made to compute the values of $\mathrm{LL}(\tilde{\mathbf{x}})_j$ corresponding to the variables $x_j$ appearing before $x_i$ in the $\lambda$ ordering are accounted for when computing $\mathrm{LL}(\tilde{\mathbf{x}})_i$.

Importantly, $x_i$ may appear in multiple constraints in $\Pi_i^+$ (resp., $\Pi_i^-$), this means that multiple lower (resp., upper) bounds will be placed on $x_i$. For this reason, from all the lower (resp., upper) bounds only the greatest (resp., least) value need to be considered when constraining the value of each $x_i$, as in Definition 3.3.3 below.

**Definition 3.3.3.** *For any $i$ in $\{1, \ldots, D\}$, given sample $\tilde{\mathbf{x}}$ and the values of $LL(\tilde{\mathbf{x}})_1$,..., $LL(\tilde{\mathbf{x}})_{i-1}$, the values associated with the greatest lower bound ($lb_i$) and the least upper bound ($ub_i$) for feature $i$ are defined to be:* [1]

$$lb_i = \max\Big(\bigcup_{\phi \in \Pi_i^+} \varepsilon_i^\phi(LL(\tilde{\mathbf{x}}))\Big) \qquad ub_i = \min\Big(\bigcup_{\phi \in \Pi_i^-} \varepsilon_i^\phi(LL(\tilde{\mathbf{x}}))\Big), \tag{3.3}$$

*where $\varepsilon_i^\phi(LL(\tilde{\mathbf{x}}))$ corresponds to $\varepsilon_i^\phi$ instantiated with the values of $LL(\tilde{\mathbf{x}})$ such that each $x_j$ is replaced by the corresponding value of $LL(\tilde{\mathbf{x}})_j$, for any $j$ in $\{1, \ldots, i-1\}$.*

Clearly, the fact that $\Pi$ is satisfiable and the definition of $ub_i$ and $lb_i$ in (3.3) which incorporates the values of $\mathrm{LL}(\tilde{\mathbf{x}})_j$ for $j < i$, ensures $lb_i \leq ub_i$ for any $i = 1, \ldots, D$ (as formally stated below in Lemma 3.3.7).

**Definition 3.3.4.** *For any $i = 1, \ldots, D$, $LL(\tilde{\mathbf{x}})_i$ is defined as:*

$$LL(\tilde{\mathbf{x}})_i = \min(\max(\tilde{x}_i, lb_i), ub_i). \tag{3.4}$$

Thus, for every $i = 1, \ldots, D$, I can define $\mathrm{LL}(\tilde{\mathbf{x}})_i$ and ultimately obtain a new sample $\mathrm{LL}(\tilde{\mathbf{x}})$, which is guaranteed to satisfy the constraints in $\Pi$.

**Example 3.3.5.** *Continuing with the example, $\Pi = \{x_1 - x_2 \geq 0, x_2 - 4.6 \geq 0\}$, $\Pi_2 = \Pi$, $\Pi_1 = \{x_1 - 4.6 \geq 0\}$. If $\tilde{x}_1 = 7$ and $\tilde{x}_2 = 3$, then $LL(\tilde{\mathbf{x}})_1 = 7$ and $LL(\tilde{\mathbf{x}})_2 = 4.6$, while if $\tilde{x}_1 = \tilde{x}_2 = 3$, then $LL(\tilde{\mathbf{x}})_1 = LL(\tilde{\mathbf{x}})_2 = 4.6$.*

[1] I assume the function $\min(V)$ over a finite set $V$ of values in $\mathbb{R}$ to be defined as $\min(\emptyset) = +\infty$, and $\min(\{v\} \cup V') = v$ if $v \leq \min(V')$ and $\min(V')$ otherwise. Analogously for the function $\max(V)$.

I will now prove that $\mathrm{LL}(\tilde{\mathbf{x}})$ satisfies the constraints in $\Pi$.

**Theorem 3.3.6.** *Let $m$ be a deep generative model. Let $\Pi$ be a satisfiable and finite set of constraints over the sample space. Let $\lambda$ be a variable ordering and LL be the constraint layer built from $\lambda$ and $\Pi$. Then, the model $m$+LL obtained by incorporating LL into $m$ is compliant with $\Pi$.*

*Proof.* The proof is by induction on the number of variables in $\Pi$, denoted by $n$. Recall that $\{x_1, \ldots, x_D\}$ is the set of variables and that I assumed, w.l.o.g., $\lambda(x_i) = i$.

For the base case $n = 0$, $\Pi$ does not contain variables and, since $\Pi$ is satisfiable, for each constraint $\sum_k w_k x_k + b \geq 0$ in $\Pi$, $w_k = 0$ for all $k$, $b \geq 0$, and any sample satisfies the constraint. Thus, any sample satisfies $\Pi$.

Assume now that $n > 1$ variables appear in $\Pi$ and let $x_k$ be the variable with the highest $\lambda$ value occurring in $\Pi$. This means that $x_D, x_{D-1}, \ldots, x_{k+1}$ do not occur in $\Pi$. So, by the definition of $\Pi_i$ in (3.2), $\Pi_D = \Pi_{D-1} = \ldots = \Pi_k = \Pi$ and exactly $n$ variables appear in $\Pi_k$. Moreover,

$$\Pi_{k-1} = \Pi_k \setminus (\Pi_k^- \cup \Pi_k^+) \cup \{red_k(\phi^1, \phi^2) \mid \phi^1 \in \Pi_k^-, \phi^2 \in \Pi_k^+\}. \tag{3.5}$$

Clearly, by construction, less than $n$ variables appear in $\Pi_{k-1}$. Then, for the inductive hypothesis, assume $\mathrm{LL}(\tilde{\mathbf{x}})$ satisfies $\Pi_{k-1}$, for an arbitrary sample $\tilde{\mathbf{x}}$.

For the induction step, I will prove that if the induction hypothesis holds, then $\mathrm{LL}(\tilde{\mathbf{x}})$ satisfies $\Pi_k$. Since Equation (3.5) implies that $\Pi_k \subseteq \Pi_{k-1} \cup \Pi_k^- \cup \Pi_k^+$, proving that $\mathrm{LL}(\tilde{\mathbf{x}})$ satisfies $\Pi_k$ reduces to proving that $\mathrm{LL}(\tilde{\mathbf{x}})$ satisfies both $\Pi_k^+$ and $\Pi_k^-$.

I will first prove that $\mathrm{LL}(\tilde{\mathbf{x}})$ satisfies each constraint $\phi \in \Pi_k^+$. From Definition (3.3.2), recall that $\phi$ can be expressed as $x_k \geq \varepsilon_k^\phi$. It is then sufficient to prove that the inequality holds if we instantiate both sides with $\mathrm{LL}(\tilde{\mathbf{x}})$, i.e., show that $\mathrm{LL}(\tilde{\mathbf{x}})_k \geq \varepsilon_k^\phi(\mathrm{LL}(\tilde{\mathbf{x}}))$ holds. From Definition (3.3.3), $lb_k \geq \varepsilon_k^\phi(\mathrm{LL}(\tilde{\mathbf{x}}))$. Additionally, by Definition (3.3.4), it is clear that $\mathrm{LL}(\tilde{\mathbf{x}})_k \geq lb_k$. Thus, $\mathrm{LL}(\tilde{\mathbf{x}})_k \geq \varepsilon_k^\phi(\mathrm{LL}(\tilde{\mathbf{x}}))$, proving that $\mathrm{LL}(\tilde{\mathbf{x}})$ satisfies $\phi$, for all $\phi \in \Pi_k^+$. Analogously, $\mathrm{LL}(\tilde{\mathbf{x}})$ satisfies each constraint $\phi \in \Pi_k^-$. □

Before proving the rest of the LL's properties, I introduce below a helpful Lemma.

**Lemma 3.3.7.** *For an arbitrary sample $\tilde{\mathbf{x}}$, let $lb_i$ and $ub_i$ be the greatest lower bound and the least upper bound, respectively, determined by the constraint layer LL for feature $x_i$. Then:*
*(a) If at least one of $lb_i$ and $ub_i$ is not finite, then $lb_i < ub_i$.*
*(b) If both $lb_i$ and $ub_i$ are finite, then $lb_i \leq ub_i$.*

*Proof.* (a) If at least one of $lb_i$ and $ub_i$ is not finite, then at least one of $\Pi_i^+$ and $\Pi_i^-$ is empty. By Definition (3.3.3), we have that (i) if $\Pi_i^+$ is empty, then $lb_i$ is $-\infty$, and (ii) if $\Pi_i^-$ is empty, then $ub_i$ is $\infty$. Thus, $lb_i < ub_i$.

(b) If both $lb_i$ and $ub_i$ are finite, then both $\Pi_i^+$ and $\Pi_i^-$ are non-empty. Then, by Definition (3.3.3), there exist constraints $\phi^+ \in \Pi_i^+$ and $\phi^- \in \Pi_i^-$ such that $lb_i = \varepsilon_i^{\phi^+}(\mathrm{LL}(\tilde{\mathbf{x}}))$ and $ub_i = \varepsilon_i^{\phi^-}(\mathrm{LL}(\tilde{\mathbf{x}}))$, respectively. Substituting $\mathrm{LL}(\tilde{\mathbf{x}})_j$ for each $x_j$ in $\varepsilon_i^{\phi^+}$ and $\varepsilon_i^{\phi^-}$, it is possible to rewrite $\phi^+$ and $\phi^-$ as $lb_i \leq x_i$ and $x_i \leq ub_i$, respectively, from which the Lemma statement immediately follows. □

**Corollary 3.3.8.** *Suppose that the assumptions of Theorem 3.3.6 hold. Then, for any $i$ in $\{1, \ldots, D\}$, the following is true:*

$$LL(\tilde{\mathbf{x}})_i = \min(\max(\tilde{x}_i, lb_i), ub_i) = \max(\min(\tilde{x}_i, ub_i), lb_i). \tag{3.6}$$

*Proof.* If both $lb_i$ and $ub_i$ are finite then, from Lemma 3.3.7, $lb_i \leq ub_i$ for any sample $\tilde{\mathbf{x}}$ produced by $m$ and any $i \in \{1, \ldots, D\}$. Hence, we have three possibilities: $\tilde{x}_i \leq lb_i \leq ub_i$, or $lb_i \leq ub_i \leq \tilde{x}_i$, or $lb_i \leq \tilde{x}_i \leq ub_i$. For each of these, it is easy to check that the value of $\min(\max(\tilde{x}_i, lb_i), ub_i)$ matches the value of $\max(\min(\tilde{x}_i, ub_i), lb_i)$. Analogously, if at least one of $lb_i$ and $ub_i$ is not finite, from Lemma 3.3.7, $lb_i < ub_i$. Thus, the similar three possibilities involving the value of $\tilde{x}_i$ can be easily checked to arrive at the same result.

□

In addition to satisfying the constraints, $\mathrm{LL}(\tilde{\mathbf{x}})$ needs to be optimal in the sense that $\mathrm{LL}(\tilde{\mathbf{x}})$ has to minimally differ from $\tilde{\mathbf{x}}$. This intuition is formalised below, in Definition 3.3.9.

**Definition 3.3.9.** *Given a sample $\tilde{\mathbf{x}}$ of a DGM $m$, a satisfiable and finite set of constraints $\Pi$, and the constrained layer LL, $LL(\tilde{\mathbf{x}})$ is optimal (with respect to $\Pi$) if (i) $LL(\tilde{\mathbf{x}})$ satisfies $\Pi$, and (ii) there does not exist another sample $\tilde{\mathbf{x}}'$, different from $LL(\tilde{\mathbf{x}})$, which satisfies $\Pi$ and such that $|\tilde{x}'_i - \tilde{x}_i| \leq |LL(\tilde{\mathbf{x}})_i - \tilde{x}_i|$, for each $i = 1, \ldots, D$.*

Indeed, if a sample $\tilde{\mathbf{x}}$ satisfies the constraints, we expect $\mathrm{LL}(\tilde{\mathbf{x}})$ to return $\tilde{\mathbf{x}}$ and, thus, be optimal. Notice that more than one value might satisfy the above definition of optimality. Given a sample $\tilde{\mathbf{x}}$, the proposed definition has the advantageous properties that (i) if $\tilde{\mathbf{x}}$ satisfies the constraints then $\mathrm{LL}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}$ is the unique optimal solution, and (ii) if $\tilde{\mathbf{x}}$ does not satisfy the constraints then it is not possible to get "closer" to $\tilde{\mathbf{x}}$ along all the dimensions while satisfying the constraints. This rather relaxed

definition of optimality allows for considering different methods to compute $\text{LL}(\tilde{\mathbf{x}})$, each corresponding to a preference about the set of features which should minimally change. In this case, the features that need to minimally change are those whose distribution is better approximated by the unconstrained DGM. For this reason, in Chapter 4, I discuss how to define different policies to compute $\text{LL}(\tilde{\mathbf{x}})$, which take into account the ability of the DGM to learn the features' distribution and different relations (such as statistical and causal relations) between the features.

**Theorem 3.3.10.** *Let $m$ be a deep generative model and $\tilde{\mathbf{x}}$ a sample produced by $m$. Let $\Pi$ be a satisfiable and finite set of constraints over the sample space. Let $\lambda$ be a variable ordering and LL be the constraint layer built from $\lambda$ and $\Pi$. If $\tilde{\mathbf{x}}$ satisfies $\Pi$, then $LL(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}$ and $LL(\tilde{\mathbf{x}})$ is optimal.*

*Proof.* I will first prove by contradiction that if $\tilde{\mathbf{x}}$ satisfies $\Pi$, then $\text{LL}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}$.

Assume $\text{LL}(\tilde{\mathbf{x}}) \neq \tilde{\mathbf{x}}$ and let $i$ be the lowest index w.r.t. $\lambda$ such that $\text{LL}(\tilde{\mathbf{x}})_i \neq \tilde{x}_i$. Then, $\text{LL}(\tilde{\mathbf{x}})_i = \min(\max(\tilde{x}_i, lb_i), ub_i) \neq \tilde{x}_i$. There are two cases in which this could happen: either (i) $\tilde{x}_i < lb_i$, or (ii) $\tilde{x}_i > ub_i$.

Consider case (i). By the assumption on $i$ and given $\lambda(x_i) = i$, $\tilde{x}_j = \text{LL}(\tilde{\mathbf{x}})_j$ holds for all $j < i$. Hence, for any $\phi \in \Pi_i^+$, $\varepsilon_i^\phi(\text{LL}(\tilde{\mathbf{x}})) = \varepsilon_i^\phi(\tilde{\mathbf{x}})$, which implies that

$$lb_i = \max(\bigcup_{\phi \in \Pi_i^+} \varepsilon_i^\phi(\text{LL}(\tilde{\mathbf{x}}))) = \max(\bigcup_{\phi \in \Pi_i^+} \varepsilon_i^\phi(\tilde{\mathbf{x}})). \tag{3.7}$$

If $\tilde{\mathbf{x}}$ satisfies $\Pi$, then it must satisfy $\Pi_i^+$, since $\Pi_i^+$ is derived from $\Pi$. Thus, $\tilde{x}_i \geq \varepsilon_i^\phi(\tilde{\mathbf{x}})$, for any $\phi \in \Pi_i^+$. Combining this with Equation (3.7), we get $\tilde{x}_i \geq lb_i$, which contradicts the assumption of case (i) under which $\tilde{x}_i < lb_i$. We thus reach a contradiction. The proof for case (ii) is obtained in a similar way. Since in both cases a contradiction is reached, $\text{LL}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}$ holds.

Now I will prove that if $\tilde{\mathbf{x}}$ satisfies $\Pi$, then $\text{LL}(\tilde{\mathbf{x}})$ is optimal. If $\tilde{\mathbf{x}}$ satisfies $\Pi$ then, from Theorem 3.3.10, $\text{LL}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}$. So there is no other sample $\tilde{\mathbf{x}}' \neq \text{LL}(\tilde{\mathbf{x}})$ for which $|\tilde{x}'_i - \tilde{x}_i| \leq |\text{LL}(\tilde{\mathbf{x}})_i - \tilde{x}_i| = 0$, for all $i$. Hence, by Definition 3.3.9 of the optimality of $\text{LL}(\tilde{\mathbf{x}})$, we have that $\text{LL}(\tilde{\mathbf{x}})$ is optimal w.r.t. $\Pi$ if $\tilde{\mathbf{x}}$ satisfies $\Pi$. □

Except in the fortunate cases where sample $\tilde{\mathbf{x}}$ already satisfies the constraints, I will show that $\text{LL}(\tilde{\mathbf{x}})$ is guaranteed to be optimal.

**Lemma 3.3.11.** *Let $m$ be a deep generative model. Let $\Pi$ be a satisfiable and finite set of constraints over the sample space. Let $\lambda$ be a variable ordering and LL be the constraint layer built from $\lambda$ and $\Pi$. Let $\tilde{\mathbf{x}}$ be a sample produced by $m$. Then $LL(\tilde{\mathbf{x}})$ is optimal with respect to $\Pi$.*

*Proof.* Recall that, w.l.o.g., $\lambda(x_i) = i$. To show that $\text{LL}(\tilde{\mathbf{x}})$ is optimal with respect to $\Pi$, it suffices to show that for every $i = 1, \ldots, D$, there is no other sample $\tilde{\mathbf{x}}'$ satisfying $\Pi$ such that (i) $\tilde{x}'_j = \text{LL}(\tilde{\mathbf{x}})_j$, for all $1 \leq j < i$, and (ii) $|\tilde{x}'_i - \tilde{x}_i| < |\text{LL}(\tilde{\mathbf{x}})_i - \tilde{x}_i|$. The proof is by induction on $i$.

For the base case $i = 1$, the only variable that can appear in $\Pi_1$ is $x_1$. Hence, there are four subcases:

1. $\Pi_1$ is either empty or contains a constraint of the form $b \geq 0$, which is always satisfied since $\Pi$ is satisfiable. Thus, $\text{LL}(\tilde{\mathbf{x}})_1 = \tilde{x}_1$.

2. $\Pi_1$ is of the form $\{x_1 + b \geq 0\}$ and, thus, $\text{LL}(\tilde{\mathbf{x}})_1 = \max(\tilde{x}_1, -b)$.

3. $\Pi_1$ is of the form $\{-x_1 + b \geq 0\}$ and, thus, $\text{LL}(\tilde{\mathbf{x}})_1 = \min(\tilde{x}_1, b)$.

4. $\Pi_1$ is of the form $\{x_1 + a \geq 0, -x_1 + b \geq 0\}$. Since $\Pi$ is satisfiable, $a + b \geq 0$. Then,
$$\text{LL}(\tilde{\mathbf{x}})_1 = \begin{cases} -a, & \text{if } \tilde{x}_1 < -a, \\ \tilde{x}_1, & \text{if } -a \leq \tilde{x}_1 \leq b, \\ b, & \text{otherwise.} \end{cases}$$

Clearly, the statement trivially holds in each of these four subcases.

For the induction hypothesis, assume that the statement holds when $i = k$, for some $k > 0$. Now, for the induction step where $i = k+1 > 1$, assume by contradiction that there exists another sample $\tilde{\mathbf{x}}'$ satisfying $\Pi$ such that $\tilde{x}'_j = \text{LL}(\tilde{\mathbf{x}})_j$, for all $1 \leq j < i$, and $|\tilde{x}'_i - \tilde{x}_i| < |\text{LL}(\tilde{\mathbf{x}})_i - \tilde{x}_i|$. From the construction of $\Pi_i$, we know that only $x_1, \ldots, x_i$ appear in $\Pi_i$. So if $\text{LL}(\tilde{\mathbf{x}})_j$ is substituted for each variable $x_j$ in $\Pi_i$, where $j < i$, then the only variable appearing in the resulting set of constraints will be $x_i$. However, this simply represents the base case, for which the statement to be proved holds. Therefore, a contradiction is reached and, thus, the statement is proved. □

## 3.4 Experimental Analysis

In this Section, I quantitatively evaluate different aspects of the approach I proposed above, in Section 3.3, to support the following claims:

1. **Background knowledge alignment:** *How often do constraint violations occur?* In Section 3.4.2, I show that standard DGMs often violate the constraints expressing background knowledge. On the other hand, the DGM+LL models guarantee the constraints' satisfaction.

2. **Synthetic data quality:** *Does background knowledge improve the synthetic data quality?* In Section 3.4.3, I empirically demonstrate that DGM+LL models outperform the standard DGMs in terms of two metrics: machine learning efficacy and detection.
3. **Post-processing ability:** *Can LL act as a guardrail at inference time?* In Section 3.4.4, I show that LL can be used as a guardrail while slightly improving the overall performance of the DGMs.
4. **Impact on generation time and training dynamics:** *Does background knowledge injection affect generation time?* In Section 3.4.5, I compare the sample generation time of the standard DGMs and the DGM+LL models and show that LL does not hinder generation at inference time. Lastly, I investigate the effect of LL on the training dynamics.

First, in Section 3.4.1, I provide details for (i) the models, (ii) the datasets, (iii) the metrics used for evaluating the performance of the models, as well as (iv) the hyperparameter search used for setting up the experiments in the following Sections.

The full code is provided on GitHub.[2]

### 3.4.1 Experimental Setup

**Models.** I experimented with five DGM models, namely WGAN (Arjovsky et al., 2017), TableGAN (Park et al., 2018), CTGAN (Xu et al., 2019), TVAE (Xu et al., 2019), and GOGGLE (Liu et al., 2022). Each model was augmented by LL, resulting in WGAN+LL, TableGAN+LL, CTGAN+LL, TVAE+LL, and GOGGLE+LL models. Below I include a short description for each base model:

- **WGAN** (Arjovsky et al., 2017) is a GAN model trained with Wasserstein loss in a typical generator-discriminator GAN-based architecture. The variant used here relies on a MinMax transformer for the continuous features and on one-hot encoding for categorical ones. Indeed, the original version of WGAN was not specifically designed for tabular data and therefore needed to be adapted for this use case.

- **TableGAN** (Park et al., 2018) is among the first GAN-based approaches proposed for tabular data generation. In addition to the typical generator and discriminator architecture for GANs, the authors proposed adding a classifier trained to learn the relationship between the labels and the other features. The

[2]The code is available at https://github.com/mihaela-stoian/ConstrainedDGM.

classifier ensures a higher number of semantically correct produced records. TableGAN uses a MinMax transformer for the features.

- **CTGAN** (Xu et al., 2019) uses a conditional generator and training-by-sampling strategy in a generator-discriminator GAN architecture to model tabular data. The conditional generator generates synthetic rows conditioned on one of the discrete columns. The training-by-sampling ensures that the data are sampled according to the log-frequency of each category. Both help to better model the imbalanced categorical columns. CTGAN transforms discrete features using one-hot encoding and a mode-based normalisation for continuous features. A variational Gaussian mixture model (Camino et al., 2018) is used to estimate the number of modes and fit a Gaussian mixture. For each continuous value, a mode is sampled based on probability densities, and its mean and standard deviation are used to normalise the value.

- **TVAE** (Xu et al., 2019) was proposed as a variation of the standard Variational AutoEncoder to handle tabular data. It uses the same transformations of data as CTGAN and trains the encoder-decoder architecture using evidence lower-bound (ELBO) loss.

- **GOGGLE** (Liu et al., 2022) is a graph-based approach to learning the relational structure of the data as well as functional relationships (dependencies between features). The relational structure of the data is learned by building a graph where nodes are variables and edges indicate dependencies between them. The functional dependencies are learned through a message passing neural network (MPNN). The generative model generates each variable considering its surrounding neighbourhood.

**Datasets.** To evaluate the models, I selected real-world datasets for which a clear description of the feature relationships was either available or derived from the domain expertise. The selection was limited to datasets with at least three known constraints. As a result, I found six such datasets, four for binary classification tasks, one for multi-class classification, and one for regression.

The selected datasets are listed below:

- URL[3] (Hannousse and Yahiouche, 2021) is used to perform webpage phishing detection with features describing statistical properties of the URL itself as well as the content of the page.

[3]Link to dataset: https://data.mendeley.com/datasets/c2gw7fy2j4/2

Table 3.1: Datasets statistics. For each dataset, the following are provided: (i) the number of samples in the training partition, (ii) the number of samples in the validation partition, (iii) the number of samples in the testing partition, (iv) the number of features, (v) the number of categorical features, (vi) the number of continuous features, and (vii) the type of downstream task (along with the number of output classes, if the task is multi-class classification).

| Dataset | # Train | # Val. | # Test | # Feats. | # Cat. | # Cont. | Task (# classes) |
|---|---|---|---|---|---|---|---|
| URL | 7K | 2K | 2K | 64 | 20 | 44 | Binary classification |
| WiDS | 22K | 6K | 7K | 109 | 9 | 100 | Binary classification |
| LCLD | 494K | 199K | 431K | 29 | 8 | 21 | Binary classification |
| Heloc | 8K | 2K | 0.2K | 24 | 8 | 16 | Binary classification |
| FSP | 2K | 0.2K | 0.2K | 28 | 0 | 28 | Multi-class class. (7) |
| News | 31K | 7K | 1K | 60 | 14 | 46 | Regression |

- WiDS[4] is used to predict if a patient is diagnosed with a particular type of diabetes named Diabetes Mellitus, using data from the first 24 hours of intensive care.

- LCLD[5] is used to predict whether the debt lent is unlikely to be collected. In particular, the LCLD version I used is the feature-engineered dataset from Simonetto et al. (2022), inspired from the LendingClub loan data. The dataset captures features related to the loan as well as client history.

- FICO's Home Equity Line of Credit dataset (Heloc[6]) from the FICO xML Challenge is used to predict whether customers will repay their credit lines within 2 years. Similarly to LCLD, the dataset has features related to the credit line and the client's history.

- FSP[7](Buscema, 1998) is used to predict 7 types of surface defects in stainless steel plates. The features approximately capture the geometric shape of the defect and its outline.

- News[8] (Fernandes et al., 2015) is used to predict the number of times a news article will be shared on social networks. The features capture properties of the text, as well as the publishing time.

[4]Link to dataset: https://www.kaggle.com/competitions/widsdatathon2021
[5]Link to dataset: https://figshare.com/s/84ae808ce6999fafd192
[6]Link to the dataset: https://huggingface.co/datasets/mstz/heloc
[7]Link to dataset: https://www.kaggle.com/datasets/uciml/faulty-steel-plates
[8]Link to dataset: https://archive.ics.uci.edu/dataset/332/online+news+popularity

Table 3.2: Constraints statistics. For each dataset, the following are provided: (i) the number of constraints, (ii) the number of features appearing in the constraints set out of the total number of features available in the dataset, (iii) the average number of features per constraint, (iv) the average number of positive features per constraint, and (v) the average number of negative features per constraint.

| Dataset | # Constr. | $F$ / $D$ | Avg. $F_\phi$ | Avg. $F_\phi^+$ | Avg. $F_\phi^-$ |
|---|---|---|---|---|---|
| URL | 8 | 24 / 64 | 4.25 | 1.00 | 3.25 |
| WiDS | 31 | 62 / 109 | 2.00 | 1.00 | 1.00 |
| LCLD | 4 | 5 / 29 | 1.50 | 0.75 | 0.75 |
| Heloc | 7 | 10 / 24 | 2.00 | 1.00 | 1.00 |
| FSP | 4 | 7 / 28 | 2.00 | 1.00 | 1.00 |
| News | 5 | 9 / 60 | 2.00 | 1.00 | 1.00 |

For URL, WiDS, and LCLD, I used the training-validation-testing splits provided by Simonetto et al. (2022). For Heloc I used the training-testing split of 80-20%, and 20% of the training set was later held out for validation. Finally, for FSP and News I split the data into 80-10-10% sets, and for the former (which is a multi-class classification dataset) I preserved the class imbalance.

The selected datasets vary not only in size (2K to 1M rows), but also in feature number (24 to 109) and feature type (continuous, categorical, or mixed). The constraints between features are equally varied in number (from 4 to 31) as well as number of features appearing in each constraint (from 1 to 17). An overview of these datasets' statistics can be found in Table 3.1.

Finally, I will outline the structure of the constraints found in the selected datasets. To this end, let $F$ be the number of features appearing in at least one constraint, and recall that $D$ is the total number of features. Then, given a constraint $\phi$, let $F_\phi^+$ (resp. $F_\phi^-$) be the number of features appearing positively (resp. negatively) in $\phi$, and let $F_\phi = F_\phi^+ + F_\phi^-$ be the number of features appearing in $\phi$.

As shown in Table 3.2, the constraints characteristics greatly vary from between the datasets. Indeed, we can have from 4 to 31 constraints annotated for each dataset, with as little as 15% of the variables appearing in at least a constraint in News to as much as 56.88% in WiDS. Further, for almost all datasets, the average number of features appearing per constraint is 2, except for (i) URL, which has a constraint where as many as 17 different features appear, and (ii) LCLD, which has two constraints where a single variable appears. For examples of constraints, see Tables H.1 and H.2 of Appendix H.1.

**Sample Quality Evaluation.** I quantitatively evaluate two characteristics of the quality of generated samples: (i) *machine learning efficacy*, i.e., whether the synthetic data can be used as an alternative to the real data to train models for downstream tasks, and (ii) *detection*, i.e., whether a classifier can be trained to tell apart the synthetic samples from the real data.

In order to evaluate the machine learning efficacy of the generated samples, I followed the *Train on Synthetic, Test on Real* (TSTR) framework (Esteban et al., 2017; Jordon et al., 2019). More precisely, I followed closely the TSTR variant presented in Kim et al. (2023). For completion, the protocol is reproduced also here:

1. First, I generated a synthetic dataset and split it into training, validation and test partitions using the same proportions as the real dataset.

2. Then, I performed a hyperparameter search using the synthetic training data partition to train different classifiers/regressors.

   (a) Specifically, for the binary classification datasets (i.e., URL, WiDS, LCLD, and Heloc) I used Decision Tree (Wu et al., 2008), AdaBoost (Schapire, 2013), Multi-layer Perceptron (MLP) (Haykin, 1994), Random Forest (Ho, 1995), XGBoost (Chen and Guestrin, 2016), and Logistic Regression (Cox, 1958) classifiers.

   (b) For multi-class classification datasets, (i.e., FSP) I used Decision Tree, MLP, Random Forest, and XGBoost classifiers.

   (c) For the regression dataset (i.e., News), I used MLP, XGBoost, and Random Forest regressors and linear regression.

   For all the classifiers and regressors above, I considered the same hyperparameter settings as those from Table 26 of (Kim et al., 2023) and picked the best hyperparameter configuration using the real validation set according to the F1-score.

3. Finally, I tested the selected best models on the real test set and averaged the results across all the classifiers/regressors to get performance measurements for the synthetic samples. More precisely, for the classification datasets, I reported the average F1-score, weighted F1-score, and Area Under the ROC Curve. On the other hand, for the regression datasets, I reported the explained variance and mean absolute error.

Table 3.3: Constraint violation rate (CVR) for each model and dataset.

| Model/Dataset | URL | WiDS | LCLD | Heloc | FSP | News |
|---|---|---|---|---|---|---|
| WGAN | 11.1±1.6 | 98.2±0.2 | 100.0±0.0 | 57.0±13.0 | 70.7±8.3 | 45.6±9.6 |
| TableGAN | 4.9±1.4 | 96.4±2.4 | 6.1±0.9 | 45.6±16.3 | 71.6±8.7 | 72.6±5.3 |
| CTGAN | 3.1±2.6 | 99.9±0.0 | 11.8±2.7 | 41.6±12.1 | 74.3±5.2 | 54.3±10.1 |
| TVAE | 3.0±0.7 | 99.9±0.0 | 3.9±0.5 | 55.5±1.4 | 66.4±3.0 | 50.3±3.9 |
| GOGGLE | 5.9±6.6 | 78.2±11.6 | 13.1±2.9 | 47.3±7.0 | 63.7±17.6 | 44.8±7.2 |
| All models+LL | **0.0±0.0** | **0.0±0.0** | **0.0±0.0** | **0.0±0.0** | **0.0 ±0.0** | **0.0±0.0** |

The above procedure was repeated five times for each standard DGM and each DGM+LL model, and the results were averaged separately for every metric.

To evaluate the models in terms of detection, I followed the popular evaluation method for tabular data generation (see, e.g., Liu et al. (2022)). More precisely, I adapted the procedure presented by Kim et al. (2023) and trained multiple classifiers (e.g., Decision Tree, AdaBoost etc.) to distinguish between the real and synthetic datasets. I first created the training, validation, and testing sets by concatenating real and synthetic data, including their targets as usual features, and adding a new target column that specifies whether the data is real or not. By construction, these datasets are binary classification datasets and, thus, are suitable for the hyperparameter search procedure presented in Kim et al. (2023) using the six different binary classifiers mentioned above. I selected the best model using the newly-created validation set, and reported the final detection performance on the newly-created testing data (which combines the real and the synthetic data) w.r.t. the average F1-score, AUROC and weighted F1-score. Again, I reported the metrics over five runs.

**Hyperparameter search.** To ensure that the comparisons between the DGM+LL models and the baseline unconstrained DGMs are meaningful, I conducted an extensive hyperparameter search to reveal the best configurations. In Table A.1 of Appendix A, I provide the best hyperparameter configurations, which I used for the experiments presented in the next Sections. Importantly, I used the same hyperparameters for the DGM+LL models. Furthermore, for the DGM+LL models, I reported the variable ordering hyperparameter that needs to be given to the LL. I give a full description of the orderings and of the impact they have on the performance in Chapter 4.

Table 3.4: Constraints violation coverage (CVC) for all datasets and models.

| Model/Dataset | URL | WIDS | LCLD | HELOC | Faults | News |
|---|---|---|---|---|---|---|
| WGAN | 11.1 ± 1.6 | 100.0 ± 0.0 | 75.0 ± 0.0 | 100.0 ± 0 | 75.0 ± 0.0 | 84.8 ± 8.7 |
| TableGAN | 24.5 ± 3.7 | 100.0 ± 0.0 | 50.0 ± 0.0 | 100.0 ± 0 | 75.0 ± 0.0 | 100.0 ± 0.0 |
| CTGAN | 16.5 ± 3.8 | 100.0 ± 0.0 | 50.0 ± 0.0 | 99.4 ± 1.3 | 75.0 ± 0.0 | 100.0 ± 0.0 |
| TVAE | 12.5 ± 0.0 | 100 ± 0.0 | 50.0 ± 0.0 | 100.0 ± 0.0 | 75.0 ± 0.0 | 100.0 ± 0.0 |
| GOGGLE | 17.5 ± 6.9 | 99.2 ± 1.7 | 50.0 ± 0.0 | 100.0 ± 0.0 | 75.0 ± 0.0 | 100.0 ± 0.0 |
| All models+LL | **0.0 ± 0.0** | **0.0 ± 0.0** | **0.0 ± 0.0** | **0.0 ± 0.0** | **0.0 ± 0.0** | **0.0 ± 0.0** |

Table 3.5: Sample-wise constraints violation coverage (sCVC) for all models and datasets.

| Model/Dataset | URL | WIDS | LCLD | HELOC | Faults | News |
|---|---|---|---|---|---|---|
| WGAN | 1.4 ± 0.2 | 21.2 ± 1.3 | 29.8 ± 0.2 | 10.5 ± 2.5 | 23.3 ± 4.3 | 12.1 ± 2.4 |
| TableGAN | 0.6 ± 0.2 | 14.4 ± 2.8 | 1.5 ± 0.2 | 9.0 ± 3.3 | 22.9 ± 3.2 | 24.1 ± 3.3 |
| CTGAN | 0.4 ± 0.3 | 21.6 ± 0.5 | 3.0 ± 0.7 | 7.6 ± 0.3 | 24.4 ± 2.4 | 15.8 ± 3.9 |
| TVAE | 0.4 ± 0.1 | 21.0 ± 0.2 | 1.0 ± 0.1 | 9.4 ± 0.2 | 21.5 ± 1.5 | 14.3 ± 1.1 |
| GOGGLE | 0.8 ± 0.9 | 10.6 ± 2.4 | 3.3 ± 0.7 | 17.2 ± 4.9 | 18.8 ± 5.3 | 10.7 ± 1.6 |
| All models+LL | **0.0 ± 0.0** | **0.0 ± 0.0** | **0.0 ± 0.0** | **0.0 ± 0.0** | **0.0 ± 0.0** | **0.0 ± 0.0** |

### 3.4.2 Background Knowledge Alignment

**Quantitative analysis.** In order to assess the ability of standard DGMs to create samples that are aligned with the available background knowledge, I measure the *constraint violation rate* (CVR). Given a set of synthetic samples $\mathcal{S}$ and a set of constraints $\Pi$, CVR computes the percentage of samples in $\mathcal{S}$ violating $\Pi$. Table 3.3 shows the CVR for each tested model and dataset. In particular, the first five rows report the CVR for the standard DGMs, while the last row reports the CVR for all their respective constrained versions (i.e., WGAN+LL, TableGAN+LL, etc.) as it is always equal to zero for all models, datasets and runs. This is expected, since the DGM+LL models offer theoretical guarantees to always generate samples compliant with the constraints. As we can see in Table 3.3, no standard DGM guarantees to produce only samples satisfying the constraints. Further, in more than half of the reported experiments, CVR $> 50\%$ (i.e., more than half of the generated samples violate the constraints), with peaks of CVR $= 100\%$ for WGAN tested on LCLD and CVR $> 95\%$ for four out of five DGMs tested on WiDS.

Additionally, to give a better overview of the results, I present the results according to two other metrics: (i) *constraints violation coverage* (CVC), which represents the percentage of constraints in $\Pi$ that have been violated at least once by any of

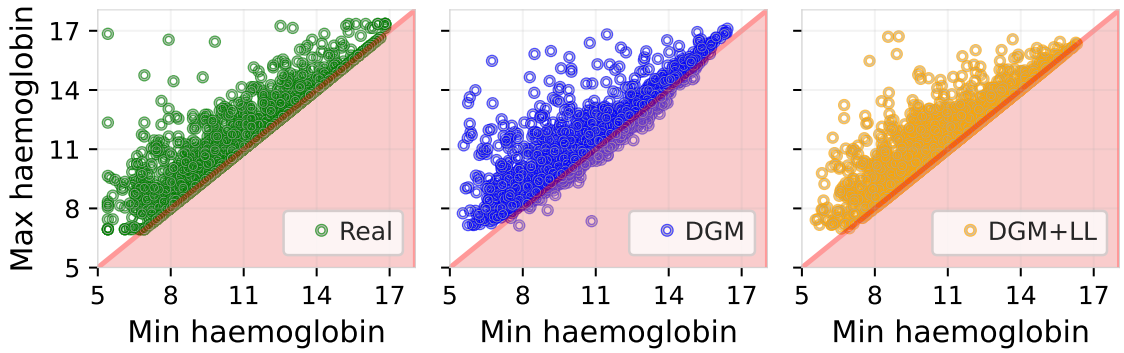



Figure 3.2: Sample distribution for real data and data obtained from an unconstrained TableGAN model and its corresponding constrained TableGAN+LL model on the WiDS dataset.

the samples in $\mathcal{S}$, and (ii) *sample-wise constraints violation coverage* (sCVC), which represents the average over the samples in $\mathcal{S}$ of the percentage of the constraints violated by each sample.

Table 3.4 shows that for 5 out of 6 datasets, for every unconstrained DGM tested, the generated samples collectively violate at least 50% of the constraints, with CVC reaching up to 100% for WiDS, Heloc and News. This entails that the models actually struggle with the majority of the constraints specified, and that the problem cannot be solved by just fixing a very small subset of the available constraints. Meanwhile, all DGM+LL models guarantee that no constraints are violated by any of the samples.

Regarding sCVC, the results in Table 3.5 show that for all models and datasets, on average each sample can violate up to 29.8% of the constraints. As a reminder, violating even one constraint makes the example infeasible in a real setting. The unconstrained DGM models learn different representations that produce different rankings for sCVC. For example, WGAN violates on average the most constraints per samples in LCLD. But, for TableGAN, CTGAN, TVAE and GOGGLE, LCLD is among the datasets with the lowest sCVC. Again, all DGM+LL models ensure that sCVC is 0, meaning not a single constraint is violated for all the samples.

**Qualitative analysis.** In order to visually demonstrate the difference between DGMs and the DGM+LL models, I first select a constraint where only two variables appear: *MaxHaemoglobinLevel* $\geq$ *MinHaemoglobinLevel* from the WiDS dataset. Then, I create two-dimensional scatter-plots of (i) the real data, (ii) the samples generated by an unconstrained DGM, and (iii) the samples generated by the corresponding DGM+LL model, as shown in Figure 3.2 for TableGAN. Note that the

Table 3.6: Results for every DGM+LL and their respective unconstrained DGM versions. The best results are in bold.

| | **Efficacy (↑)** | | | **Detection (↓)** | | |
|---|---|---|---|---|---|---|
| | F1 | *w*F1 | AUC | F1 | *w*F1 | AUC |
| WGAN | 0.463 | 0.488 | 0.730 | 0.945 | 0.943 | 0.954 |
| WGAN+LL | **0.483** | **0.502** | **0.745** | **0.915** | **0.912** | **0.934** |
| TableGAN | 0.330 | 0.400 | 0.704 | 0.908 | 0.907 | 0.926 |
| TableGAN+LL | **0.375** | **0.432** | **0.714** | **0.898** | **0.895** | **0.917** |
| CTGAN | 0.508 | 0.526 | **0.770** | 0.905 | 0.907 | 0.928 |
| CTGAN+LL | **0.510** | **0.533** | **0.770** | **0.893** | **0.896** | **0.922** |
| TVAE | 0.497 | 0.527 | 0.767 | 0.869 | 0.868 | **0.892** |
| TVAE+LL | **0.507** | **0.537** | **0.773** | **0.868** | **0.867** | 0.898 |
| GOGGLE | 0.344 | 0.373 | 0.624 | 0.926 | 0.926 | 0.943 |
| GOGGLE+LL | **0.409** | **0.427** | **0.667** | **0.925** | **0.916** | **0.937** |

variables appearing in the constraint are marked on the axes of the plots. Furthermore, in each plot, I highlight in red the region violating the constraint.

Clearly, the Figure confirms the quantitative analysis presented earlier, as the unconstrained model often violates the constraint, while the constrained counterpart produces outputs only within the feasible regions of the real data. Further, the Figure shows how the distribution of the samples generated by the DGM+LL models match more closely the distribution of the real data.

### 3.4.3 Synthetic Data Quality

*Does background knowledge improve the synthetic data quality?* To answer this question, in this Section, I compare the performance of the standard DGMs with their respective DGM+LL models in terms of machine learning efficacy and detection, as specified in Section 3.4.1. Note that the results for each DGM+LL correspond to the best ordering indicated by the hyperparameter search from Table A.1 of Appendix A.

The results of the conducted experiments are summarised in Table 3.6, where I report the average machine learning efficacy performance (left) and the average detection performance (right) over all datasets with respect to: F1-score (F1), weighted F1-score (*w*F1), and Area Under the ROC Curve (AUC).

As shown in the Table, the DGM+LL models consistently improve upon their standard unconstrained DGM counterparts. In terms of efficacy, the constrained models

show superior or equal performance across every metric for every model. For detection, the constrained models achieve a better (i.e., lower) score in all but one instance: the AUC score for TVAE+LL.

Often, the performance gain is non-negligible. For instance, in efficacy, the DGM+LL models yield an average improvement of 2.8%, 2.3%, and 1.5% w.r.t. the F1-score, weighted F1-score, and AUC, respectively. Further, in the case of GOGGLE, the constrained GOGGLE+LL model registers the highest improvement (of 6.5%) in F1-score for efficiency. These results demonstrate that incorporating background knowledge not only ensures compliance with the constraints but also enhances the overall quality of the synthetic data.

Notice that the machine learning efficacy results obtained on News are reported separately, in Table B.4. The reason is that News is a regression dataset and, as described in Section 3.4.1, the metrics (i.e., Explained Variance and Mean Absolute Error) for this type of data use different scales than the metrics reported for the classification datasets in Table 3.6. For News, in all cases except TableGAN, the DGM+LL models either yield an improvement or no change in the machine learning efficacy performance according to at least one of the two metrics.

For reference, I provide the real data efficacy performance for each model and dataset in Table F.1 of Appendix F.1, also comparing it with the results from the DGM+LL models.

### 3.4.4 Post-processing Ability

*Can LL act as a guardrail at inference time?* The DGM+LL models use the constraints to correct the predictions both at training and inference time. However, the constraints can simply be used at inference time to correct the predictions only once. Indeed, in many practical scenarios, users might not want to modify and retrain their model to add a constraint layer. For example, this might be the case for companies that already have their (possibly black-box) models and/or already generated data that simply needs to be aligned with the available background knowledge. In this Section, I show that LL can be added at inference time as a guardrail on any model without hindering the quality of the generated data.

The results of the conducted analysis are shown in Table 3.7, where P-DGM stands for a standard DGM with LL added as a post-processing step at inference time. The results demonstrate that P-DGMs outperform their standard counterparts 20 times out of 30 (in 2 other cases having equal performance), thus showing the potential of using LL as a post-processor. As expected, the difference in results is often very small,

Table 3.7: Results for every P-DGM and their respective standard version. The best results are in bold.

| | **Efficacy (↑)** | | | **Detection (↓)** | | |
|---|---|---|---|---|---|---|
| | F1 | *w*F1 | AUC | F1 | *w*F1 | AUC |
| WGAN | **0.463** | 0.488 | 0.730 | 0.945 | 0.943 | 0.954 |
| P-WGAN | 0.462 | **0.489** | **0.732** | **0.930** | **0.929** | **0.946** |
| TableGAN | **0.330** | **0.400** | 0.704 | 0.908 | 0.907 | 0.926 |
| P-TableGAN | 0.328 | 0.399 | **0.707** | **0.901** | **0.898** | **0.922** |
| CTGAN | 0.508 | 0.526 | **0.770** | 0.905 | 0.907 | 0.928 |
| P-CTGAN | **0.512** | **0.528** | **0.770** | **0.897** | **0.900** | **0.927** |
| TVAE | **0.497** | **0.527** | **0.767** | **0.869** | **0.868** | **0.892** |
| P-TVAE | 0.495 | 0.524 | **0.767** | 0.875 | 0.876 | 0.902 |
| GOGGLE | 0.344 | 0.373 | 0.624 | 0.926 | 0.926 | 0.943 |
| P-GOGGLE | **0.348** | **0.374** | **0.626** | **0.925** | **0.924** | **0.942** |

Table 3.8: Training time (in minutes). The results are reported for the unconstrained CTGAN models and the constrained CTGAN+LL models, across all datasets, where the models have been trained for the same number of epochs.

| | URL | WiDS | LCLD | Heloc | FSP | News |
|---|---|---|---|---|---|---|
| CTGAN | 10.60 | 19.23 | 54.03 | 10.40 | 1.68 | 18.51 |
| CTGAN+LL | 34.12 | 59.00 | 125.33 | 27.31 | 3.93 | 52.48 |

as the LL layer is only applied at inference time. On the contrary, comparing the P-DGMs models with the DGM+LL models from Table 3.6 shows that the DGM+LL models perform better almost always. This is again expected, as the DGM+LL models are injected with the background knowledge at training time, thus making them able to exploit it.

### 3.4.5 Impact on Generation Time and Training Dynamics

**Training time.** During training, LL requires (i) first mapping the unconstrained samples from the latent space (the neural network space) to the feature space (where the real data reside), at which point LL corrects the samples, and (ii) then mapping the corrected samples back to latent space to continue training. Given this sequence of transformations, incorporating the LL layer into DGMs is expected to introduce a computational overhead. In Table 3.8, I provide the training times for the CTGAN model and its constrained counterpart, across all six datasets, namely URL, WiDS,

Table 3.9: Sample generation time (in seconds).

| | URL | WiDS | LCLD | Heloc | FSP | News |
|---|---|---|---|---|---|---|
| WGAN | 0.02 | 0.03 | 0.01 | 0.00 | 0.00 | 0.01 |
| WGAN+LL | 0.02 | 0.04 | 0.01 | 0.01 | 0.01 | 0.02 |
| TableGAN | 0.18 | 3.21 | 0.17 | 0.17 | 0.18 | 0.20 |
| TableGAN+LL | 0.19 | 3.19 | 0.18 | 0.18 | 0.18 | 0.19 |
| CTGAN | 0.13 | 0.26 | 0.08 | 0.06 | 0.08 | 0.14 |
| CTGAN+LL | 0.14 | 0.27 | 0.08 | 0.06 | 0.08 | 0.14 |
| TVAE | 0.12 | 0.27 | 0.06 | 0.06 | 0.06 | 0.12 |
| TVAE+LL | 0.13 | 0.27 | 0.07 | 0.06 | 0.07 | 0.13 |
| GOGGLE | 0.71 | 3.99 | 9.91 | 0.16 | 0.06 | 2.01 |
| GOGGLE+LL | 0.71 | 3.86 | 10.18 | 0.16 | 0.06 | 2.04 |

LCLD, Heloc, FSP, News. On average, the constrained models need approximately 2.7 times the training duration of their unconstrained counterparts.

**Inference time.** At inference time, however, the workflow differs. While samples must still be transformed from latent to feature space, this operation is necessary also for the unconstrained models. Thus, LL is applied directly to these feature-space samples as a final step. Crucially, this eliminates the inverse transformation (back to the latent space) required during training. This distinction motivates the following investigation.

*Does background knowledge injection affect generation time?* Another important aspect of synthetic data generators is their sample generation time (see, e.g., Kim et al. (2023); Xiao et al. (2022)). To evaluate the overhead of incorporating LL into DGMs at inference time, for each dataset and model I measure the generation time of 1000 samples and report the average results over 5 runs to mitigate system variance (with all runtime experiments conducted on the same machine) in Table 3.9.

In theory, the additional processing brought by LL should strictly add runtime. However, the results show that in 15 out of 30 cases, the constrained versions (the DGMs+LL) are empirically indistinguishable from, or even slightly faster than, their unconstrained counterparts (the DGMs). In particular, in 3 of these cases (namely, for TableGAN and GOGGLE on the WiDS dataset, and for TableGAN on the News dataset), the constrained models' average is slightly lower than the unconstrained models' average runtime. These counter-intuitive discrepancies are small enough to be attributed to the operating system's scheduling variance and background load.

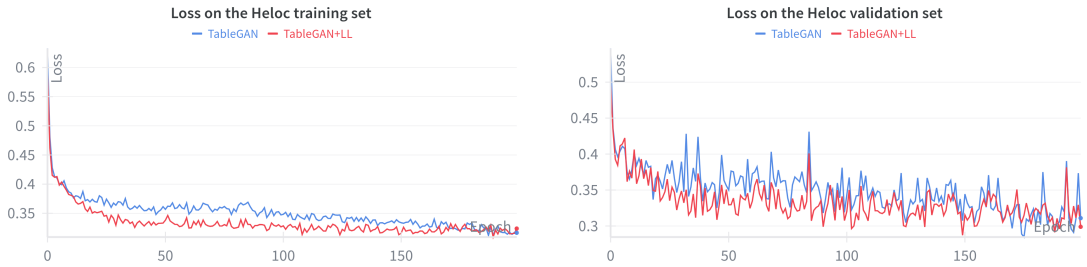



Figure 3.3: Training and validation loss curves (measuring the auxiliary classifier loss of the generator) for the Heloc dataset. The plots compare the convergence of the unconstrained TableGAN baseline against the constrained TableGAN+LL model over the same number of training epochs.

Conversely, in the cases where overhead is observed (i.e., in the remaining 15 out 30 cases), the differences are minimal. Indeed, only one such case (i.e., GOGGLE on LCLD) shows a noticeable difference of 0.27s (however, relative to the base runtime of 9.91s, this represents only an increase of 2.7%), while in the other cases, the constrained DGMs take at most 0.03s more than the unconstrained DGMs.

Overall, LL introduces no statistically significant overhead to the sampling process. To support this claim, I conducted a Wilcoxon signed-rank test across all 30 cases, which yielded a p-value of 0.058 (which is above the standard 0.05 significance threshold, thus, failing to reject the null hypothesis). This result is particularly interesting given that the proposed representation of the constraints within LL may have an exponential size (Dechter, 1999). Indeed, no experiment showed any exponential blow-up, which happens in the worst case, when the constraints have many variables in common.

**Training dynamics.** Finally, to investigate the training dynamics, in Figure 3.3, I show the auxiliary classifier loss of the TableGAN generator trained on the Heloc dataset. I selected this model as its classifier loss is able to directly evaluate the discrepancy between the target feature (which is used in downstream tasks) of a generated sample and the label predicted by TableGAN's classifier for that sample (Park et al., 2018). The Figure empirically validates that the gradients flow through the LL layer. Further, the validation loss closely tracks the training loss, indicating that the LL layer accelerates early convergence and maintains a more stable, lower loss throughout the majority of training, without causing the model to overfit.

## 3.5 Related Work

The framework presented in this Chapter lies at the intersection of standard tabular data synthesis and neuro-symbolic AI for its ability to incorporate background knowledge into neural architectures.

**Tabular Data Synthesis.** Several approaches based on DGMs have been specifically designed to address particular challenges in generating tabular data such as mixed types of features, and imbalanced categorical data. Notable among these are GAN-based approaches like TableGAN (Park et al., 2018), CTGAN (Xu et al., 2019), OCT-GAN (Kim et al., 2021), and IT-GAN (Lee et al., 2021). These methods leverage the power of GANs to model the underlying data distribution and generate synthetic samples that closely resemble real-world tabular data. Few approaches focus especially on particular domains like healthcare where privacy is important (see, e.g., (Choi et al., 2017; Che et al., 2017)). Following privacy concerns, two approaches, i.e., DPGAN (Xie et al., 2018) and PATE-GAN (Jordon et al., 2019), incorporate differential privacy techniques to ensure that the generated synthetic data does not reveal sensitive information about the individuals in the original dataset. Alternatively to GANs, Xu et al. (2019) proposed TVAE as a variation of the standard Variational AutoEncoder, while TabDDPM (Kotelnikov et al., 2023) and STaSy (Kim et al., 2023) were proposed following the achievements of score-based models. Finally, Liu et al. (2022) proposed GOGGLE, a model that uses graph learning to infer relational structure from the data. Although GOGGLE takes a step forward in incorporating prior knowledge into the tabular data generation process, it only allows for injecting background knowledge about existing correlations between pairs of features, differently from the framework I introduced in this Chapter, which can capture any relationship among any group of features that can be expressed as a linear inequality.

**Neuro-symbolic AI Methods.** Neuro-symbolic AI raised great interest in the recent past for many reasons (see, e.g., (Raedt et al., 2018; d'Avila Garcez and Lamb, 2023), among which there is the ability to incorporate complex background knowledge in deep learning models. Standard approaches in the field incorporate logical constraints in the training of a neural network by introducing additional terms in the loss function that penalise the network for violating them (see, e.g., (Diligenti et al., 2012, 2017b; Xu et al., 2018; Fischer et al., 2019; Badreddine et al., 2022; Stoian et al., 2023)). These approaches, while often easy to incorporate into neural models, do not give any guarantee that the constraints are actually satisfied. Alternative approaches, e.g., DeepProbLog (Manhaeve et al., 2018), rely on a solver to inject background

knowledge as a set of definite clauses both at training and inference time. A more recent work (van Krieken et al., 2023) in this line proposed using neural networks for performing approximate inference in polynomial time to address the scalability problem of probabilistic neuro-symbolic learning frameworks. More closely to the framework here, there are the recent methods that are able to incorporate the background knowledge into the topology of the network itself. However, these methods are either only able to deal with constraints expressed in propositional logic (Ahmed et al., 2022b) or less expressive constraints (Giunchiglia and Lukasiewicz, 2021) or become quickly intractable for even moderately complex logical constraints (Hoernle et al., 2022). Regarding the application of neuro-symbolic AI in generative tasks: Liello et al. (2020) incorporate propositional logic constraints on GANs for structured objects generation, while Misino et al. (2022) show how to integrate ProbLog (Raedt et al., 2007) with VAEs. The framework here differs from the last two both in the application domain and in the type of background knowledge incorporated into the neural networks.

## 3.6 Conclusion and Discussion

In this Chapter, I introduced a novel method able to translate complex constraints, expressed as linear inequalities, into a seamlessly-integrated layer, called the Linearly-constrained Layer (LL), within Deep Generative Models (DGMs), thereby yielding Linearly-constrained Deep Generative Models (DGM+LL models). This infusion of domain-specific knowledge directly into the network's architecture facilitates the generation of more realistic tabular data samples, as DGM+LL models are guaranteed to comply with the specified constraints. Furthermore, the conducted experiments reveal that including the new layer during the training phase enhances the quality of the generated samples, thus underlining the importance of constraint integration not only during data generation but also throughout the model's learning process.

In this Chapter, I focused on constraints that can be expressed as linear inequalities. While this covers a wide range of scenarios, some relationships among features may demand more expressive constraint representations. In Chapter 5, I will introduce a framework where even more complex constraints are considered.

# Chapter 4

# Feature Ordering in Linearly-constrained Deep Generative Models

In this Chapter, which is based on (Stoian et al., 2024a) and (Stoian et al., 2025), I propose and evaluate several methods for determining the feature ordering used by the Linearly-constrained Layer (LL). This ordering is critical as it defines the sequence in which the LL corrects the features' values.

## 4.1 Overview and Motivation

The Linearly-constrained Layer (LL), which I introduced in Chapter 3, requires two inputs to be applied to any deep generative model, namely: the constraints provided in a finite set $\Pi$, and an ordering $\lambda$ of the tabular data features, which correspond to a finite set of variables $\mathcal{X} = \{x_1, x_2, \ldots, x_D\}$. The ordering $\lambda : \mathcal{X} \mapsto \{1, 2, \ldots, D\}$ is user-defined and maps every variable (i.e., feature) $x_i$ to a unique number that identifies its position in the order in which LL computes the values of the features.

The Linearly-constrained Layer corrects the value of the $i$-th feature in a given sample whenever that value is not compatible with the values of the features in the $\lambda$ ordering already corrected by LL (i.e., whenever it violates any of the constraints). Thus, given a sample $\tilde{\mathbf{x}}$, the value of LL($\tilde{\mathbf{x}}$) may depend on the chosen $\lambda$ ordering (i.e., different variable orderings may lead to different LL($\tilde{\mathbf{x}}$) values). One advantage of customising the order is that it can be tailored to the user's needs. For example, if users require orderings that consider how closely the generated data distribution matches the real data distribution, there are a number of ways to achieve this, with two of them being presented in this Chapter (i.e., the KDE- and Wasserstein distance-

based methods, as detailed later). Ideally, the ordering should be selected such that it would help in generating samples of the highest possible quality.

In this Chapter, I address the problem of how to order the variables. To this end, my intuition is that features whose distributions (and/or relationships with the other features) are easier to learn should be assigned a lower $\lambda$ value, so that their value can be fruitfully used to compute the values associated with features with higher $\lambda$ values. Using this intuition as a starting point, I design five different methods to determine such variable orderings. As a baseline ordering method, I randomly permute the features. Then, I consider two orderings that depend on the performance of the unconstrained model. In particular, one of the ordering methods I propose relies on comparing the real and synthetic feature-wise distributions using the Wasserstein distance. I selected this metric over the standard f-divergences (such as Jensen-Shannon) or Bregman divergences due to its desirable properties when comparing distributions whose support might not overlap. Indeed, as noted by Zhao et al. (2021) and Srivastava et al. (2019), these standard divergences are often numerically unstable (or undefined) in such cases (a frequent problem with continuous-valued features), whereas the Wasserstein distance remains robust. Moreover, as a separate method, I propose ordering the features based on comparisons between the real and synthetic joint feature distributions by relying on a Kernel Density Estimator. As an alternative, I also propose two methods that order the features based on intrinsic relationships in the data, specifically correlations and cause-effect relationships. Finding the best ordering can be a challenge, as different orderings work better in different scenarios. Nevertheless, through an extensive experimental analysis, I show that three out of the four proposed heuristic-based orderings provide clear improvements in performance over the random-based ordering and over the unconstrained models.

This Chapter is organised as follows: in Section 4.2, I present five different methods to determine an order in which LL will correct the tabular data features, then, in Section 4.3, I conduct an in-depth experimental analysis of how the different orderings perform with each model w.r.t. efficacy and detection, and I conclude in Section 4.4 with a summary of the findings and a discussion on the potential future directions.

## 4.2 Methods to Order Data Features

In this Section, I propose five methods for computing feature orderings, namely: (i) random, (ii) correlation-based, (iii) kernel density estimation-based, (iv) Wasserstein

distance-based, and (v) causal-based methods. These can be grouped along two primary axes: their analytical focus, and their dependency on the unconstrained deep generative model.

First, the methods differ in their analytical focus. The KDE- and Wasserstein distance-based methods prioritise the feature distributions when determining the variable orderings, while the correlation- and causal-based methods focus on the relationships between the features. By definition, the random-based method does not rely on any specific data characteristics.

Second, the methods are distinguished by their model dependency. The random- and causal-based methods are model-independent, requiring only the real dataset $\mathcal{D}$ to establish an ordering. Conversely, the remaining three of my proposed methods (i.e., the correlation-, KDE-, and Wasserstein distance-based methods) are model-dependent, determining the variable orderings according to how well the features are modelled by the standard deep generative models. They require that for each DGM+LL, the corresponding unconstrained DGM is first trained to generate a set of synthetic samples, $\mathcal{S}$. The final ordering is then computed by applying each method's protocol to both the real dataset $\mathcal{D}$ and the generated samples $\mathcal{S}$.

### 4.2.1 Random-based Ordering

Selected as a baseline to compare against the other ordering methods, the random-based ordering method simply generates a random permutation of the integers in the set $\{1, \ldots, D\}$. It is the simplest ordering method because it is model-independent and, unlike other approaches, does not require any information on the distribution of the features, nor on the relationships between the features.

### 4.2.2 Correlation-based Ordering

For establishing an ordering of the features I propose the following steps for the correlation-based method:

1. Given the initial real dataset $\mathcal{D}$ and a DGM, I train the DGM on $\mathcal{D}$ to obtain the synthetic dataset $\mathcal{S}$.

2. I compute the $D \times D$ correlation matrices $\mathbf{C}^{\mathcal{D}}$ and $\mathbf{C}^{\mathcal{S}}$ separately for $\mathcal{D}$ and $\mathcal{S}$, respectively. Precisely, the element of $\mathbf{C}^{\mathcal{D}}$ (resp., $\mathbf{C}^{\mathcal{S}}$) at $(i, j)$ (i.e., $i$-th row, $j$-th column) is computed as the correlation between the vector of the values of the $i$-th column and the vector of the values of the $j$-th column in $\mathcal{D}$ (resp. $\mathcal{S}$).

3. Then, I assign a score $\sigma_i$ to each variable $x_i$, computed as the average of the absolute differences between the correlation between $x_i$ and $x_j$ in the real data and the correlation between $x_i$ and $x_j$ in the synthetic data, over all $x_j$, where $j \neq i$:

$$\sigma_i = \frac{1}{D-1} \sum_{j \neq i} |\mathbf{C}^{\mathcal{S}}_{i,j} - \mathbf{C}^{\mathcal{D}}_{i,j}|.$$

4. Finally, I sort the variables in ascending order of their scores $\sigma_i$.

The intuition is that a lower $\sigma_i$ score indicates that the variable's relationships with other features are well-modelled by the DGM, justifying its placement earlier in the variable ordering $\lambda$.

### 4.2.3 Kernel Density Estimation-based Ordering

A different ordering approach that I propose is a *kernel density estimation (KDE)-based method*. To this end, I design a heuristic that assigns scores to features using a joint density model, ultimately ordering the features based on these scores. The steps of this method are outlined below:

1. Similarly to the correlation-based method above, given the initial real dataset $\mathcal{D}$ and a DGM, I first train the DGM on $\mathcal{D}$ to obtain the synthetic dataset $\mathcal{S}$.

2. Using *scikit-learn*'s [1] (Pedregosa et al., 2011) implementation of KDE, I estimate a probability density function of the multidimensional real data. To this end, I use a Gaussian kernel for the KDE estimator, with bandwidth 1.

3. I then compute the log-likelihood of each sample belonging to $\mathcal{D}$ and $\mathcal{S}$ under the estimated model, using again *scikit-learn*.

4. It is worth noting that *scikit-learn* provides log probability densities, rather than probability values normalised over the sample spaces under evaluation. Thus, I handle this separately by exponentiating and re-normalising them to obtain a valid probability distribution over the samples.

5. Using these probability values, I treat the problem in a discrete setting and approximate the marginal probability mass function of each variable. More specifically, for each feature $x_i$, I determine its unique values from $\mathcal{D} \cup \mathcal{S}$ and, for each unique value $u_{ij}$ I sum the KDE scores of the samples for which $x_i = u_{ij}$,

[1] https://scikit-learn.org/stable/index.html

separately for $\mathcal{D}$ and $\mathcal{S}$. In case no sample is observed with $x_i = u_{ij}$, I set the value of the marginal probability mass function of $x_i$ to 0[2].

6. Hence, I obtain two marginal probability mass functions for each feature $x_i$ (one for the real data $\mathcal{D}$ and one for the synthetic samples $\mathcal{S}$) and compute the Kullback-Leibler divergence (KL) between them to get a single value $\sigma_i$.

7. Finally, I order the features $x_i$ based on their KL scores $\sigma_i$ in ascending order.

One disadvantage of this method is that the marginals are computed in a discrete setting, which may be a coarse approximation for continuous data. Further, its reliability is limited by the curse of dimensionality affecting the initial KDE fit. Nevertheless, this heuristic provides an approach to obtain feature orderings which account for the joint distribution of the features. As future work, one interesting direction would be to use a higher-fidelity approximation of the marginal distributions, based on which the KL divergence scores, and hence the variable ranks in the ordering, are computed.

### 4.2.4 Wasserstein Distance-based Ordering

Previously I described the KDE-based ordering method that compares the joint distributions of the features. However, a simpler alternative method would be to use the Wasserstein distance, which differs from the KDE approach in that it compares individual distributions of the features rather than joint ones. Also known as the earth mover's distance, Wasserstein distance can be interpreted as the minimum effort needed to transform a distribution (e.g., a pile of sand) into another distribution. In particular, the effort refers to the amount of mass (e.g., sand) required to be moved, multiplied by the distance it has to be moved to reach the desired goal. To compute a Wasserstein-based ordering, I follow the steps below:

1. Similarly to the previous two methods, I first train a given DGM on the real data $\mathcal{D}$ to obtain the synthetic dataset $\mathcal{S}$.

2. Then, for each feature, I calculate a score $\sigma_i$ determined by the Wasserstein distance (WD) between the values observed in the feature's real and synthetic data empirical distributions:

$$\sigma_i = \mathrm{WD}(\mathbf{c}_i^{\mathcal{S}}, \mathbf{c}_i^{\mathcal{D}}),$$

[2]It is worth noting that such cases can be dealt with by using a KDE with a triangular or Epanechnikov kernel, which would allow for zero probabilities in the tails.

where $\mathbf{c}_i^{\mathcal{D}}$ (resp. $\mathbf{c}_i^{\mathcal{S}}$) is the vector corresponding to the values of the $i$-th column of $\mathcal{D}$ (resp., $\mathcal{S}$).

3. Finally, I arrange the features in ascending order based on the calculated scores $\sigma_i$, such that the features with generated data distributions furthest from the real data distributions would be assigned a higher $\lambda$ value and would, thus, be corrected later by LL.

It is important to note that the Wasserstein distance can only be defined on a metric space, making it impractical for use with categorical features without additional modifications (e.g., it is possible to define distance metrics for each of the categorical features and then apply Wasserstein distance on these features). However, here I assume that LL can only be applied on continuous features and, thus, the position of the categorical features in the orderings will not impact the final results, as the constraints exclude such features. As an alternative, I investigated using the Jensen-Shannon divergence to derive another ordering that compares individual feature distributions. And while this method offers the advantage that it can be applied on both continuous and categorical features, I found that it was unstable in its results, aligning with the observations made by Zhao et al. (2021).

### 4.2.5 Causal-based Ordering

The final ordering method is, like the random method, model-independent. It relies solely on relationships within the real data, but instead of using statistical correlations, it leverages cause-effect relationships between features. The resulting causal-based ordering adheres to a strict principle: an effect should never appear before its cause. More precisely, for any cause-effect pair of features $(x_i, x_j)$, where $x_i$ is the *cause* and $x_j$ is the *effect*, $x_j$ cannot be assigned a lower $\lambda$ value than $x_i$.

To satisfy this property, the first step is to obtain a directed acyclic graph (DAG) that captures the cause-effect relationships. Since none of the datasets I experiment with provide any information on causal relations between features, I use DAG-GNN (Yu et al., 2019), which is a gradient-based causal discovery method employing graph neural networks to learn a DAG structure which encodes cause-effect relations. However, using such a discovery method poses challenges. In fact, DAG-GNN does not guarantee that its output structure is a DAG, as the acyclicity constraint is enforced via a loss term during training. Consequently, its output is prone to containing self-loops (edges connecting a node to itself) and cycles.

To address these problems and compute the causal-based ordering, I follow the steps below:

1. First, I use DAG-GNN (Yu et al., 2019) to obtain cause-effect relations between the features in the training partition of a given dataset $\mathcal{D}$.

2. Then, I eliminate the self-loops and break the cycles by randomly selecting an edge from each cycle and removing it from the graph.

3. The resulting graph is a DAG, from which I extract the final feature ordering by performing a topological sort.

One important aspect of this method is interpretability. Depending on the dataset, it may be possible to manually check whether the causal relations found by DAG-GNN are sensible. For example, the feature descriptions in the WiDS dataset are clear enough to allow for such a qualitative evaluation. As a future research direction, other causal discovery methods that guarantee a DAG output could be explored as an alternative to DAG-GNN.

## 4.3 Experimental Analysis

While in the previous Chapter, I treated the ordering as a hyper-parameter (varying over the orderings obtained using three of the methods presented here, namely the random-, correlation- and KDE-based methods) and only reported results using the ordering method that yielded the best performance w.r.t. machine learning efficacy and detection, here I focus on the effect of using different feature orderings with LL from the perspective of both of these two measures. Recall from Chapter 3 that machine learning efficacy measures how well a classifier (or regressor) performs when it is trained on synthetic data and tested on real data, while detection measures how well a classifier can distinguish the real from the synthetic data.

### 4.3.1 Ordering Impact on Models Trained with LL

In this Section, I assess the impact of using the five proposed orderings during training. For each DGM+LL model, Table 4.1 summarises (i) the average machine learning efficacy performance over the binary and multiclass classification datasets (i.e., all datasets except News, for which the results can be found in the Appendix B in Table B.4) according to three different metrics: F1-score (F1), weighted F1-score (wF1), and Area Under the ROC Curve (AUC), and (ii) the average detection performance

Table 4.1: Feature orderings comparison for DGMs+LL, where the constraints have been used during both training and inference. The results are reported as averages over all the datasets for detection, and over all classification datasets (i.e., excluding the regression dataset, News) for efficacy. The best results are in bold.

| | | **Efficacy (↑)** | | | **Detection (↓)** | | |
|---|---|---|---|---|---|---|---|
| | | F1 | *w*F1 | AUC | F1 | *w*F1 | AUC |
| WGAN+LL | Random | 0.453 | 0.477 | 0.735 | 0.936 | 0.933 | 0.950 |
| | Correlation | 0.475 | 0.497 | 0.746 | 0.919 | 0.915 | 0.936 |
| | KDE | 0.485 | 0.504 | 0.748 | **0.917** | 0.915 | 0.935 |
| | Wasserstein | 0.442 | 0.478 | 0.731 | 0.921 | 0.916 | 0.938 |
| | Causal | **0.527** | **0.522** | **0.766** | 0.920 | **0.908** | **0.931** |
| TableGAN+LL | Random | 0.346 | 0.409 | 0.710 | 0.908 | 0.904 | 0.926 |
| | Correlation | **0.361** | **0.422** | **0.717** | 0.897 | 0.893 | 0.916 |
| | KDE | 0.349 | 0.413 | 0.706 | 0.895 | 0.893 | 0.916 |
| | Wasserstein | 0.360 | **0.422** | 0.714 | **0.894** | **0.889** | **0.914** |
| | Causal | 0.341 | 0.401 | 0.702 | 0.896 | 0.895 | 0.917 |
| CTGAN+LL | Random | 0.509 | 0.530 | 0.770 | 0.894 | 0.896 | 0.924 |
| | Correlation | 0.514 | 0.536 | 0.772 | 0.887 | 0.890 | 0.920 |
| | KDE | **0.516** | **0.537** | 0.773 | 0.888 | 0.889 | 0.919 |
| | Wasserstein | 0.509 | 0.532 | **0.777** | 0.878 | **0.879** | **0.910** |
| | Causal | 0.506 | 0.528 | 0.769 | **0.873** | **0.879** | 0.912 |
| TVAE+LL | Random | 0.487 | 0.522 | 0.766 | 0.873 | 0.870 | 0.901 |
| | Correlation | 0.504 | 0.534 | 0.771 | **0.868** | **0.868** | **0.898** |
| | KDE | 0.504 | **0.536** | **0.774** | 0.875 | 0.875 | 0.905 |
| | Wasserstein | 0.488 | 0.519 | 0.769 | 0.877 | 0.878 | 0.907 |
| | Causal | **0.506** | **0.536** | 0.771 | 0.876 | 0.877 | 0.907 |
| GOGGLE+LL | Random | 0.384 | 0.406 | 0.653 | **0.922** | **0.916** | **0.937** |
| | Correlation | 0.409 | 0.424 | 0.661 | 0.929 | 0.918 | 0.940 |
| | KDE | 0.393 | 0.416 | 0.661 | 0.928 | 0.926 | 0.941 |
| | Wasserstein | **0.444** | **0.432** | **0.679** | 0.940 | 0.930 | 0.945 |
| | Causal | 0.413 | 0.415 | 0.663 | 0.928 | 0.921 | **0.937** |

over all datasets according to the same three metrics. Note that a higher efficacy score (according to any of the three metrics) indicates better performance. Conversely, a lower detection score indicates better performance. For completeness, the full results for each dataset and each model can be found in Appendix B in Tables B.1-B.7.

First of all, the Table shows that the random-based ordering, which I proposed as a baseline for the rest of the ordering methods, is almost always outperformed by the other four, more sophisticated, ordering methods. Surprisingly, however, it achieves the best detection performance on GOGGLE+LL, showing that even simple ordering methods can be effective in some specific contexts.

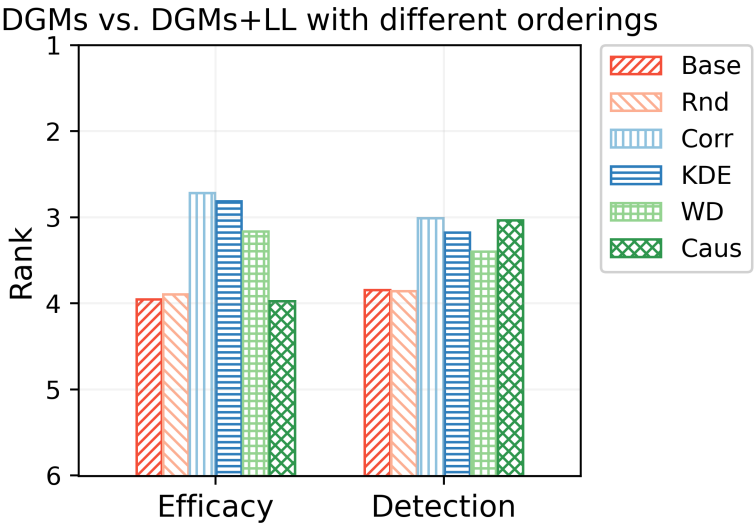



Figure 4.1: The average ranking (over all models, all datasets, and all metrics) w.r.t. the efficacy and detection performance of the unconstrained DGMs (i.e. Base in the plot) and constrained DGM+LL models using five different orderings: random-, correlation-, KDE-, Wasserstein distance-, and causal-based orderings. A lower rank value (i.e. a taller bar) corresponds to a better performance. In case of ties, each tied model is assigned the average of the ranks that would have been assigned to these models if there were no ties.

The correlation-based ordering consistently provides an increase in performance over the random-based ordering across most models. It achieves the best efficacy performance in TableGAN+LL and the best detection scores in TVAE+LL models. While it is not always the best performing ordering, it can be seen as a reliable choice for an ordering that also offers a predictable gain, aside from being based on a simple, well-understood statistical relationship between the features.

The KDE-based ordering yields the best efficacy performance for the more recent and complex GAN- and VAE-based architectures, specifically the CTGAN+LL and TVAE+LL models, although it is less effective for models such as TableGAN+LL or GOGGLE+LL.

The Wasserstein distance-based achieves the best efficacy performance for GOGGLE+LL and the best detection scores for TableGAN+LL. The results suggest that it is a heuristic that generalises well across several different types of deep generative models, similarly to the correlation-based ordering.

Less reliable than the above three orderings (i.e., correlation-, KDE-, and Wasserstein distance-based), the causal-based ordering is not consistently improving the performance over the random ordering, but it presents the highest performance gain

across all of the results, specifically on WGAN+LL in efficacy (i.e., a 16.3% F1-score improvement over the random-based ordering) and detection.

Overall, there is no single variable ordering method that is better across all models and all metrics and the optimal choice appears to be highly dependent on the specific DGM being used (as well as on the specific dataset being used, as shown in the full results from Appendix B in Tables B.1-B.7). Nevertheless, the results show that the ordering methods that are based on statistical properties and that are model-dependent (i.e., correlation, KDE and Wasserstein) tend to be more reliable than methods that solely rely on the relationships between the real data features (e.g., causal-based methods), providing more consistent improvements over the random-based ordering.

To put the above analysis into context, in Figure 4.1, I illustrate the performance rankings of DGM+LL models using each ordering method against the unconstrained DGM baselines. First of all, the Figure clearly shows that applying the LL framework improves performance. Specifically, for both efficacy and detection, the DGM+LL models using three of the four proposed heuristic-based ordering methods (i.e., the correlation-, KDE-, and Wasserstein distance-based methods) achieve on average a better rank (i.e., a lower rank) than the unconstrained DGMs.

Further, the DGM+LL models using these three orderings consistently outperform the random ordering method across both measures. Out of all the orderings, the causal-based one has the highest discrepancy in its ranking between efficacy and detection. Its efficacy rank is worse than that of the random-based ordering, while its detection rank is strong, performing nearly as well as the correlation-based method. This result indicates that the causal-based method is a high-risk, high-reward ordering choice. Finally, the Figure complements and strengthens the earlier analysis, identifying the correlation-based ordering as a particularly reliable choice.

**Adversarially-chosen ordering.** To further investigate the layer's sensitivity to feature ordering, I evaluated the layer with a *reversed causal* ordering using the TableGAN+LL model on the Heloc dataset. This dataset was selected because the causal ordering resulted in the highest F1-score among all orderings used with TableGAN+LL, as shown in Table B.1 of Appendix B. The reversed ordering represents an adversarially-chosen ordering contrary to the optimal causal structure, derived by inverting the topological ordering of the DAG induced by the cause-effect relations among the tabular data features. Interestingly, this reversed causal ordering resulted

in an absolute increase of 1.7% and 1.6% in Area Under the ROC Curve with respect to the causal- and random-based orderings, respectively. This suggests that the adversarial ordering did not negatively impact the model's global ranking capability. However, this ranking robustness did not translate to the F1-score and weighted F1-score performance, indicating that the adversarial ordering degraded the specific decision boundary required for high performance across these two metrics. Indeed, the reversed causal ordering resulted in an absolute decrease of 10.7% and 2.7% in F1- and weighted F1-score, respectively, compared to the causal-based ordering, as well as an absolute decrease of 7.8% and 1.6% in F1- and weighted F1-score, respectively, compared to the random-based ordering.

**Are heuristic-based orderings needed or is the random-based ordering enough?** To rigorously evaluate the random-based baseline ordering, I train TableGAN+LL models on the FSP dataset with the random-based ordering using 35 different seeds. Unlike heuristic-based methods, where the feature ordering is fixed, the random-based method generates a different ordering with every seed, thus inducing high variance in the order in which the features are corrected by LL. Therefore, a larger number of seeds is necessary to fully capture the performance variance that is intrinsic to the baseline, whereas 5 seeds are sufficient to quantify the variance arising from initialisation and training dynamics for the heuristic-based ordering methods. As shown in Figure 4.2, the heuristic-based orderings result in a lower variance compared to the broad spread of the random baseline, particularly regarding detection. While the random-based ordering occasionally yields individual results with high efficacy or low detection, it is not a reliable method due to its high unpredictability.

This lack of reliability is further highlighted in Figure 4.3, where I show the relationship between efficacy and detection scores. The random-based ordering yields a Pearson correlation coefficient of just 0.03, indicating a decoupling of the two metrics. In other words, a high efficacy score obtained with a random-based ordering is just as likely to be paired with poor detection performance as it is with good performance. This lack of correlation, combined with the extreme variance observed in the boxplots from Figure 4.2, shows that the random-based ordering offers no control on the trade-off between machine learning efficacy and detection.

**Navigating the machine learning efficacy vs. detection trade-off.** To analyse the overall impact of the different feature orderings relative to the two measures (i.e., machine learning efficacy and detection), I compare the results for three datasets

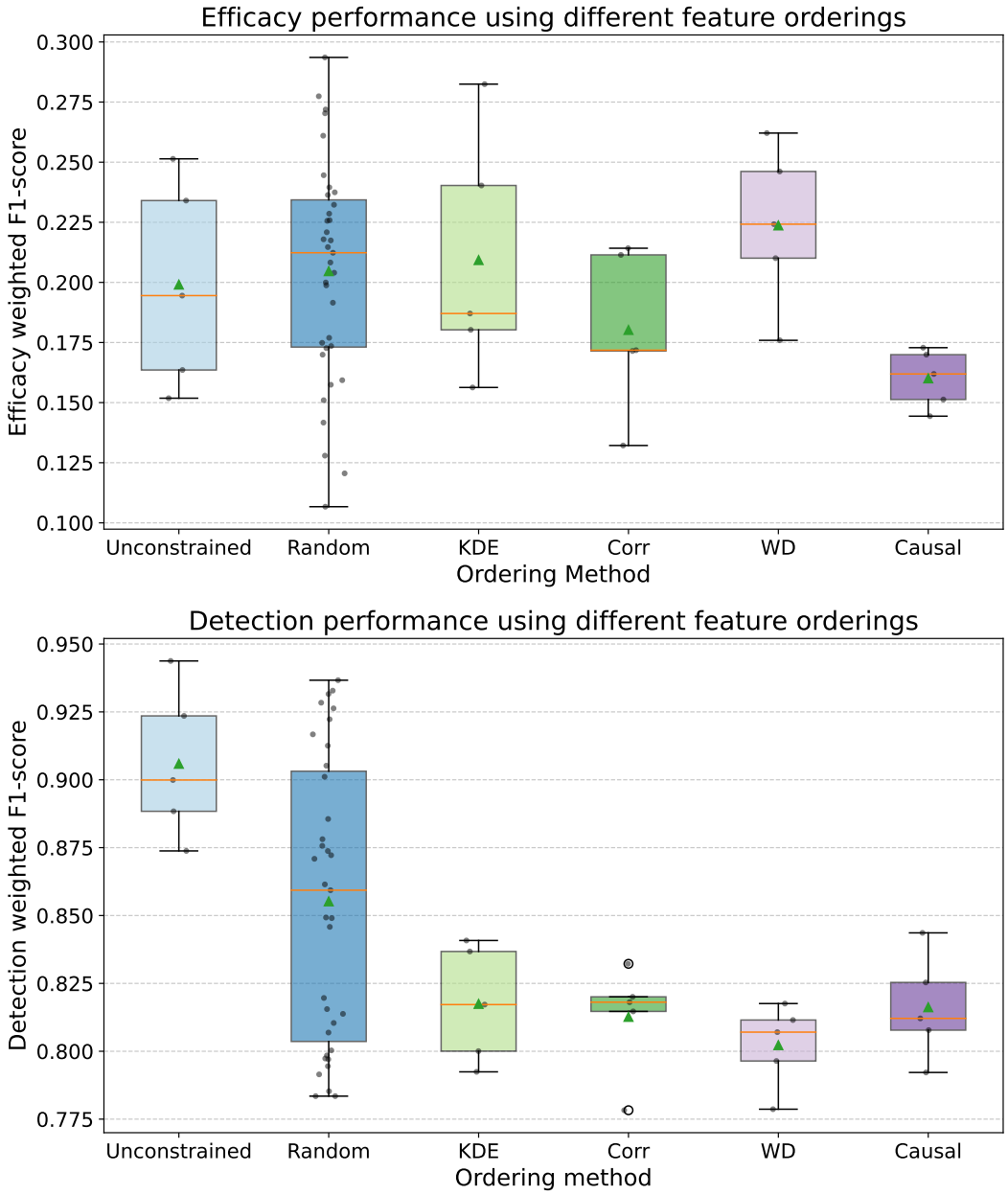



Figure 4.2: Boxplots for machine learning efficacy (top), and detection (bottom), both in terms of weighted F1-score. The results are reported for the unconstrained TableGAN model and for the constrained TableGAN+LL models using different feature orderings (random-, KDE-, correlation- (corr), WD-, and causal-based), each trained with 5 different seeds on the FSP dataset, except for random-based ordering where 35 seeds were used. For each model, each point (i.e., circular marker) represents an individual result, and the mean (resp., median) of these results are shown using a green triangle (resp., an orange line). The maximum/minimum values are shown for each model (the top and bottom black horizontal bars, along with a box (in varying colours), which captures the middle 50% of the points (i.e., the points that lie between the 25th and the 75th percentile of the data). The circles represent outliers.

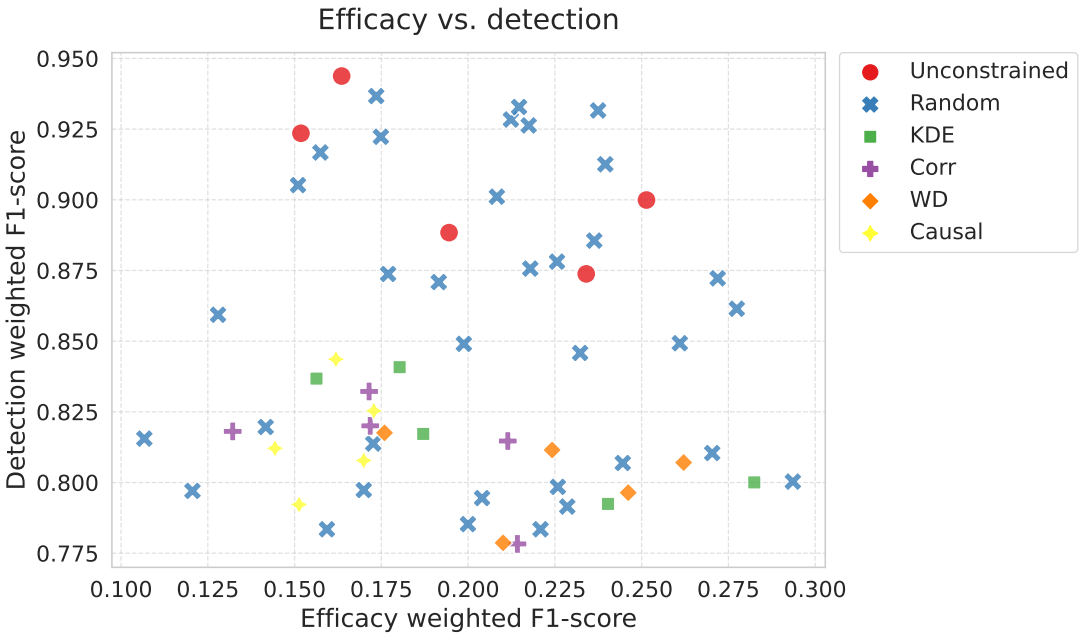



Figure 4.3: Machine learning efficacy vs. detection (in terms of weighted F1-score) for TableGAN models, unconstrained and constrained using different feature orderings (random-, KDE-, correlation- (corr), WD-, and causal-based), trained on the FSP dataset.

that capture the diverse range of dataset complexities examined in the experiments, namely: (i) FSP, the smallest (multi-class) classification dataset with only 2K training samples, (ii) LCLD, the largest (binary) classification dataset with more than 490K samples, and (iii) News, a regression dataset having 31K samples.

For the smallest dataset (FSP) the trade-off between the two measures is most pronounced. However, the Wasserstein distance-based ordering for TableGAN+LL successfully navigates this challenge, lowering detection scores across all three metrics when compared to the unconstrained TableGAN (e.g., from 90.6% to 80.2%, the lowest among competitive models, w.r.t. the weighted F1-score), while also improving machine learning efficacy. A similar trend is observed for CTGAN. On the other hand, this "win-win" dynamic is not observed in WGAN or GOGGLE, which follow the traditional trade-off (where lower detection scores are accompanied by lower efficacy), nor in TVAE. For the large-scale dataset (LCLD), using heuristic-based orderings has a positive impact on detection. In particular, TVAE+LL with correlation-based ordering achieves a detection weighted F1-score of 79.5%, which is notably lower (better) than all of the unconstrained models' performance (e.g., WGAN with 99.9% or TableGAN with 89.5%). Simultaneously, it maintains a competitive efficacy score. In such cases where data volume is high, the results indicate that using a correlation-

Table 4.2: Feature orderings comparison for P-DGMs, where the constraints have been used only for post-processing the generated data. The results are reported as averages over all the datasets for detection, and over all classification datasets (i.e., excluding the regression dataset, News) for efficacy. Best results are in bold.

| | | **Efficacy (↑)** | | | **Detection (↓)** | | |
|---|---|---|---|---|---|---|---|
| | | F1 | *w*F1 | AUC | F1 | *w*F1 | AUC |
| P-WGAN | Random | 0.456 | 0.484 | 0.731 | **0.930** | **0.928** | **0.945** |
| | Correlation | 0.462 | **0.489** | **0.732** | 0.931 | 0.930 | 0.947 |
| | KDE | **0.463** | **0.489** | 0.731 | 0.933 | 0.931 | 0.948 |
| | Wasserstein | 0.459 | 0.487 | 0.730 | 0.931 | 0.929 | 0.947 |
| | Causal | 0.460 | 0.488 | 0.730 | 0.931 | 0.929 | 0.947 |
| P-TableGAN | Random | 0.328 | 0.398 | 0.705 | 0.902 | 0.901 | 0.925 |
| | Correlation | 0.328 | 0.399 | 0.703 | 0.900 | 0.900 | **0.922** |
| | KDE | 0.326 | 0.398 | **0.706** | 0.901 | 0.901 | 0.924 |
| | Wasserstein | **0.330** | **0.400** | 0.702 | 0.900 | 0.901 | 0.923 |
| | Causal | 0.329 | 0.398 | 0.704 | **0.899** | **0.899** | 0.923 |
| P-CTGAN | Random | 0.508 | 0.524 | 0.769 | 0.899 | 0.901 | 0.926 |
| | Correlation | 0.508 | 0.527 | 0.770 | 0.899 | 0.902 | 0.928 |
| | KDE | 0.507 | 0.524 | 0.769 | 0.899 | 0.903 | 0.927 |
| | Wasserstein | **0.515** | **0.532** | **0.772** | 0.903 | 0.903 | 0.927 |
| | Causal | 0.509 | 0.524 | 0.767 | **0.898** | **0.900** | **0.925** |
| P-TVAE | Random | 0.490 | 0.521 | 0.764 | 0.879 | 0.879 | 0.905 |
| | Correlation | 0.493 | 0.523 | 0.763 | 0.876 | 0.879 | 0.904 |
| | KDE | **0.494** | **0.524** | **0.767** | 0.876 | **0.875** | **0.901** |
| | Wasserstein | 0.493 | **0.524** | 0.761 | 0.877 | 0.879 | 0.906 |
| | Causal | 0.489 | 0.519 | 0.766 | **0.875** | 0.877 | 0.904 |
| P-GOGGLE | Random | 0.347 | 0.373 | **0.626** | **0.925** | 0.925 | 0.943 |
| | Correlation | **0.348** | **0.375** | 0.625 | 0.929 | 0.926 | 0.945 |
| | KDE | 0.347 | 0.374 | **0.626** | 0.929 | 0.927 | 0.944 |
| | Wasserstein | 0.344 | 0.372 | 0.625 | 0.929 | 0.926 | 0.944 |
| | Causal | 0.347 | 0.373 | **0.626** | **0.925** | **0.920** | **0.939** |

based feature ordering is a robust method to lower detection risk, while also achieving high efficacy in downstream tasks. For the regression dataset (News), the correlation- and causal-based orderings allow the model to generate data that is both more useful in practice (i.e., lower MAE scores for the machine learning efficacy) and often harder to detect compared to the random-based ordering.

### 4.3.2 Ordering Impact on Models Post-processed with LL

In this Section, I assess the impact of using the five proposed orderings when LL is applied as a postprocessing step. For each P-DGM model, Table 4.2 summarises (i)

the average machine learning efficacy performance over the binary and multi-class classification datasets (i.e., all datasets except News, for which the results can be found in the Appendix B in Table B.11), and (ii) the average detection performance over all datasets. For completeness, the full results for each dataset and each model can be found in Appendix B in Tables B.8-B.14.

For efficacy, the results show that the key contrast with the scenario where LL is used during training is the scale of improvement. Indeed, when using LL during training, the maximum improvement brought by the more complex orderings over the random-based ordering is of 7.4% w.r.t. F1-score (recorded for WGAN+LL with the causal-based ordering in Table 4.1), while when using LL only during post-processing, the maximum improvement is of only 0.7% w.r.t. F1-score (for P-CTGAN with the Wasserstein distance-based ordering).

A similar trend can be seen for detection, where the more complex orderings barely surpass the random-based ordering baseline. The causal-based ordering, for example, frequently ranks as the best method for detection (e.g., for P-CTGAN and for P-GOGGLE w.r.t. all three metrics, and for P-TableGAN w.r.t. two out of three metrics). In spite of this, the performance gains are minimal. The largest improvement it offers over the random baseline is of only 0.4% on the F1-score (recorded for P-TVAE). Further, in some cases (e.g., P-WGAN) the more sophisticated orderings perform worse than the random baseline.

As opposed to the clearer trends observed for the DGM+LL models, it is difficult to establish patterns in the preference of the P-DGMs towards any of the orderings and to distinguish trends where the heuristic-based orderings might be more helpful than the random-based ordering. This is mainly due to the very small performance differences between the ordering methods, as it can be seen from the Table. Thus, the simpler random-based method often suffices for selecting a feature ordering for LL when constraining the DGMs only as a post-processing step.

## 4.4 Conclusions

The work conducted in this Chapter highlights the importance of choosing a feature ordering in an informed way when applying LL during training. Complementing the findings from Chapter 3, the results here show that in order to achieve state-of-the-art performance, a heuristics-based ordering method, such as correlation-based, as opposed to random-based, must be used when training with LL. Furthermore, the analysis of the efficacy-detection trade-off from Section 4.3.1 shows that the choice of

feature ordering depends on the dataset complexity, as well as on the underlying model architectures. While the correlation-based ordering generally offers a robust balance for large-scale or regression datasets (e.g., LCLD and News), it is not universally optimal. For small, data-scarce environments (e.g., FSP), the results indicate that distance-based orderings (specifically, the Wasserstein distance) are better suited, especially when the underlying model has a GAN-based architecture. In these scenarios, informed orderings provide a win-win outcome (i.e., lowering detection without sacrificing efficacy), a balance that the random-based ordering typically fails to achieve.

Crucially, the findings reveal it is not simply that ordering matters, but also when it is applied. Indeed, there is a strong contrast in the results between applying LL with different orderings during the training and applying it only as a post-processing step. When used during training, informed orderings can offer large improvements over the random-based baseline. For example, the causal-based ordering improved the F1-score of WGAN+LL by 7.4% when evaluating efficacy. Conversely, when used for post-processing, the benefits of informed orderings are small, if any. With a maximum efficacy gain of only 0.7%, the additional complexity of these methods often provides little advantage over a simple random ordering, making the random-based ordering sufficient in many cases when applying LL as a post-processing step.

While none of the experimented orderings can be considered a universal gold standard (as results indicate the best choice depends on the specific model and dataset), future work could explore methods to learn an optimal feature ordering for each specific case. To this end, recent approaches such as Implicit Maximum Likelihood Estimation (IMLE) (Niepert et al., 2021) allow for backpropagation through combinatorial optimisation solvers, theoretically enabling the model to learn a discrete ordering that maximises the specific loss function. A similar approach could be applied here to find an optimal feature ordering, albeit at significant computational cost, as IMLE would require solving, e.g., an integer linear programming (ILP) at each step or epoch. To mitigate these costs, a simpler approach could involve dynamically adjusting the ordering after a set number of training iterations, although this would necessitate rebuilding the LL layer each time the ordering changes. Alternatively, a reinforcement learning module could be integrated to explicitly learn the optimal sequence as part of the training process.

# Chapter 5

# Deep Generative Modelling with Quantifier-Free Linear Real Arithmetic Constraints

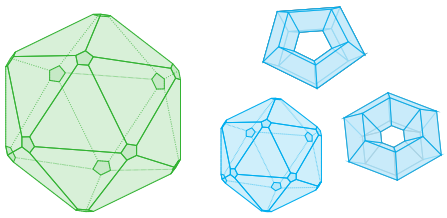
In this Chapter, which is based on (Stoian and Giunchiglia, 2025), I introduce a framework for deep generative modelling using background knowledge which can define non-convex and even disconnected spaces.

## 5.1 Motivation and Overview

The problem of tabular data generation is a critical area of research, driven by its numerous practical applications across various domains. High-quality synthetic data offer solutions to pressing challenges such as data scarcity (Choi et al., 2017), bias in unbalanced datasets (van Breugel et al., 2021), and the general need for privacy protection (Lee et al., 2021). However, due to the varied nature of the data distributions in the tabular domain—which are often multi-modal, and present complex dependencies among features—it is difficult to create models able to generate realistic data. Indeed, no matter the Deep Generative Model (DGM) used, when synthetic datapoints are tested for alignment with the available background knowledge, they frequently fail such a test. Even when considering simple knowledge like *"the feature representing the maximum recorded level of haemoglobin should be greater than or equal to the one representing its minimum"*, DGMs often generate datapoints violating it. So far, this problem has only been solved by either rejecting the non-aligned samples, or by adding a layer to the DGM that restricts its output space to coincide with the one defined by linear inequalities (Stoian et al., 2024a), as presented in Chapter 3. However, while the first solution is not feasible in the presence of a high

violation rate, the second is only available when the knowledge can be captured by linear inequalities, which have limited expressivity.

In this Chapter, I propose a novel layer—called Disjunctive Refinement Layer (DRL)—able to constrain any DGM output space according to background knowledge expressed as quantifier-free linear real arithmetic (QFLRA) formulae. QFLRA formulae can capture any relationship over the features that can be represented as a combination of conjunctions, disjunctions and negations of linear inequalities. Due to their expressivity, QFLRA formulae can define spaces that are not only non-convex but can also be disconnected. On the other hand, linear inequalities can only capture convex output spaces. See Figure 5.1 for an example of spaces defined by linear inequalities and QFLRA formulae.



(a) (b)

Figure 5.1: Example of spaces defined by (a) a set of linear inequalities and (b) a set of QFLRA formulae.

While linear inequalities establish a single lower and upper bound (if existent) for each feature, QFLRA formulae define multiple intervals where the background knowledge holds, each with its own boundaries. This significantly increases the complexity of the problem, as compiling knowledge into DRL not only requires keeping track of these intervals but also deriving the intricate hidden interactions among variables. And, while integrating such constraints into DGMs to guarantee their satisfaction is increasingly challenging, it is essential, as real-world data often involve non-linear relationships among features, e.g., such as those captured using constraints of the type *if ... then ...* and/or disjoint intervals.

**Example 5.1.1.** *Consider the following background knowledge statement: "The value of $x_5$ should be always at least $x_1$ and, if greater than $x_2$, then it should also be at least equal to $x_3$. Furthermore, $x_5$ should never be greater than $x_4$". Clearly, this statement cannot be expressed by a set of linear inequalities. Instead, it corresponds to the QFLRA formula:*

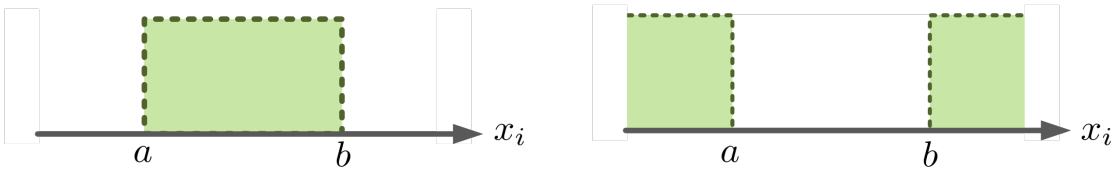
$$(x_5 \geq x_1) \wedge \big((x_5 > x_2) \rightarrow (x_5 \geq x_3)\big) \wedge (x_5 \leq x_4). \tag{5.1}$$

*Moreover, this formula entails other hidden relations among the variables such as, e.g., $\neg(x_1 > x_4)$, which can be immediately derived from the first term and the last term of the disjunction.*

To derive such additional hidden relations, I introduce a novel variable elimination method which generalises the analogous procedure for systems of linear inequalities based on the Fourier-Motzkin result (see, e.g., (Dechter, 1999)). Once built, by definition, DRL (i) guarantees the satisfaction of the constraints, (ii) can be seamlessly added to the architecture of any neural network model, (iii) allows the backpropagation of the gradients at training time, (iv) performs all the computations in a single forward pass (i.e., no cycles), and (v) given a sample generated by a DGM, it returns a new one that is optimal with respect to the original (intuitively, which minimally differs from the original sample while taking into account the user preferences on which features should be changed first).

I conduct an extensive experimental analysis, which shows that adding DRL to DGMs improves their machine learning efficacy (Xu et al., 2019) on a range of different scenarios. This is the most widely used performance measure for evaluating the quality of synthetic data, as it assesses how useful the generated data is for downstream tasks. In particular, I considered five different DGMs (i.e., WGAN (Arjovsky et al., 2017), TableGAN (Park et al., 2018), CTGAN (Xu et al., 2019), TVAE (Xu et al., 2019), and GOGGLE (Liu et al., 2022)) and added DRL into their topology. The results show that DRL yields improvements for all datasets of up to 21.4%, 20.5%, and 20.9% in terms of F1-score, weighted F1-score, and Area Under the ROC Curve, respectively. Finally, the experiments demonstrate a strong need for a method like the one proposed here. Indeed, DGMs generate synthetic datapoints violating the background knowledge more often than expected. In 13 out of 25 scenarios, the DGMs produced datasets with over 50% datapoints violating the constraints, and in five cases this reached 100%. Clearly, the standard rejection sampling technique cannot be used in the latter case. Moreover, I show that even in the other cases, where not all samples produced by the DGMs violate the constraints, rejection sampling hinders both the efficacy and the sample generation runtime. On the other hand, DRL not only improves the machine learning efficacy, but is able to provide runtimes comparable to the unconstrained DGMs.

This Chapter is organised as follows: in Section 5.2 I first provide the problem statement, then in Section 5.3 I introduce DRL, the first-ever layer that can be integrated into any DGM to enforce background knowledge expressed as QFLRA formulae, in Section 5.4 I show experimentally how integrating DRL in DGMs improves their machine learning efficacy, even when the constraints define a possibly non-convex and disconnected space, in Section 5.5 I describe the related work, and,




(a) (b)

Figure 5.2: Example of valid spaces w.r.t. variable $x_i$ defined by (a) a set of linear inequalities of the form $\{x_i \geq a, x_i \leq b\}$ and (b) a QFLRA constraint of the form $(x_i \leq a) \vee (x_i \geq b)$.

finally, in Section 5.6 I provide concluding remarks along with a discussion on future directions.

## 5.2 Problem Statement

Suppose the following are given: an unknown distribution $p_X$ over $X \in \mathbb{R}^D$, a training dataset $\mathcal{D}$ consisting of $N$ independent and identically distributed (i.i.d.) samples drawn from $p_X$, and formally expressed background knowledge about the problem. In particular, the background knowledge states which samples are admissible and which are not. Recall from Chapter 3, *constrained generative modelling* is defined as the problem of learning the parameters $\theta$ of a generative model, such that (i) the model distribution $p_\theta$ approximates $p_X$, and (ii) the sample space of $p_\theta$ is aligned with what is stated in the background knowledge. As described in the introduction, so far this problem has only been solved by either rejecting the non-aligned samples or by including a layer into the DGM that restricts its output space to coincide with the one defined by the linear inequalities (Stoian et al., 2024a).

Here, the background knowledge can be expressed as a set of formulae, each being a disjunction of linear inequalities. This allows for capturing any relationships among features which can be represented as Quantifier-Free Linear Real Arithmetic (QFLRA) formulae. Indeed, through syntactic manipulation using De Morgan's laws and the mathematical properties of linear inequalities, any knowledge formulated as a combination of conjunctions, disjunctions, and negations of linear inequalities can be rewritten as a set of disjunctions over linear inequalities. Formally, I consider a set $\Pi$ of constraints, where a *constraint* is a disjunction of $n_\Psi \in \mathbb{N}$ linear inequalities of the form:

$$\Psi = \Phi_1 \vee \Phi_2 \vee \cdots \vee \Phi_{n_\Psi}, \tag{5.2}$$

where each $\Phi_i$ is a linear inequality over the set of variables $\mathcal{X} = \{x_k \mid k = 1, \ldots, D\}$, each variable uniquely corresponding to a feature in the dataset. Further, I assume each linear inequality has the following form:

$$\sum_k w_k x_k + b \geq 0, \tag{5.3}$$

with $w_k \in \mathbb{R}$, $b \in \mathbb{R}$, and $x_k$ ranging over $\mathbb{R}$. Variable $x_i$ *occurs* in (5.3) whenever $w_i \neq 0$. Moreover, $x_i$ *occurs positively* if $w_i > 0$ and *negatively* otherwise.

For an ease of notation, given two linear expressions $\varphi = \sum_k w_k x_k + b$ and $\varphi' = \sum_k w'_k x_k + b'$, I write $(\varphi + \varphi')$ for the linear expression $\sum_k (w_k + w'_k) x_k + (b + b')$. Similarly, I write $(\varphi - \varphi')$ and $\varphi / w$, if $w \in \mathbb{R} \setminus \{0\}$. Additionally, I express the linear inequality (5.3) as $w_i x_i + \varphi \geq 0$, by this implicitly assuming that $w_i \neq 0$ and that $x_i$ does not occur in $\varphi$, i.e., that $\varphi = \sum_{k \neq i} w_k x_k + b$.

Given a DGM with distribution $p_\theta$, a *sample* $\tilde{\mathbf{x}}$ drawn from $p_\theta$ is an assignment to the variables in $\mathcal{X}$, and $\tilde{x}_k$ indicates the value assigned by $\tilde{\mathbf{x}}$ to the variable $x_k$. Then, a sample $\tilde{\mathbf{x}}$ *satisfies*

- the linear inequality (5.3) if $\sum_k w_k \tilde{x}_k + b \geq 0$,
- the constraint $\Psi$ with form (5.2) if $\Phi_i$ is satisfied by $\tilde{\mathbf{x}}$ for some $i \in \{1, \ldots, n_\Psi\}$, and
- a set $\Pi$ of constraints if $\tilde{\mathbf{x}}$ satisfies all the constraints in $\Pi$.

Further, each linear inequality $\Phi$ (resp. constraint $\Psi$, resp. set of constraints $\Pi$) is associated with the set $\Omega(\Phi)$ (resp. $\Omega(\Psi)$, resp. $\Omega(\Pi)$) of the points in $\mathbb{R}^D$ that satisfy $\Phi$ (resp. $\Psi$, resp. $\Pi$). Clearly, $\Omega(\Phi)$, $\Omega(\Psi)$ and $\Omega(\Pi)$ define a subspace of $\mathbb{R}^D$ and have the following properties:

1. $\Omega(\Phi)$ is non-empty and convex,
2. $\Omega(\Psi)$ is non-empty but may be non-convex and also disconnected, and
3. $\Omega(\Pi)$ may be empty or non-convex and also disconnected,

all the above assuming some variable occurs in $\Phi$, $\Psi$ and $\Pi$. A linear inequality $\Phi$ (resp. a constraint $\Psi$, resp. a set of constraints $\Pi$) is *violated* by a sample $\tilde{\mathbf{x}}$ if $\tilde{\mathbf{x}}$ does not belong to the corresponding set $\Omega(\Phi)$ (resp. $\Omega(\Psi)$, resp. $\Omega(\Pi)$). A linear inequality $\Phi$ (resp. a constraint $\Psi$, resp. a set of constraints $\Pi$) is *satisfiable* if the corresponding set $\Omega(\Phi)$ (resp. $\Omega(\Psi)$, resp. $\Omega(\Pi)$) is not empty.

For the sake of simplicity, I do not consider strict inequalities (i.e., inequalities with $>$). From a theoretical perspective, the entire theory can be easily generalised to consider them. From a practical perspective, each strict inequality can be simply rewritten as an inequality of the following form: $\sum_k w_k x_k + b > 0$ as $\sum_k w_k x_k + b - \epsilon \geq 0$, where $\epsilon > 0$ denotes the desired precision of the representation, taking into account the limitations of floating-point accuracy.

Finally, I represent each constraint $\Psi \in \Pi$ of the form given in (5.2) also as the set $\{\Phi_1, \Phi_2, \ldots, \Phi_{n_\Psi}\}$. Under this notation, $\Pi$ is a set of sets of linear inequalities. Hence, the linear inequalities in a set should be interpreted as disjunctively defining a constraint in $\Pi$, while the constraints should be interpreted as conjunctively defining $\Pi$.

## 5.3 Disjunctive Refinement Layer

Given a finite set of constraints $\Pi$ and a DGM, in this Section, I propose a method for constructing a layer that satisfies all of the following desirable properties: (i) it guarantees satisfaction of the constraints in $\Pi$, (ii) it can be seamlessly integrated into the architecture of any DGM, (iii) it supports gradient backpropagation during training, (iv) all computations can be performed in a single forward pass (i.e., without introducing cycles), and (v) given a sample $\tilde{\mathbf{x}}$ generated by the DGM, it returns a new sample that is optimal w.r.t. $\tilde{\mathbf{x}}$.

Here, I assume that the function $\min(\mathcal{S})$ over a finite set $\mathcal{S}$ of values in $\mathbb{R}$ is defined as follows: $\min(\emptyset) = +\infty$, and for any $v \in \mathbb{R}$ and finite set $\mathcal{S}' \subset \mathbb{R}$,

$$\min(\{v\} \cup \mathcal{S}') = \begin{cases} v, & \text{if } v \leq \min(\mathcal{S}'), \\ \min(\mathcal{S}'), & \text{otherwise.} \end{cases}$$

An analogous definition applies to the function $\max(\mathcal{S})$, with $\max(\emptyset) = -\infty$.

Before illustrating the general case in Section 5.3.2, I will present a basic case in Section 5.3.1 and assume $\Pi$ is a finite set of constraints on a single variable $x_i$.

### 5.3.1 Single Variable Case

Given an arbitrary variable $x_i$, a constraint $\Psi$ of form (5.2) may contain multiple linear inequalities of the form (5.3) in which $x_i$ occurs. More precisely, any inequality written as $w_i x_i + \varphi \geq 0$ in which $x_i$ occurs negatively (resp., positively) defines an upper (resp., a lower) bound on $x_i$ of the form $x_i \leq -\frac{\varphi}{w_i}$ (resp., $x_i \geq -\frac{\varphi}{w_i}$). Then, finding the largest interval (or union of intervals) in which $x_i$ can take values that

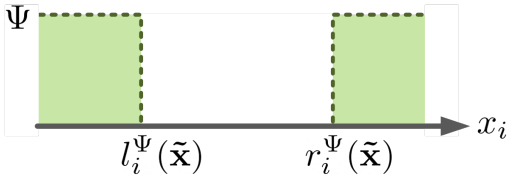



Figure 5.3: Visualisation of the left and right boundaries defined by constraint $\Psi$. The green regions correspond to the values of $x_i$ such that $x_i \in \Omega(\Psi)$.

satisfy $\Psi$ amounts to identifying the largest upper bound and the smallest lower bound on $x_i$ imposed by $\Psi$. I will refer to these bounds the *left* and *right bound*[1], respectively.

Formally, each constraint $\Psi$ defines a single *left bound* $l_i^\Psi$ and a single *right bound* $r_i^\Psi$ on variable $x_i$ as follows:

$$\begin{aligned} l_i^\Psi &= \max_{(w_i x_i + \varphi \geq 0) \in \Psi : w_i < 0} \Big( -\frac{\varphi}{w_i} \Big), \\ r_i^\Psi &= \min_{(w_i x_i + \varphi \geq 0) \in \Psi : w_i > 0} \Big( -\frac{\varphi}{w_i} \Big). \end{aligned} \tag{5.4}$$

In a single variable case, assuming $\Psi$ contains a linear inequality in which $w_i \neq 0$, a sample $\tilde{\mathbf{x}}$ satisfies $\Psi$ if and only if either $\tilde{x}_i \leq l_i^\Psi$ or $\tilde{x}_i \geq r_i^\Psi$, as represented in Figure 5.3[2]. As the Figure clearly shows, $\Omega(\Psi)$ is already non-convex whenever both $l_i^\Psi$ and $r_i^\Psi$ are finite and $l_i^\Psi < r_i^\Psi$.

The above discussion pertains to a single constraint $\Psi$. In general, computing the intervals for the whole set of constraints $\Pi$ requires finding the satisfying boundaries for each constraint in $\Pi$ and then ordering these boundaries. However, given a sample $\tilde{\mathbf{x}}$ violating some constraint in $\Pi$, in order to obtain a new sample that is optimal w.r.t. $\tilde{\mathbf{x}}$ it is enough to set $\mathrm{DRL}(\tilde{\mathbf{x}})_i$ equal to the bound that satisfies $\Pi$ and that is at minimal Euclidean distance from $\tilde{x}_i$. To this end, I first define the *closest satisfying left and right boundary for* $\tilde{x}_i$ as:

$$\begin{aligned} l_i^\Pi(\tilde{\mathbf{x}}) &= \max_{\Psi \in \Pi}(\{l_i^\Psi : \tilde{x}_i > l_i^\Psi, l_i^\Psi \in \Omega(\Pi)\}), \\ r_i^\Pi(\tilde{\mathbf{x}}) &= \min_{\Psi \in \Pi}(\{r_i^\Psi : \tilde{x}_i < r_i^\Psi, r_i^\Psi \in \Omega(\Pi)\}), \end{aligned} \tag{5.5}$$

[1] I use the terms "left" and "right" because, in any non-vacuous constraint, the largest upper bound will be less than the smallest lower bound.

[2] For ease of notation, I will write $x_i \in \Omega(\Psi)$ to denote the values of $x_i$ that do not violate $\Psi$. The same convention applies to $\Omega(\Phi)$, $\Omega(\Pi)$.

respectively. Then, $\text{DRL}(\tilde{\mathbf{x}})_k = \tilde{x}_k$ for $k \neq i$ (as the set of constraints $\Pi$ is written over a single variable, $x_i$), and

$$\text{DRL}(\tilde{\mathbf{x}})_i = \begin{cases} \tilde{x}_i & \text{if } \tilde{\mathbf{x}} \in \Omega(\Pi), \\ l_i^{\Pi}(\tilde{\mathbf{x}}) & \text{if } \tilde{\mathbf{x}} \notin \Omega(\Pi) \text{ and } |\tilde{x}_i - l_i^{\Pi}(\tilde{\mathbf{x}})| < |\tilde{x}_i - r_i^{\Pi}(\tilde{\mathbf{x}})|, \\ r_i^{\Pi}(\tilde{\mathbf{x}}) & \text{otherwise.} \end{cases} \tag{5.6}$$

By construction, $\text{DRL}(\tilde{\mathbf{x}})$ satisfies the constraints in $\Pi$ and is *optimal w.r.t.* $\tilde{\mathbf{x}}$. In other words, there does not exist another sample satisfying $\Pi$ with smaller Euclidean distance from $\tilde{\mathbf{x}}$.

**Lemma 5.3.1.** *Let $\Pi$ be a finite and satisfiable set of constraints in a single variable $x_i$. For every sample $\tilde{\mathbf{x}}$, DRL($\tilde{\mathbf{x}}$) satisfies $\Pi$ and is optimal w.r.t. $\tilde{\mathbf{x}}$.*

*Proof.* The proof is in two parts. First, I will prove that for every sample $\tilde{\mathbf{x}}$, $\text{DRL}(\tilde{\mathbf{x}})$ satisfies $\Pi$. If $\tilde{\mathbf{x}} \in \Omega(\Pi)$, then by definition $\text{DRL}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}$, and the result of the Lemma trivially follows. Now consider the other possible case where $\tilde{\mathbf{x}} \notin \Omega(\Pi)$. I will prove that $\text{DRL}(\tilde{\mathbf{x}}) \in \Pi$ by contradiction. Assume, for the sake of contradiction, that $\text{DRL}(\tilde{\mathbf{x}}) \notin \Pi$. Then, $\Pi \neq \emptyset$ and, since $\Pi$ is a satisfiable set of constraints, it must be the case that $l_i^{\Pi}(\tilde{\mathbf{x}})$ and $r_i^{\Pi}(\tilde{\mathbf{x}})$ exist and at least one of them is finite. By definition, $l_i^{\Pi}(\tilde{\mathbf{x}})$ and $r_i^{\Pi}(\tilde{\mathbf{x}})$ satisfy $\Pi$. Thus, $\text{DRL}(\tilde{\mathbf{x}})$ also satisfies $\Pi$ (by the definition of $\text{DRL}(\tilde{\mathbf{x}})$ in (5.6)), and a contradiction is reached.

In the second part, I will prove that for every sample $\tilde{\mathbf{x}}$, $\text{DRL}(\tilde{\mathbf{x}})$ is optimal w.r.t. $\tilde{\mathbf{x}}$. If $\tilde{\mathbf{x}} \in \Omega(\Pi)$, then by definition $\text{DRL}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}$, and the result of the Lemma trivially follows. Now consider the other possible case where $\tilde{\mathbf{x}} \notin \Omega(\Pi)$. It suffices to show that for every $\tilde{\mathbf{x}}$, $\text{DRL}(\tilde{\mathbf{x}})_i$ is the solution of $\Pi$ with minimal Euclidean distance from $\tilde{x}_i$.

Let $d$ be the minimum Euclidean distance between any point in $\Omega(\Pi)$ and $\tilde{\mathbf{x}}$. Let $\mathbf{r}$ and $\mathbf{l}$ be two samples with $r_k = l_k = \tilde{x}_k = \text{DRL}(\tilde{\mathbf{x}})_k$ for all $k \neq i$ and $k \in \{1, \ldots, D\}$, and $l_i = \tilde{x}_i - d$ and $r_i = \tilde{x}_i + d$. Either $\mathbf{l}$ or $\mathbf{r}$, or both, belong to $\Omega(\Pi)$.

Now let $\mathbf{v}$ be $\mathbf{l}$ if $\mathbf{l} \in \Omega(\Pi)$, and $\mathbf{r}$ otherwise. By definition, $\mathbf{v} \in \Omega(\Pi)$ and is optimal w.r.t. $\tilde{\mathbf{x}}$. This means that there is no other sample $\mathbf{v}' \in \Omega(\Pi)$ such that $|\tilde{x}_i - v'_i| < |\tilde{x}_i - v_i| = d$. Hence, there must exist a constraint $\Psi$ such that:

1. $v_i = l_i^{\Psi}$, if $\mathbf{v} = \mathbf{l}$. From the optimality of $\mathbf{v}$, it must then be the case that $l_i^{\Pi}(\tilde{\mathbf{x}}) = l_i^{\Psi}$. Then, by construction, $\text{DRL}(\tilde{\mathbf{x}})_i = l_i^{\Pi}(\tilde{\mathbf{x}})$.

2. $v_i = r_i^{\Psi}$, otherwise. From the optimality of $\mathbf{v}$, it must then be the case that $r_i^{\Pi}(\tilde{\mathbf{x}}) = r_i^{\Psi}$. Then, by construction, $\text{DRL}(\tilde{\mathbf{x}})_i = r_i^{\Pi}(\tilde{\mathbf{x}})$.

Clearly, in both cases, $\text{DRL}(\tilde{\mathbf{x}}) = \mathbf{v}$. Thus, $\text{DRL}(\tilde{\mathbf{x}})$ is optimal w.r.t. $\tilde{\mathbf{x}}$. □

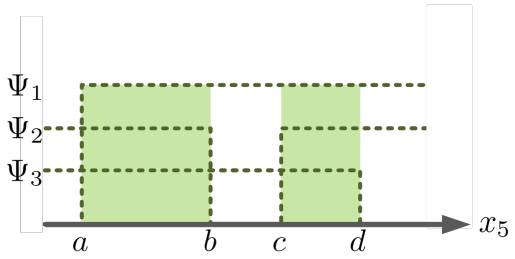



Figure 5.4: Constraints for Example 5.3.2. The green regions correspond to the values of $x_5 \in \Omega(\Pi)$.

**Example 5.3.2.** *Let* $\Pi$ *be the set of constraints* $\{\Psi_1, \Psi_2, \Psi_3\}$ *over the unique variable* $x_5$*, with* $\Psi_1, \Psi_2$ *and* $\Psi_3$ *defined as follows:*

$$\begin{aligned}\Psi_1 &= (x_5 \geq a),\\ \Psi_2 &= (x_5 \leq b) \vee (x_5 \geq c),\\ \Psi_3 &= (x_5 \leq d).\end{aligned}$$

*The constraints, along with the set* $\Omega(\Pi)$*, can be visualised in Figure 5.4. The left and right boundaries corresponding to each constraint are as follows:*

$$\begin{aligned} l_5^{\Psi_1} &= -\infty, & l_5^{\Psi_2} &= b, & l_5^{\Psi_3} &= d,\\ r_5^{\Psi_1} &= a, & r_5^{\Psi_2} &= c, & r_5^{\Psi_3} &= +\infty. \end{aligned}$$

*Depending on the value of* $\tilde{x}_5$*,* $DRL(\tilde{\mathbf{x}})_5$ *gets correspondingly different values. In particular:*

1. *if* $\tilde{x}_5 < a$ *then* $DRL(\tilde{\mathbf{x}})_5 = a$,
2. *if* $a \leq \tilde{x}_5 \leq b$ *then* $DRL(\tilde{\mathbf{x}})_5 = \tilde{x}_5$,
3. *if* $b < \tilde{x}_5 < \frac{b+c}{2}$ *then* $DRL(\tilde{\mathbf{x}})_5 = b$,
4. *if* $\frac{b+c}{2} \leq \tilde{x}_5 < c$ *then* $DRL(\tilde{\mathbf{x}})_5 = c$,
5. *if* $c \leq \tilde{x}_5 \leq d$ *then* $DRL(\tilde{\mathbf{x}})_5 = \tilde{x}_5$,
6. *if* $\tilde{x}_5 > d$ *then* $DRL(\tilde{\mathbf{x}})_5 = d$.

*Independently from the value of* $\tilde{x}_5$*,* $DRL(\tilde{\mathbf{x}})_5$ *satisfies the constraints and is optimal w.r.t.* $\tilde{\mathbf{x}}$.

### 5.3.2 General Case

As stated in the beginning of this Section, one of the desiderata for DRL is that all the necessary computations need to be done in a single forward pass. To this end, consider a *variable ordering* $x_1; x_2; \ldots; x_D$ corresponding to the order of computation of the features. The ordering can be arbitrarily selected or, more appropriately, may

reflect the user preferences on which features should be changed first when the sample violates the constraints. Indeed, the value of each feature $x_i$ will be computed taking into account the values of the features $x_1, \ldots, x_{i-1}$, with the latter considered immutable. To make this possible, when building the layer, it is crucial to ensure that the chosen value for the variables $x_j$, with $j < i$, guarantees the existence of a value for $x_i$ satisfying the constraints.

Starting from $\Pi_D = \Pi$ and $i = D$, this amounts to deriving a finite set $\Pi_{i-1}$ of constraints in the variables $x_1, x_2, \ldots, x_{i-1}$ such that, for every value of $x_1, x_2, \ldots, x_{i-1}$ satisfying $\Pi_{i-1}$, there must exist a value for $x_i$ satisfying $\Pi_i$ (or, alternatively, each assignment to $x_1, x_2, \ldots, x_{i-1}$ that satisfies $\Pi_{i-1}$ can be extended to satisfy also $\Pi_i$). In order to define $\Pi_{i-1}$, it is possible to use standard propositional and cutting planes (CP) rules, as defined, e.g., in (Krajícek, 1998), to derive a new rule that can be used to construct $\Pi_{i-1}$ with the desired properties stated above. I refer to this rule as the *cutting planes resolution rule* and formally describe it in Definition 5.3.3 below.

**Definition 5.3.3.** *Given two constraints,* $\Psi = (\bigvee_{k=1}^{n}(w_k x_i + \varphi_k \geq 0) \vee \Phi)$ *and* $\Psi' = (\bigvee_{j=1}^{m}(w'_j x_i + \varphi'_j \geq 0) \vee \Phi')$*, with* $w'_1, \ldots, w'_m < 0 < w_1, \ldots, w_n$ *and* $m, n \geq 1$*, the cutting planes (CP) resolution rule between* $\Psi$ *and* $\Psi'$ *on* $x_i$ *is as follows:*

$$\frac{\bigvee_{j=1}^{m}(w'_j x_i + \varphi'_j \geq 0) \vee \Phi' \qquad \bigvee_{k=1}^{n}(w_k x_i + \varphi_k \geq 0) \vee \Phi}{\bigvee_{j=1}^{m}\bigvee_{k=1}^{n}(\varphi_k/w_k - \varphi'_j/w'_j \geq 0) \vee \Phi \vee \Phi'}. \tag{5.7}$$

In the above rule, $\Psi$ and $\Psi'$ are the *premises*, and the formula below the line is the *conclusion* denoted with $CPres_i(\Psi, \Psi')$, which forms a new constraint.

**Lemma 5.3.4.** *The CP resolution rule is sound: the premises entail the conclusion of the rule.*

*Proof.* Given the CP resolution rule (5.7) between $\Psi$ and $\Psi'$ on $x_i$ from Definition 5.3.3, where $\Psi$ and $\Psi'$ are the *premises* and $CPres_i(\Psi, \Psi')$ is the conclusion, proving that the rule is sound amounts to proving that if the premises hold, then the conclusion also holds.

To this end, consider an arbitrary instantiation $\tilde{\mathbf{x}}$ of the variables $x_1, x_2, \ldots, x_D$. There are two cases:

1. If $\Phi$ and $\Phi'$ hold, then the conclusion trivially holds.

2. If the premises hold, but none of $\Phi$ and $\Phi'$ are true, then it must be the case that:

$$\tilde{x}_i \geq \min_{k=1}^{n} -\frac{\varphi_k(\tilde{\mathbf{x}})}{w_k} \qquad \text{and} \qquad \tilde{x}_i \leq \max_{j=1}^{m} -\frac{\varphi'_j(\tilde{\mathbf{x}})}{w'_j},$$

where, $\varphi(\tilde{\mathbf{x}})$ is simply the value obtained by substituting $\tilde{x}_j$ for each variable $x_j$ in a given expression $\varphi$.

Clearly, if these two inequalities hold, then the following must be true:

$$\min_{k=1}^{n} -\frac{\varphi_k(\tilde{\mathbf{x}})}{w_k} \leq \max_{j=1}^{m} -\frac{\varphi'_j(\tilde{\mathbf{x}})}{w'_j}.$$

This means there is a pair $(j, k)$ such that $-\frac{\varphi_k(\tilde{\mathbf{x}})}{w_k} \leq -\frac{\varphi'_j(\tilde{\mathbf{x}})}{w'_j}$. Thus, the conclusion of the CP resolution rule must hold.

□

As Lemma 5.3.4 states, the rule is sound for any $\Phi$ and $\Phi'$ for which the assumptions hold. However, for the sake of simplicity, I assume that (i) $\Phi$ does not contain positive occurrences of $x_i$, and (ii) $\Phi'$ does not contain negative occurrences of $x_i$. As shown later, it is possible to impose much stronger conditions on the applicability of the rule, still enabling the derivation of a set of constraints $\Pi_{i-1}$ with the desired properties.

**Example 5.3.5** (Example 5.1.1, cont'd)**.** *Recall that the QFLRA formula in (5.1) from the first example translates to the set of constraints* $\Pi = \{\Psi_1, \Psi_2, \Psi_3\}$ *with:* $\Psi_1 = (x_5 \geq x_1), \Psi_2 = ((x_5 \leq x_2) \vee (x_5 \geq x_3))$ *and* $\Psi_3 = (x_5 \leq x_4)$*. By repeatedly applying the CP resolution rule, a new set of constraints is obtained that (i) is entailed by* $\Pi$ *and (ii) is logically equivalent to the statement that there is a value for* $x_5$ *such that* $\bigwedge_{\Psi \in \Pi} \Psi$ *holds. The step-by-step process is as follows:*

1. *First I identify the pairs of constraints* $\Psi$ *and* $\Psi'$ *on the variable* $x_5$*, where* $x_5$ *occurs only positively in* $\Psi$ *and negatively in* $\Psi'$*. There are only two such pairs, namely: (*$\Psi_1$*,* $\Psi_2$*) and (*$\Psi_1$*,* $\Psi_3$*). By applying the rule the following new constraints are derived:*

$$CPres_5(\Psi_1, \Psi_2) = ((x_1 \leq x_2) \vee (x_5 \geq x_3)),$$
$$CPres_5(\Psi_1, \Psi_3) = (x_1 \leq x_4).$$

2. *Then, I identify all pairs of constraints* $\Psi$ *and* $\Psi'$ *on the variable* $x_5$*, where (i)* $\Psi$ *is in* $\Pi$ *and* $\Psi'$ *is in the set of constraints previously derived using the CP resolution rule, and (ii)* $x_5$ *occurs positively in* $\Psi$ *and negatively in* $\Psi'$*. There is only one such pair, namely (*$\Psi_3$*,* $CPres_5(\Psi_1, \Psi_2)$*), from which the following new constraint is obtained:*

$$CPres_5(\Psi_3, CPres_5(\Psi_1, \Psi_2)) = ((x_1 \leq x_2) \vee (x_3 \leq x_4)).$$

*As it is derived from the multiple application of the CP resolution rule, the above set of constraints admits a solution for $x_5$ if and only if $(x_1 \leq x_4) \wedge (x_1 \leq x_2 \vee x_3 \leq x_4)$.*

In the example, there is only one constraint with both positive and negative occurrences of $x_i$ and the CP resolution of any two distinct constraints always leads to a conclusion with either only positive or only negative occurrences of $x_i$. However, in general, the CP resolution of two constraints $\Psi$ and $\Psi'$ will lead to a new constraint $CPres_i(\Psi, \Psi')$ which might contain both positive and negative occurrences of $x_i$. This new constraint can be one of the premises of other CP resolutions, which can produce new constraints and the process can iterate. Nevertheless, the goal is to derive the constraints in the variables $x_1, \ldots, x_{i-1}$ whose satisfying assignments can be extended to satisfy also the constraints with $x_i$. The standard solution to make all the possible CP resolutions on $x_i$ while considering also the conclusion of the already done resolution may turn out to be too computationally expensive. To avoid such scenarios, it is possible to further restrict to CP resolutions between two constraints $\Psi$ and $\Psi'$ in which $\Psi$ does not contain negative occurrences of $x_i$. To this end, let:

1. $\Pi_i^+$ be the set of constraints in $\Pi_i$ with only positive occurrences of $x_i$ (i.e., without negative occurrences of $x_i$);

2. $\Pi_i^-$ be the set of constraints in $\Pi_i$ with only negative occurrences of $x_i$ (i.e., without positive occurrences of $x_i$);

3. $\Pi_i^\pm$ be the set of constraints in $\Pi_i$ in which $x_i$ occurs both positively and negatively;

4. $\Pi_i^\ddagger$ be the set of constraints obtained by the recursive application of the CP-resolution between one constraint without negative occurrences of $x_i$ and one constraint in $\Pi_i^\pm$:

$$\Pi_i^\ddagger = \bigcup_{k=0}^{|\Pi_i^\pm|} \Pi_i^k$$

where $\Pi_i^0 = \Pi_i^+$ and

$$\Pi_i^k = \{CPres_i(\Psi, \Psi') \mid \Psi \in \Pi_i^{k-1}, \Psi' \in \Pi_i^\pm\}, \text{ for any } k > 0.$$

By construction, every constraint in $\Pi_i^\ddagger$ has only positive occurrences of $x_i$.

Then, $\Pi_{i-1}$ is the set of constraints in $\Pi_i$ in which $x_i$ does not occur plus the set of constraints obtained by the CP resolution of the constraints in $\Pi_i^+ \cup \Pi_i^- \cup \Pi_i^{\pm}$. More precisely,

$$\Pi_{i-1} = (\Pi_i \setminus (\Pi_i^+ \cup \Pi_i^- \cup \Pi_i^{\pm})) \cup \{CPres_i(\Psi, \Psi') \mid \Psi \in \Pi_i^{\pm}, \Psi' \in \Pi_i^-\}. \tag{5.8}$$

Clearly, for any $i$, the set of constraints $\Pi_{i-1}$ does not contain any occurrence of $x_i$ and can contain a non-polynomial number of constraints, the latter fact echoing similar results for variable elimination methods in propositional logic and sets of linear inequalities (Dechter, 1999). The above definition generalises the standard variable elimination procedure for systems of linear inequalities—based on the Fourier-Motzkin result—to disjunctions of linear inequalities.

**Example 5.3.6** (Example 5.3.5, cont'd)**.** *Consider the variable ordering* $x_1, x_2, x_3, x_4, x_5$. *Then,* $\Pi_5 = \Pi, \Pi_5^- = \{\Psi_3\}, \Pi_5^{\pm} = \{\Psi_2\}, \Pi_5^+ = \Pi_5^0 = \{\Psi_1\}, \Pi_5^1 = \{((x_1 \leq x_2) \vee (x_5 \geq x_3))\}$, *and* $\Pi_5^{\pm} = \Pi_5^0 \cup \Pi_5^1$. *As a consequence,* $\Pi_4 = \{(x_1 \leq x_4), ((x_1 \leq x_2) \vee (x_3 \leq x_4))\}$, *and* $\Pi_3 = \Pi_2 = \Pi_1 = \emptyset$.

For each set of constraints $\Pi_i$, the set $\Pi_{i-1}$ has the desired properties above, as formally stated in Lemma 5.3.7.

**Lemma 5.3.7.** *Let* $\Pi$ *be a set of constraints in the variables* $x_1, \ldots, x_i$. *Then* $\Pi_i = \Pi$ *and* $\Pi_{i-1}$ *are equisatisfiable, and each assignment to the variables* $x_1, \ldots, x_{i-1}$ *satisfying* $\Pi_{i-1}$ *can be extended in order to satisfy* $\Pi_i$.

*Proof.* Clearly, given the soundness of the CP resolution rule shown in Lemma 5.3.4, if $\Pi_i$ is satisfiable, then $\Pi_{i-1}$ is also satisfiable. This is because every constraint in $\Pi_{i-1}$ that is not in $\Pi_i$ is entailed by $\Pi_i$.

It remains to show that if $\tilde{\mathbf{x}}^{:i}$ is an assignment to the variables $x_1, \ldots, x_{i-1}$ satisfying $\Pi_{i-1}$, the set of constraints $\Pi_i(\tilde{\mathbf{x}}^{:i})$ is satisfiable. Similarly to the notation used in the proof of Lemma 5.3.4, the expression $\Pi_i(\tilde{\mathbf{x}}^{:i})$ denotes the set of constraints in the variable $x_i$ obtained by substituting the value $\tilde{x}_j^{:i}$ for the variable $x_j$ in the constraints in $\Pi$, for all $j < i$.

Assume, for the sake of contradiction, that $\Pi_i(\tilde{\mathbf{x}}^{:i})$ is not satisfiable. Then, there are two constraints $\Psi, \Psi'$ in $\Pi_i(\tilde{\mathbf{x}}^{:i})$ such that[3] $\Psi \equiv (x_i \geq r_i)$ and $\Psi' \equiv (x_i \leq l_i)$. There are two possible cases:

[3] I use $\equiv$ to denote logical equivalence.

1. In this case, $r_i > l_i$. The constraint $CPres_i(\Psi, \Psi')$ belongs to $\Pi_{i-1}(\tilde{\mathbf{x}}^{:i})$ and is equivalent to $r_i \leq l_i$. Thus, a contradiction is reached.

2. In the remaining case, $r_i \leq l_i$, and there are additional $n \geq 1$ constraints $\{\Psi_1, \Psi_2, \ldots, \Psi_n\}$ in $\Pi_i(\tilde{\mathbf{x}}^{:i})$, with each $\Psi_j \equiv (x_i \leq l_i^{\Psi_j}) \vee (x_i \geq r_i^{\Psi_j})$, such that:

$$\begin{aligned} &l_i^{\Psi_1} < r_i \leq r_i^{\Psi_1}, \\ &l_i^{\Psi_2} < r_i^{\Psi_1} \leq r_i^{\Psi_2}, \\ &\ldots, \\ &l_i^{\Psi_n} < r_i^{\Psi_{n-1}} \leq l_i < r_i^{\Psi_n}. \end{aligned}$$

   This simply results in the following:

$$l_i^{\Psi_1} < r_i \leq r_i^{\Psi_1} \leq r_i^{\Psi_2} \leq \ldots \leq r_i^{\Psi_{n-1}} \leq l_i < r_i^{\Psi_n},$$

   and thus $l_i^{\Psi_1} < r_i \leq l_i < r_i^{\Psi_n}$. In this case, $\Omega(\Psi) \cap \Omega(\Psi') \cap \left(\bigcap_{j=1}^{n} \Omega(\Psi_j)\right) = \emptyset$.

   Given this construction, $\Pi_i^{+}(\tilde{\mathbf{x}}^{:i})$ contains the following constraints:

$$\begin{aligned} &\Upsilon_1 = CPres_i(\Psi, \Psi_1) \equiv (l_i^{\Psi_1} \geq r_i) \vee (x_i \geq r_i^{\Psi_1}) \equiv x_i \geq r_i^{\Psi_1}, \\ &\Upsilon_2 = CPres_i(\Upsilon_1, \Psi_2) \equiv (l_i^{\Psi_2} \geq r_i^{\Psi_1}) \vee (x_i \geq r_i^{\Psi_2}) \equiv x_i \geq r_i^{\Psi_2}, \\ &\ldots, \\ &\Upsilon_n = CPres_i(\Upsilon_{n-1}, \Psi_n) \equiv (l_i^{\Psi_n} \geq r_i^{\Psi_{n-1}}) \vee (x_i \geq r_i^{\Psi_n}) \equiv x_i \geq r_i^{\Psi_n}. \end{aligned}$$

   Therefore, $\Pi_{i-1}(\tilde{\mathbf{x}}^{:i})$ contains $CPres_i(\Upsilon_n, \Psi') \equiv l_i \geq r_i^{\Psi_n}$, which contradicts the above.

□

As a corollary of the above lemma we have that the CP resolution is *refutationally complete*. More precisely, if $\Pi$ is unsatisfiable then it is possible to derive a disjunction of linear inequalities in $\Pi_0$ in which each inequality of the form (5.3) has $w_i = 0$ for $i = 1, \ldots, D$ and $b < 0$. Thus, at the end of the layer construction, $\Pi$ unsatisfiability is automatically detected.

**Corollary 5.3.8.** *For any finite set of constraints $\Pi$, the CP resolution rule is refutationally complete: if $\Pi$ is unsatisfiable, then repeated application of the CP resolution rule will eventually derive a contradiction.*

Starting from $i = 1$, the value of $\mathrm{DRL}(\tilde{\mathbf{x}})_i$ is computed considering the constraints in $\widetilde{\Pi}_i$, where $\widetilde{\Pi}_i$ is the set of constraints in the variable $x_i$ obtained by substituting $\mathrm{DRL}(\tilde{\mathbf{x}})_1, \mathrm{DRL}(\tilde{\mathbf{x}})_2, \ldots, \mathrm{DRL}(\tilde{\mathbf{x}})_{i-1}$ for the variables $x_1, x_2, \ldots, x_{i-1}$ in $\Pi_i$. As in the single variable case, assuming $\tilde{x}_i$ violates some constraint in $\widetilde{\Pi}_i$, I define the *closest satisfying left and right boundary for* $\tilde{x}_i$ as:

$$\begin{aligned} l_i^{\widetilde{\Pi}_i}(\tilde{\mathbf{x}}) &= \max_{\Psi \in \widetilde{\Pi}_i}(\{l_i^{\Psi} : \tilde{x}_i > l_i^{\Psi}, l_i^{\Psi} \in \Omega(\widetilde{\Pi}_i)\}), \\ r_i^{\widetilde{\Pi}_i}(\tilde{\mathbf{x}}) &= \min_{\Psi \in \widetilde{\Pi}_i}(\{r_i^{\Psi} : \tilde{x}_i < r_i^{\Psi}, r_i^{\Psi} \in \Omega(\widetilde{\Pi}_i)\}). \end{aligned} \tag{5.9}$$

Then,

$$\mathrm{DRL}(\tilde{\mathbf{x}})_i = \begin{cases} \tilde{x}_i & \text{if } \tilde{x}_i \in \Omega(\widetilde{\Pi}_i), \\ l_i^{\widetilde{\Pi}_i}(\tilde{\mathbf{x}}) & \text{if } \tilde{x}_i \notin \Omega(\widetilde{\Pi}_i) \text{ and } |\tilde{x}_i - l_i^{\widetilde{\Pi}_i}(\tilde{\mathbf{x}})| < |\tilde{x}_i - r_i^{\widetilde{\Pi}_i}(\tilde{\mathbf{x}})|, \\ r_i^{\widetilde{\Pi}_i}(\tilde{\mathbf{x}}) & \text{otherwise.} \end{cases} \tag{5.10}$$

**Example 5.3.9** (Examples 5.3.2, 5.3.6, cont'd)**.** *Consider a sample* $\tilde{\mathbf{x}}$ *where* $\tilde{x}_1 = a, \tilde{x}_2 = b, \tilde{x}_3 = c$*, and* $\tilde{x}_4 = d$*, arranged as in Figure 5.4. Then, since* $\Pi_3 = \Pi_2 = \Pi_1 = \emptyset$*, DRL does not change the values of the features* $x_1$*,* $x_2$*, and* $x_3$*, and so:* $DRL(\tilde{\mathbf{x}})_1 = a$*,* $DRL(\tilde{\mathbf{x}})_2 = b$*, and* $DRL(\tilde{\mathbf{x}})_3 = c$*. For* $\tilde{x}_4$*, we have that* $\widetilde{\Pi}_4 = \{(x_4 \geq a), ((a \leq b) \vee (x_4 \geq c))\}$ *which reduces to* $\{(x_4 \geq a)\}$*. Since* $\widetilde{\Pi}_4$ *is satisfied,* $DRL(\tilde{\mathbf{x}}) = d$*. Finally,* $\widetilde{\Pi}_5 = \{(x_5 \geq a), ((x_5 \leq b) \vee (x_5 \geq c)), (x_5 \leq d)\}$ *and the value of* $DRL(\tilde{\mathbf{x}})_5$ *can be computed on the ground of* $\tilde{x}_5$*, as detailed in Example 5.3.2.*

**Theorem 5.3.10.** *Let* $\Pi$ *be a finite and satisfiable set of constraints. For any sample* $\tilde{\mathbf{x}}$ *and variable ordering, the corresponding sample* $DRL(\tilde{\mathbf{x}})$ *satisfies* $\Pi$*.*

*Proof.* The statement can be proved by induction over the number $n$ of variables appearing in $\Pi$.

Base case: Let $n = 0$. Then, $\Pi$ is satisfied by any sample $\tilde{\mathbf{x}}$, and $\mathrm{DRL}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}$.

Inductive step: Assume $n > 0$ and let $x_i$ be the last variable in the given ordering that occurs in $\Pi$. Clearly, $\Pi_{i-1}$ contains $(n-1)$ variables. Thus, for the inductive hypothesis, assume that $\mathrm{DRL}(\tilde{\mathbf{x}})$ satisfies $\Pi_{i-1}$. Then, it remains to show that $\mathrm{DRL}(\tilde{\mathbf{x}})$ also satisfies $\Pi_i$. From the inductive hypothesis and from Lemma 5.3.7, we have that $\widetilde{\Pi}_i$ is satisfiable. Hence, the result of the Theorem follows from Lemma 5.3.1. □

Further, given the variable ordering $x_1, x_2, \ldots, x_D$, $\mathrm{DRL}(\tilde{\mathbf{x}})$ is *optimal w.r.t.* $\tilde{\mathbf{x}}$, $\Pi$ *and the variable ordering*: for each $i = 1, \ldots, D$ there does not exist a sample $\tilde{\mathbf{x}}' \in \Omega(\Pi)$ such that (i) $|\tilde{x}_i - \tilde{x}'_i| < |\tilde{x}_i - \mathrm{DRL}(\tilde{\mathbf{x}})_i|$, and (ii) $\tilde{x}'_j = \mathrm{DRL}(\tilde{\mathbf{x}})_j$ for all $j < i$.

**Theorem 5.3.11.** *Let* $\Pi$ *be a finite and satisfiable set of constraints. For any sample* $\tilde{\mathbf{x}}$ *and variable ordering, the corresponding sample DRL*$(\tilde{\mathbf{x}})$ *is optimal w.r.t.* $\tilde{\mathbf{x}}$, $\Pi$ *and the variable ordering.*

*Proof.* The statement can be proved by induction over the number $n$ of variables occurring in $\Pi$.

Base case: Let $n = 0$. Then, $\Pi$ is satisfied by any sample $\tilde{\mathbf{x}}$, and $\text{DRL}(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}}$. Thus, the result of the Theorem trivially follows.

Inductive step: Assume $n > 0$ and let $x_i$ be the last variable in the given ordering that occurs in $\Pi$. Clearly, $\Pi_{i-1}$ contains only the variables $x_1, x_2, \ldots x_{i-1}$. Thus, for the inductive hypothesis, assume that $\text{DRL}(\tilde{\mathbf{x}})$ is optimal with respect to $\tilde{\mathbf{x}}$, $\Pi_{i-1}$ and the variable ordering, for any sample $\tilde{\mathbf{x}}$. Then, it remains to show that $\text{DRL}(\tilde{\mathbf{x}})$ is also optimal with respect to $\widetilde{\Pi}_i$.

From the inductive hypothesis and from Lemma 5.3.7, we have that $\widetilde{\Pi}_i$ is satisfiable. Furthermore, from Lemma 5.3.1 we have that for every $\tilde{\mathbf{x}}$, $\text{DRL}(\tilde{\mathbf{x}})$ is optimal w.r.t. $\tilde{\mathbf{x}}$ and $\widetilde{\Pi}_i$. Hence, the result of the Theorem follows. □

## 5.4 Experimental Analysis

To assess how DRL[4] performs in practice, I conduct the following studies. First, in Section 5.4.2, I investigate whether the DRL layer improves the quality of the synthetic data generated by standard DGMs. In Section 5.4.3, I then compare the constrained models (referred to as DGMs+DRL) with the models obtained by considering only the linear constraints in each dataset and adding the layer introduced in Chapter 3. More precisely, the latter models are constrained using only linear inequalities, rather than QFLRA constraints, and are obtained by adding a constraint layer on top of each DGM to create corresponding DGM + Linearly-constrained Layer (DGM+LL) models (i.e., WGAN+LL, TableGAN+LL, CTGAN+LL, TVAE+LL, and GOGGLE+LL). Then, in Section 5.4.4, I conduct experiments to determine how the background knowledge injection affects the sample generation time.

The results of the experimental analysis reveal that DRL is not only able to correct the samples such that they are compliant with the background knowledge, but it is also able to improve the machine learning efficacy of the models across the diverse range of dataset complexities examined in the experiments. Out of all datasets, on the smallest dataset (CCS) which contains only 835 training samples and 36 features,

[4]The code is available at https://github.com/mihaela-stoian/DRL_DGM.

the unconstrained DGMs yield constraint violation rates as high as 78.5% as well as low machine learning efficacy. DRL not only eliminates these violations, but also provides consistent improvements in all three metrics across four out of five models. Moreover, these improvements are often non-negligible (with the largest ones being 21.4%, 20.5%, and 20.9% on GOGGLE, in terms of F1-, weighted F1-score, and AUC, respectively). On the other hand, the largest dataset (LCLD) which contains more than 490K training samples, demonstrates the scalability of DRL. Indeed, despite the very large number of training samples, DRL consistently improves nearly all metrics (improving 13 out of 15 cases). Finally, on the regression dataset (House), where every baseline model results in 100% constraint violation rate, DRL successfully corrects this to 0%, while improving all metrics (i.e., MAE and RMSE) for every model.

Before delving into these studies, I describe the metrics used to compute the sample quality, along with the models and datasets.

### 5.4.1 Experimental Setup

**Models.** Below I give a brief description of each of the five models used in the experimental analysis. For each of the following DGMs, I build the DRL on top of the DGM to create corresponding DGM+DRL models.

- WGAN (Arjovsky et al., 2017): is a GAN-based model which has been trained by using the Wasserstein distance as a loss, which improves the stability of learning.

- TableGAN (Park et al., 2018): is a GAN-based model designed for generating realistic tabular data. It uses a convolutional neural network (CNN) as a discriminator to better the capture dependencies among features.

- CTGAN (Xu et al., 2019): is again a GAN-based model which uses a conditional generator to model feature distributions and applies a mode-specific normalisation technique to improve the generation of imbalanced categorical data.

- TVAE (Xu et al., 2019): uses a variational autoencoder architecture to capture both continuous and categorical feature distributions, learning a probabilistic latent space representation of the data. By optimising the evidence lower bound, it balances reconstruction accuracy and regularisation.

- GOGGLE (Liu et al., 2022): uses a VAE framework combined with a graph neural network, which allows the model to learn complex feature relationships

Table 5.1: Dataset statistics. For each dataset, the following are provided: (i) the number of samples in the training partition, (ii) the number of samples in the validation partition, (iii) the number of samples in the test partition, (iv) the number of features, (v) the number of constraints, and (vi) the type of downstream task.

| Dataset | # Train | # Val | # Test | # Features | # Constraints | Task (# classes) |
|---|---|---|---|---|---|---|
| URL | 7K | 2K | 2K | 64 | 18 | Binary classification |
| CCS | 1K | 0.02K | 0.15K | 36 | 12 | Binary classification |
| LCLD | 494K | 199K | 431K | 29 | 16 | Binary classification |
| Heloc | 8K | 2K | 0.2K | 24 | 12 | Binary classification |
| House | 17K | 0.5K | 4K | 20 | 13 | Regression |

by representing the data as a graph, where each node corresponds to a feature, and edges capture dependencies between features.

**Datasets.** For the experimental analysis, I consider five real-world datasets and associated constraints. Four datasets (i.e., URL, CCS, LCLD, and Heloc) are used for classification tasks, while one dataset (i.e., House) is used for regression.

Below I provide a brief description for each dataset and the links to the pages where they can be downloaded:

- URL[5] (Hannousse and Yahiouche, 2021) is used to perform webpage phishing detection with features describing statistical properties of the URL itself as well as the content of the page.
- CCS[6] is used to identify individuals at high risk of cervical cancer from features describing the patients' demographic and medical history.
- LCLD[7] is used to predict whether the debt lent is unlikely to be collected from features related to the loan as well as client history. In particular, I use the feature-engineered dataset from (Simonetto et al., 2022), inspired from the LendingClub loan data.
- HELOC[8] is a dataset from FICO used to predict whether customers will repay their credit lines within 2 years from features related to the credit line and the client's history.

[5]Link to dataset: https://data.mendeley.com/datasets/c2gw7fy2j4/2
[6]Link to dataset:https://www.kaggle.com/datasets/ranzeet013/cervical-cancer-dataset/data
[7]Link to dataset: https://figshare.com/s/84ae808ce6999fafd192
[8]Link to dataset: https://huggingface.co/datasets/mstz/heloc

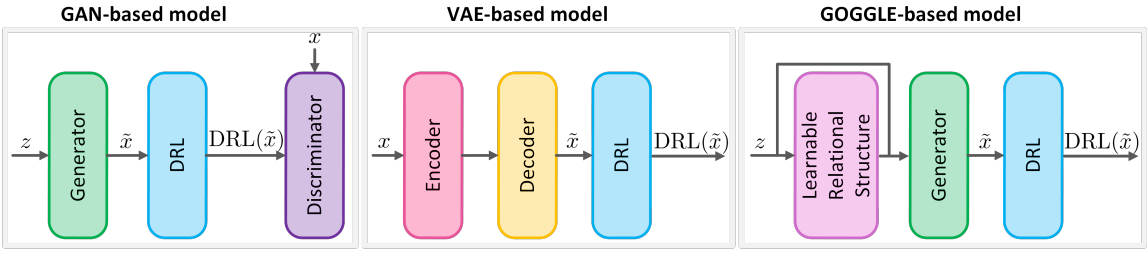



Figure 5.5: Visualisation of the considered types of DGMs and how to add DRL in their topology.

- House[9] is used to predict the prices of houses in King County (USA) and contains data collected from May 2014 to May 2015. The features describe various features of the sold houses, including the date of sale, house prices, the number of bedrooms and bathrooms etc.

For each dataset above, Table 5.1 shows the number of samples in the train, validation and test partitions, along with the number of features and the number of constraints. In all cases, the real data always satisfy the constraints. In particular, the constraints were already included in the original URL, Heloc and LCLD datasets. Regarding the CCS and House datasets, I manually annotated the constraints using background knowledge about the problem, then checked whether the data were compliant with the constraints and finally retained only the constraints that were satisfied by all the datapoints. For examples of constraints, see Table H.3 of Appendix H.2.

**Sample Quality Evaluation.** To judge the quality of the synthetic samples I measure (i) how well they align with the background knowledge and (ii) how well they can replace the real data in downstream tasks. In particular, to measure *background knowledge alignment* of the synthetic data in (i), I consider the metrics proposed in (Stoian et al., 2024a): i.e, *constraint violation rate* (CVR), *constraint violation coverage* (CVC), and *samplewise constraints violation coverage* (sCVC), as described in Chapter 3. To determine the usability of the synthetic data in downstream tasks for (ii), I evaluate the *machine learning efficacy* (Kim et al., 2023), also known as utility (e.g., in (Liu et al., 2022)) or efficiency. To compute it, I follow the "Train on Synthetic, Test on Real" protocol (Esteban et al., 2017), along with the steps from (Kim et al., 2023). For clarity, I describe the protocol below.

First, I generate a synthetic dataset and split it into training, validation, and testing sets, maintaining the same proportions as in the real dataset. Next, I conduct a

[9]Link to dataset: https://www.kaggle.com/datasets/harlfoxem/housesalesprediction/data

hyperparameter search using the synthetic training set to train various classifiers and regressors. Specifically, for the classification datasets (i.e., URL, CCS, LCLD, Heloc), I use the following six classifiers: AdaBoost (Schapire, 2013), Decision Tree (Wu et al., 2008), Logistic Regression (Cox, 1958) Multi-layer Perceptron (MLP) (Haykin, 1994), Random Forest (Ho, 1995), and XGBoost (Chen and Guestrin, 2016). For the regression dataset (i.e., House), I use: Linear Regression, MLP, Random Forest regressors, and XGBoost. Across all classifiers and regressors, I use the same hyperparameter settings as those in Table 26 of (Kim et al., 2023). Then, based on the F1-score obtained on the real validation set, I select the best hyperparameter configuration. As a last step, I evaluate the selected models on the real test set and average the performance across all classifiers/regressors.

For classification datasets, I report: F1-score (F1), weighted F1-score ($w$F1), and Area Under the ROC Curve (AUC), while for the regression dataset, I compute the mean absolute error (MAE) and the root mean square error (RMSE). I run the entire process described above five times for each model, then compute the average results for each metric individually across the repetitions.

**Hyperparameter search.** In order to provide strong baselines for the DGMs, I conducted a hyperparameter search for each model and each dataset. Notice that the hyperparameter configurations selected for a specific DGM have been used for all associated DGMs equipped with the LL and the DRL layers (i.e., the associated DGM+LL and DGM+DRL models). To ensure the reproducibility of the results, I included all the necessary details for the hyperparameter search in Appendix C, along with the best hyperparameter configurations in Table C.1 of Appendix C.

**DGM+DRL schemas.** In Figure 5.5 I give an overview on how to add DRL in the topology of the three types of DGM architectures considered in the experimental analysis. In all figures I indicate with $\mathbf{z}$ a noise vector, with $\mathbf{x}$ a real datapoint from the original dataset, with $\tilde{\mathbf{x}}$ a sample generated by the DGM, and with DRL($\tilde{\mathbf{x}}$) the final sample obtained from DRL. In particular, we can see that:

- The leftmost panel in the Figure shows that DRL needs to be added on top of the generator module in GAN-based models,
- The middle panel shows that DRL needs to be added after the decoder module in VAE-based models, and
- The rightmost panel shows that DRL needs to be added after the generator module in GOGGLE-like models.

Table 5.2: CVR for each model and dataset. Cases with CVR≥50% are underlined. Best results are in bold.

| | URL | CCS | LCLD | Heloc | House |
|---|---|---|---|---|---|
| WGAN | 22.8±4.9 | 44.7±7.1 | 47.5±14.5 | 80.6±9.3 | 100.0±0.0 |
| TableGAN | 8.5±2.2 | 61.2±13.3 | 32.0±4.7 | 59.9±16.7 | 100.0±0.0 |
| CTGAN | 9.7±2.0 | 78.5±5.7 | 7.1±1.3 | 56.6±9.8 | 100.0±0.0 |
| TVAE | 10.3±1.1 | 16.9±1.6 | 10.3±0.6 | 44.9±1.0 | 100.0±0.0 |
| GOGGLE | 7.3±8.1 | 60.3±6.8 | 70.4±16.1 | 52.7±6.3 | 100.0±0.0 |
| All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 |

Table 5.3: Constraint violation rate (CVR) for each unconstrained DGM model and each dataset.

| Constraint Type | Model/Dataset | URL | CCS | LCLD | Heloc | House |
|---|---|---|---|---|---|---|
| Linear | WGAN | 17.9±5.0 | 15.0±5.6 | 28.5±16.3 | 69.1±8.6 | 100.0±0.0 |
| | TableGAN | 5.4±1.4 | 14.5±3.6 | 19.1±3.7 | 45.6±16.3 | 100.0±0.0 |
| | CTGAN | 3.8±1.3 | 56.1±7.5 | 1.9±1.1 | 55.8±10.2 | 100.0±0.0 |
| | TVAE | 3.0±0.7 | 8.6±1.9 | 3.9±0.5 | 44.8±1.0 | 100.0±0.0 |
| | GOGGLE | 47.3±6.9 | 42.5±3.9 | 16.5±13.2 | 47.3±6.9 | 99.9±0.1 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |
| Disjunctive | WGAN | 8.6±1.7 | 38.3±10.0 | 26.8±5.2 | 47.3±15.5 | 74.2±4.4 |
| | TableGAN | 3.9±1.4 | 56.1±14.4 | 16.0±3.5 | 33.8±14.7 | 73.7±14.8 |
| | CTGAN | 6.3±1.0 | 58.8±6.4 | 5.3±0.8 | 1.7±1.4 | 42.8±23.1 |
| | TVAE | 7.6±0.7 | 11.8±0.7 | 6.6±0.5 | 0.0±0.0 | 52.7±24.7 |
| | GOGGLE | 2.0±2.8 | 37.1±15.8 | 65.3±15.8 | 35.2±4.2 | 20.4±20.5 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |

In general, DRL can be added in many different DGMs, and it simply needs to be added right after the sample $\tilde{\mathbf{x}}$ is generated.

### 5.4.2 Synthetic Data Quality

**Background knowledge alignment.** To assess how often the samples violate the constraints, I calculate the CVR, which is defined as the percentage of samples that violate at least one constraint. Table 5.2 shows the CVR for each unconstrained model (first five rows) and the models equipped with the DRL (last row). As expected, the models with DRL always satisfy the constraints, while the samples obtained with standard DGMs very often violate them. Additionally, in many cases, the CVR is extremely high: out of 25 cases, 13 cases have CVR greater than 50% and 5 cases have CVR equal to 100%, thus making the standard procedure of rejecting non-aligned

Table 5.4: Samplewise constraints violation coverage (sCVC) for each unconstrained DGM model and each dataset.

| Constraint Type | Model/Dataset | URL | CCS | LCLD | Heloc | House |
|---|---|---|---|---|---|---|
| Linear | WGAN | 2.2±0.6 | 7.9±2.9 | 15.1±9.4 | 14.3±2.5 | 50.0±0.0 |
| | TableGAN | 0.7±0.2 | 7.5±1.9 | 9.8±1.9 | 9.0±3.3 | 50.0±0.0 |
| | CTGAN | 0.5±0.2 | 32.1±5.6 | 1.0±0.5 | 9.9±2.8 | 50.0±0.0 |
| | TVAE | 0.4±0.1 | 4.4±1.0 | 2.0±0.3 | 7.4±0.2 | 50.0±0.0 |
| | GOGGLE | 17.2±4.9 | 22.0±2.3 | 8.6±7.1 | 17.2±4.9 | 50.0±0.0 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |
| Disjunctive | WGAN | 0.9±0.2 | 5.3±1.7 | 2.2±0.5 | 11.5±4.4 | 7.0±0.4 |
| | TableGAN | 0.4±0.1 | 6.8±1.9 | 1.5±0.3 | 7.9±3.7 | 6.7±1.3 |
| | CTGAN | 0.6±0.1 | 8.0±1.2 | 0.4±0.1 | 0.3±0.3 | 3.9±2.1 |
| | TVAE | 0.8±0.1 | 1.3±0.1 | 0.5±0.0 | 0.0±0.0 | 4.8±2.3 |
| | GOGGLE | 0.2±0.3 | 5.0±2.1 | 10.0±3.4 | 7.1±0.9 | 1.9±1.9 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |

Table 5.5: Constraints violation coverage (CVC) for each unconstrained DGM model and each dataset.

| Constraint Type | Model/Dataset | URL | CCS | LCLD | Heloc | House |
|---|---|---|---|---|---|---|
| Linear | WGAN | 45.0±5.0 | 100.0±0.0 | 100.0±0.0 | 100.0±0.0 | 100.0±0.0 |
| | TableGAN | 34.0±4.2 | 100.0±0.0 | 100.0±0.0 | 100.0±0.0 | 100.0±0.0 |
| | CTGAN | 17.5±4.3 | 100.0±0.0 | 100.0±0.0 | 100.0±0.0 | 100.0±0.0 |
| | TVAE | 12.5±0.0 | 100.0±0.0 | 100.0±0.0 | 99.4±1.3 | 100.0±0.0 |
| | GOGGLE | 100.0±0.0 | 100.0±0.0 | 100.0±0.0 | 100.0±0.0 | 100.0±0.0 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |
| Disjunctive | WGAN | 50.0±0.0 | 40.0±0.0 | 28.0±1.3 | 80.0±0.0 | 34.9±2.4 |
| | TableGAN | 53.6±3.8 | 40.0±0.0 | 51.3±3.9 | 80.0±0.0 | 36.4±0.0 |
| | CTGAN | 56.4±5.0 | 40.0±0.0 | 22.3±2.2 | 55.2±6.6 | 34.9±3.3 |
| | TVAE | 55.2±4.6 | 40.0±0.0 | 20.3±3.4 | 24.8±11.1 | 36.0±0.8 |
| | GOGGLE | 28.4±21.3 | 40.0±0.0 | 49.1±8.1 | 48.8±15.6 | 33.3±5.2 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |

samples unfeasible.

In addition to the CVR metric, I also compute the samplewise constraints violation coverage (sCVC) and the constraint violation coverage (CVC). Recall that sCVC indicates the average percentage of constraints violated per sample, across all samples, and CVC represents the percentage of constraints violated at least once by the samples. Moreover, as constraints expressed as linear inequalities are a special case of the QFLRA constraints, I have also partitioned the constraints between those that can be captured as linear inequalities and those that cannot. In Tables 5.3, 5.4, and 5.5, I provide a breakdown in terms of the CVR, sCVC, and CVC, respectively,

Table 5.6: Machine learning efficacy comparison between the unconstrained DGMs, and their +DRL and +RS counterparts. The performance is measured using F1, *w*F1, and AUC, for each classification dataset.

| | F1 | | | | wF1 | | | | AUC | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | URL | CCS | LCLD | Heloc | URL | CCS | LCLD | Heloc | URL | CCS | LCLD | Heloc |
| WGAN | 0.794 | 0.303 | 0.139 | 0.665 | 0.796 | 0.330 | 0.296 | 0.648 | 0.870 | 0.814 | 0.605 | **0.717** |
| + RS | 0.792 | 0.051 | 0.156 | 0.628 | 0.794 | 0.088 | 0.312 | 0.617 | 0.862 | 0.570 | 0.611 | 0.685 |
| + DRL | **0.800** | **0.313** | **0.197** | **0.721** | **0.801** | **0.340** | **0.339** | **0.652** | **0.875** | **0.885** | **0.623** | **0.717** |
| TableGAN | 0.562 | **0.196** | 0.259 | 0.593 | 0.659 | **0.228** | 0.393 | 0.615 | 0.843 | **0.802** | 0.655 | 0.707 |
| + RS | 0.544 | 0.138 | 0.251 | 0.568 | 0.648 | 0.172 | 0.389 | 0.599 | 0.854 | 0.682 | 0.653 | 0.685 |
| + DRL | **0.619** | 0.163 | **0.269** | **0.628** | **0.693** | 0.196 | **0.401** | **0.628** | **0.865** | 0.742 | **0.657** | **0.709** |
| CTGAN | 0.822 | 0.145 | 0.247 | 0.736 | 0.799 | 0.159 | 0.379 | 0.675 | 0.859 | 0.914 | **0.651** | 0.744 |
| + RS | 0.817 | 0.086 | 0.201 | 0.706 | 0.795 | 0.095 | 0.342 | 0.650 | 0.856 | 0.515 | 0.615 | 0.706 |
| + DRL | **0.836** | **0.288** | **0.288** | **0.744** | **0.815** | **0.308** | **0.409** | **0.680** | **0.883** | **0.955** | 0.643 | **0.745** |
| TVAE | 0.810 | 0.325 | 0.185 | 0.717 | 0.802 | 0.351 | **0.330** | 0.686 | 0.863 | 0.858 | 0.631 | 0.750 |
| + RS | 0.788 | 0.024 | **0.237** | 0.420 | 0.778 | 0.061 | 0.283 | 0.465 | 0.846 | 0.522 | 0.480 | 0.497 |
| + DRL | **0.835** | **0.467** | 0.189 | **0.731** | **0.832** | **0.487** | **0.330** | **0.694** | **0.893** | **0.926** | **0.635** | **0.752** |
| GOGGLE | 0.622 | 0.039 | 0.248 | 0.596 | 0.648 | 0.076 | 0.296 | 0.566 | 0.742 | 0.549 | 0.551 | 0.600 |
| + RS | 0.608 | 0.047 | 0.235 | 0.577 | 0.639 | 0.084 | **0.322** | 0.549 | 0.727 | 0.571 | 0.532 | 0.592 |
| + DRL | **0.720** | **0.253** | **0.298** | **0.698** | **0.673** | **0.281** | 0.310 | **0.636** | **0.747** | **0.758** | **0.563** | **0.691** |

for the unconstrained DGM models, according to the two possible types: linear constraints and constraints that have at least two linear inequalities in disjunction. This breakdown allows for checking how much each of the two partitions contributes to the high CVR results. To this end, I found that in 13 (resp. 6) out of 25 cases, the CVR was greater than 25% (resp. 50%) on the constraints presenting disjunctions alone.

**Machine Learning Efficacy.** Table 5.6 shows that: (i) making the samples compliant with the constraints via rejection sampling (RS) often reduces their machine learning efficacy (indicated as DGMs+RS), and that (ii) adding DRL improves the performance of the unconstrained models according to at least one metric in all cases but one (TableGAN over the CCS dataset). Indeed, we can see that rejection sampling decreases the performance with respect to the standard DGMs in 17, 17 and 17 out of 20 cases for F1, *w*F1 and AUC, respectively.

On the other hand, DRL improves the performance w.r.t. the unconstrained models in 19, 18 and 17 out of 20 cases for F1, *w*F1 and AUC, respectively. Additionally, the improvements are often non-negligible. For F1, in more than half of the cases, the improvement is of at least 3.5%, with the largest one recorded on GOGGLE for CCS of 21.4%. For *w*F1, in more than half of the cases, the improvement is of at least

Table 5.7: Machine learning efficacy comparison between the unconstrained DGM models and their DRL counterparts in terms of F1. The performance, along with the standard deviation, is reported for each classification dataset.

| | URL | CCS | LCLD | Heloc |
|---|---|---|---|---|
| WGAN | 0.794±0.041 | 0.303±0.060 | 0.139±0.053 | 0.665±0.050 |
| WGAN+RS | 0.792±0.031 | 0.051±0.037 | 0.156±0.074 | 0.628±0.043 |
| WGAN+DRL | **0.800**±0.011 | **0.313**±0.127 | **0.197**±0.060 | **0.721**±0.027 |
| TableGAN | 0.562±0.051 | **0.196**±0.037 | 0.259±0.011 | 0.593±0.058 |
| TableGAN+RS | 0.544±0.071 | 0.138±0.025 | 0.251±0.020 | 0.568±0.077 |
| TableGAN+DRL | **0.619**±0.046 | 0.163±0.079 | **0.269**±0.025 | **0.628**±0.083 |
| CTGAN | 0.822±0.017 | 0.145±0.040 | 0.247±0.087 | 0.736±0.035 |
| CTGAN+RS | 0.817±0.008 | 0.086±0.016 | 0.201±0.066 | 0.706±0.014 |
| CTGAN+DRL | **0.836**±0.004 | **0.288**±0.116 | **0.288**±0.013 | **0.744**±0.020 |
| TVAE | 0.810±0.008 | 0.325±0.190 | 0.185±0.021 | 0.717±0.013 |
| TVAE+RS | 0.788±0.023 | 0.024±0.011 | **0.237**±0.018 | 0.420±0.007 |
| TVAE+DRL | **0.835**±0.009 | **0.467**±0.100 | 0.189±0.022 | **0.731**±0.009 |
| GOGGLE | 0.622±0.094 | 0.039±0.016 | 0.248±0.156 | 0.596±0.072 |
| GOGGLE+RS | 0.608±0.098 | 0.047±0.024 | 0.235±0.149 | 0.577±0.093 |
| GOGGLE+DRL | **0.720**±0.086 | **0.253**±0.144 | **0.298**±0.153 | **0.698**±0.023 |

Table 5.8: Machine learning efficacy comparison between the unconstrained DGM models and their DRL counterparts in terms of $w$F1. The performance, along with the standard deviation, is reported for each classification dataset.

| | URL | CCS | LCLD | Heloc |
|---|---|---|---|---|
| WGAN | 0.796±0.026 | 0.330±0.057 | 0.296±0.037 | 0.648±0.027 |
| WGAN+RS | 0.794±0.020 | 0.088±0.035 | 0.312±0.056 | 0.617±0.018 |
| WGAN+DRL | **0.801**±0.014 | **0.340**±0.122 | **0.339**±0.049 | **0.652**±0.036 |
| TableGAN | 0.659±0.035 | **0.228**±0.035 | 0.393±0.010 | 0.615±0.030 |
| TableGAN+RS | 0.648±0.046 | 0.172±0.024 | 0.389±0.015 | 0.599±0.036 |
| TableGAN+DRL | **0.693**±0.028 | 0.196±0.076 | **0.401**±0.018 | **0.628**±0.038 |
| CTGAN | 0.799±0.033 | 0.159±0.042 | 0.379±0.061 | 0.675±0.015 |
| CTGAN+RS | 0.795±0.014 | 0.095±0.019 | 0.342±0.054 | 0.650±0.009 |
| CTGAN+DRL | **0.815**±0.011 | **0.308**±0.118 | **0.409**±0.007 | **0.680**±0.011 |
| TVAE | 0.802±0.012 | 0.351±0.182 | **0.330**±0.016 | 0.686±0.004 |
| TVAE+RS | 0.778±0.026 | 0.061±0.010 | 0.283±0.007 | 0.465±0.001 |
| TVAE+DRL | **0.832**±0.014 | **0.487**±0.096 | **0.330**±0.014 | **0.694**±0.006 |
| GOGGLE | 0.648±0.074 | 0.076±0.015 | 0.296±0.066 | 0.566±0.050 |
| GOGGLE+RS | 0.639±0.068 | 0.084±0.023 | **0.322**±0.065 | 0.549±0.051 |
| GOGGLE+DRL | **0.673**±0.039 | **0.281**±0.139 | 0.310±0.057 | **0.636**±0.020 |

Table 5.9: Machine learning efficacy comparison between the unconstrained DGM models and their DRL counterparts in terms of AUC. The performance, along with the standard deviation, is reported for each classification dataset.

| | URL | CCS | LCLD | Heloc |
|---|---|---|---|---|
| WGAN | 0.870±0.012 | 0.814±0.072 | 0.605±0.010 | **0.717**±0.021 |
| WGAN+RS | 0.862±0.019 | 0.570±0.070 | 0.611±0.022 | 0.685±0.023 |
| WGAN+DRL | **0.875**±0.007 | **0.885**±0.050 | **0.623**±0.023 | **0.717**±0.029 |
| TableGAN | 0.843±0.020 | **0.802**±0.044 | 0.655±0.011 | 0.707±0.007 |
| TableGAN+RS | 0.854±0.016 | 0.682±0.086 | 0.653±0.010 | 0.685±0.020 |
| TableGAN+DRL | **0.865**±0.022 | 0.742±0.096 | **0.657**±0.007 | **0.709**±0.011 |
| CTGAN | 0.859±0.040 | 0.914±0.039 | **0.651**±0.020 | 0.744±0.009 |
| CTGAN+RS | 0.856±0.010 | 0.515±0.083 | 0.615±0.031 | 0.706±0.014 |
| CTGAN+DRL | **0.883**±0.009 | **0.955**±0.022 | 0.643±0.019 | **0.745**±0.008 |
| TVAE | 0.863±0.011 | 0.858±0.100 | 0.631±0.004 | 0.750±0.004 |
| TVAE+RS | 0.846±0.024 | 0.522±0.040 | 0.480±0.008 | 0.497±0.006 |
| TVAE+DRL | **0.893**±0.010 | **0.926**±0.039 | **0.635**±0.002 | **0.752**±0.003 |
| GOGGLE | 0.742±0.071 | 0.549±0.051 | 0.551±0.034 | 0.600±0.056 |
| GOGGLE+RS | 0.727±0.060 | 0.571±0.077 | 0.532±0.049 | 0.592±0.052 |
| GOGGLE+DRL | **0.747**±0.029 | **0.758**±0.091 | **0.563**±0.027 | **0.691**±0.039 |

1.0%, with the largest improvement, of 20.5%, again recorded on GOGGLE for CCS. And, for AUC, the improvement is at least 1.2% in half of the cases, with the largest improvement, of 20.9%, recorded on GOGGLE for CCS.

For detailed results, Tables 5.7, 5.8, 5.9 show the efficacy, along with the standard deviations from the mean, for each unconstrained DGM model and the corresponding DGM+DRL models on every classification dataset, using the F1, $w$F1, and AUC metrics, respectively.

Further, in Table 5.10, I report MAE and RMSE for each unconstrained DGM model and the corresponding DGM+DRL models on the regression dataset (i.e., House). We find a similar trend in terms of improvements brought by DRL, where the DGM+DRL models improve the performance w.r.t. the unconstrained models in all cases.

It is worth noting that the CCS dataset, which sees the highest improvements overall when using DRL, is the smallest out of all datasets, with only 835 samples and 36 features. For a comparison of the datasets, see Table 5.1.

I also verify the statistical significance of the results following the recommendation of (Demsar, 2006). To this end, I perform the Wilcoxon signed-rank test on the efficacy results for the classification datasets, obtaining p-value $< 0.001$ w.r.t. the F1 and $w$F1

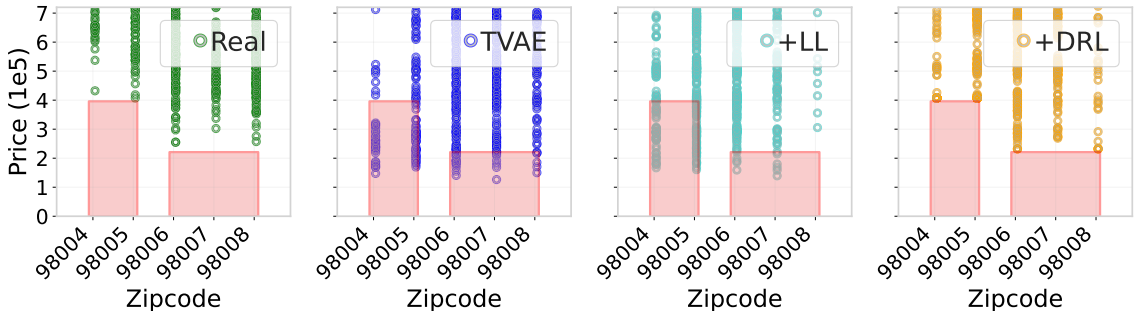



Figure 5.6: Sample distributions w.r.t. the *Zipcode* and *Price* features from the House dataset, for real data (first panel) and synthetic data from TVAE (second panel), TVAE+LL (third panel) and TVAE+DRL (last panel). The regions where samples violate the constraints are highlighted in red.

Table 5.10: Machine learning efficacy performance comparison between DGM and DGM+DRL models trained on House, using MAE and RMSE.

| | MAE | RMSE |
|---|---|---|
| WGAN | 547652.6±6.1 | 688130.1±4.8 |
| WGAN+DRL | **547637.5**±17.4 | **688118.0**±14.9 |
| TableGAN | 547655.3±5.9 | 688132.7±4.9 |
| TableGAN+DRL | **547653.6**±22.8 | **688131.4**±18.7 |
| CTGAN | 547652.9±2.7 | 688130.4±2.0 |
| CTGAN+DRL | **547642.9**±18.1 | **688122.1**±14.0 |
| TVAE | 547650.0±5.1 | 688128.5±4.2 |
| TVAE+DRL | **547645.4**±31.8 | **688124.6**±27.8 |
| GOGGLE | 547639.5±13.8 | 688119.6±11.5 |
| GOGGLE+DRL | **547633.9**±16.0 | **688115.7**±11.3 |

results and $< 0.005$ w.r.t. AUC. Importantly, the test confirms that DRL significantly improves the performances of DGMs.

**Comparison with real data performance.** For reference, I report the efficacy performance when training on the real data in Table F.2 of Appendix F.2. From the Table, we can see that the synthetic data generated with all the DGMs (unconstrained, and +DRL) manage to obtain very good results. On multiple occasions, the models trained on the synthetic data not only get results comparable with the ones obtained with the real data, but actually perform better than them. In spite of this, the classification models trained on synthetic data never manage to get better performance than those trained on the real data for all metrics. On the other hand,

the regressors trained on the synthetic version of the House dataset always manage to get lower MAE and RMSE, no matter the DGM used to generate the synthetic data.

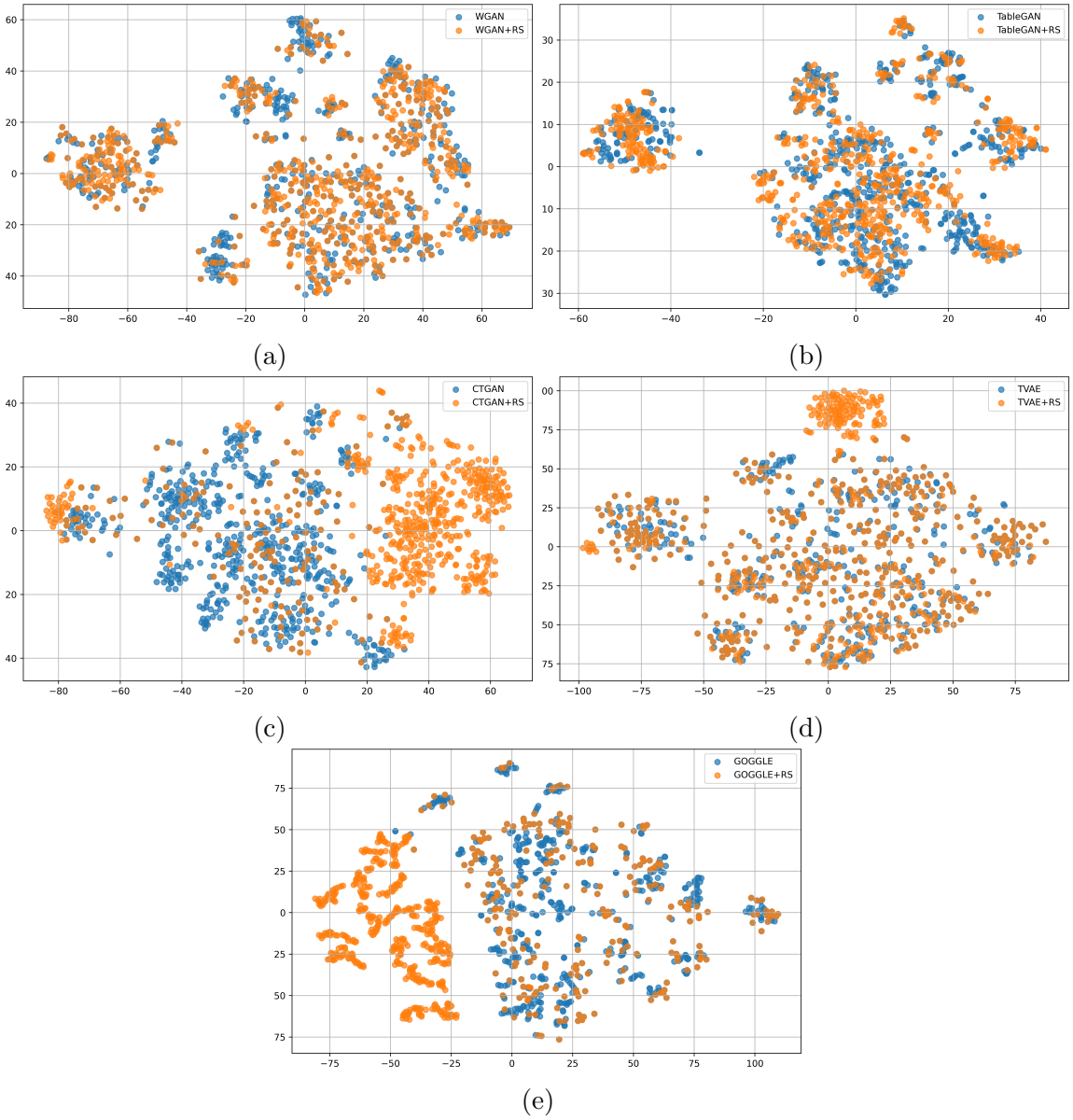



Figure 5.7: t-SNE visualisations of the distribution of the samples generated for the CCS dataset by (i) WGAN and WGAN+RS in Figure 5.7a, (ii) TableGAN and TableGAN+RS in Figure 5.7b, (iii) CTGAN and CTGAN+RS in Figure 5.7c, (iv) TVAE and TVAE+RS in Figure 5.7d, and (v) GOGGLE and GOGGLE+RS in Figure 5.7e.

**Training dynamics.** To evaluate training dynamics and convergence speed, I show the auxiliary classifier loss of the TableGAN generator. As noted by Park et al. (2018),

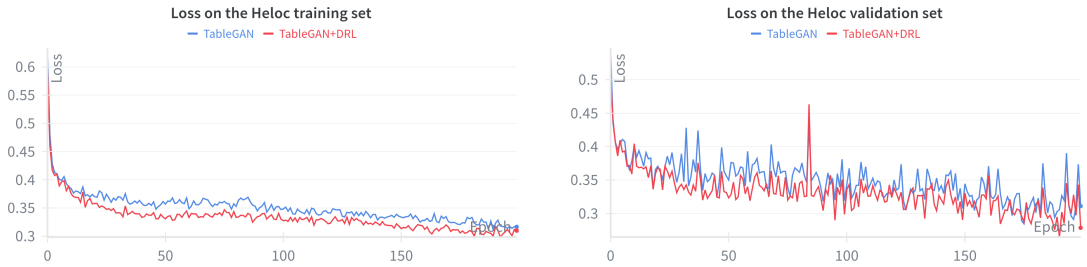



Figure 5.8: Training and validation loss curves (measuring the auxiliary classifier loss of the generator) for the Heloc dataset. The plots compare the convergence of the unconstrained TableGAN baseline against the constrained TableGAN+DRL model over the same number of training epochs.

this loss directly evaluates the discrepancy between the label (assigned to the target feature of the dataset) of a generated sample and the label predicted by TableGAN's classifier for that sample. By monitoring this loss (in Figure 5.8), we can observe the training dynamics and empirically validate the gradient flow through the DRL layer. To this end, the learning curves show that DRL is fully differentiable and does not impede the gradients. In fact, it provides the generator with informative gradient signals, effectively guiding the model towards valid data distributions (according to the classifier loss) faster than the unconstrained baseline. Further, comparing the plots reveals that the validation loss closely tracks the training loss, indicating that the DRL layer leads to improved convergence without causing the model to overfit. Finally, regarding computational training time, the integration of the DRL layer introduces an expected overhead during the training process, similar to the one reported for LL (in Chapter 3), with the constrained model from the Figure taking 2.9× the time to train the unconstrained model for the same number of epochs. However, as a future experiment, the faster convergence speed of the DRL could be leveraged to implement early stopping, potentially allowing the constrained models to achieve optimal performance in fewer epochs than the unconstrained models.

**Qualitative analysis.** Further, to visualise the impact of DRL, I consider the features *Price* and *Zipcode* from the House dataset and create the scatter plots of (i) the real data, and the synthetic data from (ii) the unconstrained DGMs, (iii) the DGMs+LL, and (iv) the DGMs+DRL. The constraints considered are the following: (i) *if the Zipcode is 98004 or 98005 then the Price is greater than 400K USD* and (ii) *if the Zipcode is between 98006 and 98008 then the Price exceeds 225K USD*. In each plot, the regions that violate the constraints are highlighted in red.

Table 5.11: CVR for each DGM+LL model and dataset. Cases with CVR≥50% are underlined. Best results are in bold.

| | URL | CCS | LCLD | Heloc | House |
|---|---|---|---|---|---|
| WGAN+LL | 8.9±3.2 | 51.5±11.2 | 27.0±3.6 | 20.6±6.3 | 100.0±0.0 |
| TableGAN+LL | 3.6±0.8 | 54.0±17.8 | 11.3±0.9 | 26.6±7.7 | 23.9±2.7 |
| CTGAN+LL | 7.0±2.6 | 55.7±16.3 | 2.6±1.1 | 2.6±2.4 | 10.8±7.8 |
| TVAE+LL | 6.8±0.6 | 8.4±2.0 | 5.8±0.8 | 0.0±0.0 | 13.0±12.6 |
| GOGGLE+LL | 6.5±7.0 | 23.0±10.7 | 81.9±6.5 | 11.5±7.1 | 2.6±2.6 |
| All + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 |

The scatter plots obtained from the real datapoints and from the samples generated using TVAE, with and without LL and DRL, are shown in Figure 5.6. The plots obtained from the other models are given in Figure E.1, Appendix E. The Figures clearly show that standard DGMs and DGMs+LL fail to comply with the constraints, with many of the samples falling in the red-shaded regions. On the contrary, the samples obtained using DRL not only never violate the constraints, but also better match the real data distribution.

Finally, in order to show the unintended consequences of using rejection sampling, in Figure 5.7, I provide t-SNE (van der Maaten and Hinton, 2008) visualisations of the differences between the distributions of the samples generated by the standard DGMs and by DGMs+RS (i.e, with rejection sampling). As we can see in the Figure, there can be cases where rejection sampling actually creates some changes in the distribution, which can then affect the machine learning efficacy.

### 5.4.3 Linear vs. QFLRA Constraints

**Background knowledge alignment.** Table 5.11 shows the CVR for each DGM+LL model (first five rows) and for the DGM+DRL models (last row). As expected, DGMs+LL cannot guarantee the compliance with QFLRA constraints and in 5 out of 25 cases we see a CVR greater than 50%. Moreover, there is one case (i.e., House for the WGAN+LL model) where the CVR is 100%, thus demonstrating the need for models that support more expressive constraints.

Overall, the linearly-constrained models do show a decrease in CVR, when compared to the unconstrained models. However, their results also demonstrate the need for having models that can support more expressive constraints, such as the QFLRA constraints, further motivating the DRL layer's applicability. This is detailed in Table D.1 of Appendix D, which breaks down the CVR results by the two constraint types: linear inequalities and disjunctions of linear inequalities.

Table 5.12: Machine learning efficacy comparison between the DGM+LL models and the models with DRL. The performance is measured using F1, *w*F1, and AUC, for each classification dataset.

| | F1 | | | | wF1 | | | | AUC | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | URL | CCS | LCLD | Heloc | URL | CCS | LCLD | Heloc | URL | CCS | LCLD | Heloc |
| WGAN+LL | **0.803** | **0.359** | 0.183 | 0.694 | 0.799 | **0.383** | 0.330 | **0.662** | 0.869 | 0.857 | 0.608 | **0.732** |
| WGAN+DRL | 0.800 | 0.313 | **0.197** | **0.721** | **0.801** | 0.340 | **0.339** | 0.652 | **0.875** | **0.885** | **0.623** | 0.717 |
| TableGAN+LL | 0.612 | **0.169** | 0.232 | **0.638** | **0.695** | **0.203** | 0.373 | **0.633** | **0.868** | **0.794** | 0.640 | 0.704 |
| TableGAN+DRL | **0.619** | 0.163 | **0.269** | 0.628 | 0.693 | 0.196 | **0.401** | 0.628 | 0.865 | 0.742 | **0.657** | **0.709** |
| CTGAN+LL | **0.836** | 0.250 | 0.265 | 0.729 | **0.820** | 0.271 | 0.392 | **0.688** | 0.880 | **0.959** | 0.641 | **0.755** |
| CTGAN+DRL | **0.836** | **0.288** | **0.288** | **0.744** | 0.815 | **0.308** | **0.409** | 0.680 | **0.883** | 0.955 | **0.643** | 0.745 |
| TVAE+LL | 0.824 | 0.413 | 0.158 | 0.730 | 0.816 | 0.436 | 0.310 | 0.691 | 0.878 | **0.933** | 0.633 | 0.747 |
| TVAE+DRL | **0.835** | **0.467** | **0.189** | **0.731** | **0.832** | **0.487** | **0.330** | **0.694** | **0.893** | 0.926 | **0.635** | **0.752** |
| GOGGLE+LL | **0.787** | 0.233 | 0.284 | **0.723** | **0.749** | 0.262 | **0.310** | **0.663** | **0.802** | **0.765** | 0.554 | **0.719** |
| GOGGLE+DRL | 0.720 | **0.253** | **0.298** | 0.698 | 0.673 | **0.281** | **0.310** | 0.636 | 0.747 | 0.758 | **0.563** | 0.691 |

In Appendix D, in Tables D.2 and D.3, I also report the results for the sCVC and CVC metrics for each DGM+LL model, again according to the the two constraint types.

**Machine Learning Efficacy.** For the sake of completeness, in Table 5.12 I include the comparison between DGMs+DRL and DGMs+LL on the classification datasets. Since in Chapter 3, I already reported improvements over the unconstrained DGMs by adding LL, as expected in this scenario, the improvements brought by the DGM+DRL models over the DGMs+LL are more modest than the ones registered above for the DGM+DRL w.r.t. the unconstrained models. As shown in the Table, the DGM+DRL models improve the efficacy w.r.t. the DGMs+LL for at least one metric in 17 out of 20 cases. Similarly, the number of times DGM+DRL outperforms the respective DGM+LL is lower than the number of times it outperforms its unconstrained counterpart. Indeed, out of 20 comparisons, the models with DRL outperform their linearly constrained counterparts 13, 10 and 11 times for F1, *w*F1, and AUC, respectively. Tables D.4, D.5, D.6 in Appendix D show the efficacy, along with the standard deviations from the mean, for each DGM+LL model and the corresponding DGM+DRL models on every classification dataset, using the F1, *w*F1, and AUC metrics, respectively.

Regarding the regression dataset, House, Table D.7 in Appendix D shows that the DGM+DRL models have a comparable performance to the DGM+LL models, with 6 out of 10 the cases showing an improvement in performance when using DRL.

Table 5.13: Sample generation time (in seconds) for all DGMs and their respective DGM+RS and DGM+DRL models, for all datasets. The last three rows show the average runtimes for the unconstrained DGMs, the DGMs+RS and the DGM+DRL models. The hyphen indicates timeout after 24h.

| | URL | CCS | LCLD | Heloc | House |
|---|---|---|---|---|---|
| WGAN | 0.01 | 0.01 | 0.01 | 0.01 | 0.00 |
| WGAN+RS | 0.08 | 0.11 | 0.12 | 0.25 | - |
| WGAN+DRL | 0.08 | 0.07 | 0.08 | 0.04 | 0.09 |
| TableGAN | 0.21 | 0.10 | 0.09 | 0.10 | 0.07 |
| TableGAN+RS | 0.43 | 1.57 | 0.38 | 0.78 | - |
| TableGAN+DRL | 0.28 | 0.14 | 0.19 | 0.15 | 0.18 |
| CTGAN | 0.16 | 0.11 | 0.10 | 0.09 | 0.07 |
| CTGAN+RS | 0.45 | 1.71 | 0.37 | 1.01 | - |
| CTGAN+DRL | 0.28 | 0.19 | 0.22 | 0.14 | 0.16 |
| TVAE | 0.14 | 0.08 | 0.07 | 0.06 | 0.05 |
| TVAE+RS | 0.37 | 0.31 | 1.08 | 0.96 | - |
| TVAE+DRL | 0.25 | 0.16 | 0.20 | 0.11 | 0.14 |
| GOGGLE | 0.22 | 0.08 | 0.08 | 0.06 | 0.06 |
| GOGGLE+RS | 0.48 | 0.35 | 5.63 | 0.25 | - |
| GOGGLE+DRL | 0.29 | 0.14 | 0.11 | 0.09 | 0.16 |
| **Avg. DGM** | 0.15 | 0.08 | 0.07 | 0.06 | 0.05 |
| **Avg. DGM+RS** | 0.37 | 0.83 | 1.54 | 0.66 | - |
| **Avg. DGM+DRL** | 0.22 | 0.13 | 0.14 | 0.10 | 0.13 |

As in the previous experiment, I use the Wilcoxon signed-rank test to assess whether adding DRL significantly improves over the linear layer LL. As expected, the test reveals that the performances of DGMs+DRL and DGMs+LL are not statistically different w.r.t. any of the metrics.

### 5.4.4 Sample Generation Time

To assess the impact of constraints on sample generation time, I compare the runtimes of the unconstrained DGMs, the DGMs+RS, and the DGMs+DRL. I generate 1,000 samples for each model and dataset using five different seeds and report the mean runtime (over the different seeds) in Table 5.13. Additionally, in the last three rows of the Table, I report the average runtime across all datasets. As expected, DGMs+DRL are slower on average than their unconstrained counterparts. However, they are faster than the DGM+RS models.

The largest runtime difference registered between an unconstrained DGM and its constrained counterpart is of only 0.12 seconds (i.e., for CTGAN on the URL and LCLD datasets). Notably, in 20 out of 25 cases, the additional runtime recorded for the DGM+DRL models is less than 0.1 seconds.

On the other hand, for the DGM+RS models, the sample generation procedure timed out after 24h for all the models tested on the House dataset (where the CVR reached 100%). The registered times are also not very promising for any of the other datasets when using DGM+RS, where the minimum absolute difference registered equals 0.07 seconds and the maximum equals 5.55 seconds (notice that no DGM nor DGM+DRL model has sampling time above 0.29 seconds). Indeed, excluding extreme cases with 100% CVR (where it was not possible to generate samples even in 24h), in all other cases, the DGMs+RS take more than twice as long as the unconstrained DGMs.

## 5.5 Related Work

This framework lies at the intersection of two fields: neuro-symbolic AI and tabular data generation. Thus, this section on the related work will mirror this duality.

**Neuro-symbolic AI.** Neuro-symbolic AI (Raedt et al., 2020; d'Avila Garcez and Lamb, 2023) refers to the broad area of AI that combines the strengths of symbolic reasoning with neural networks. As the framework I introduced in this Chapter falls into the more specific field of injection of background knowledge into neural models, (see, e.g., (Stewart and Ermon, 2017; Hoernle et al., 2022; Giunchiglia et al., 2023a; Daniele et al., 2023b; Calanzone et al., 2025)), I will focus the discussion on this topic.

Many methods for this task are based on the intuition that logical constraints can be transformed into differentiable loss function terms that penalise the networks for violating them (see, e.g., (Xu et al., 2018; Badreddine et al., 2022; Diligenti et al., 2012; Fischer et al., 2019)). As expected, since these methods operate at a loss level, they give no guarantee that the constraints will be satisfied. Other works manage to integrate neural networks and probabilistic reasoning by mapping the predicates appearing in logical formulae into neural networks (Manhaeve et al., 2018; Yang et al., 2020; Sachan et al., 2018; Pryor et al., 2023; van Krieken et al., 2023). This procedure allows for performing reasoning on the networks' predictions as well as constraining the output according to the background knowledge. The most similar line of work

to this Chapter's is the one where the constraints in input are automatically compiled into neural layers (Giunchiglia and Lukasiewicz, 2021; Ahmed et al., 2022b; Giunchiglia et al., 2024). However, these methods can compile and incorporate constraints that are at best as expressive as propositional logic formulae.

Focusing specifically on the incorporation of constraints on generic generative models, we can find the work by Stoian et al. (2024a) (i.e., the work on which Chapter 3 was based on), where the tabular generation process was simply constrained by linear inequalities. When considering different application domains, we can find the work proposed by Misino et al. (2022), where ProbLog (Raedt et al., 2007) works in tandem with variational-autoencoders, and the one by Liello et al. (2020), where the authors incorporate propositional logic constraints on GANs for structured objects generation. For surveys on the integration of formally expressed background knowledge in neural networks see (Giunchiglia et al., 2022; Dash et al., 2022).

**Tabular Data Generation.** In recent years, various DGMs have been proposed to tackle the problem of tabular data synthesis. Many of these approaches are based on Generative Adversarial Networks (GANs), like TableGAN (Park et al., 2018), CTGAN (Xu et al., 2019), IT-GAN (Lee et al., 2021), OCT-GAN (Kim et al., 2021), and PacGAN (Lin et al., 2018). Other methods try to reduce the problems that often characterise GANs, such as mode collapse and unstable training, by introducing Variational AutoEncoders (VAEs) based models, see, e.g., (Xu et al., 2018; Srivastava et al., 2017; Wan et al., 2017). An alternative solution to such problems is given by the usage of denoising diffusion probabilistic models as done in (Kotelnikov et al., 2023) or (Kim et al., 2023), where the authors designed a self-paced learning method and a fine-tuning approach to adapt the standard score-based generative modelling to the challenges of tabular data generation. Finally, GOGGLE (Liu et al., 2022) uses graph learning to infer relational structure from the data and use it to their advantage especially in data-scarce settings. Since synthetic tabular data are often used to replace the original dataset to preserve privacy in sensitive settings, a parallel line of research revolves around the development of DGMs with privacy guarantees. Examples of models that have such privacy guarantees are PATEGAN (Yoon et al., 2020) and DP-CGAN (Torkzadehmahani et al., 2020). For a survey on deep generative modelling see (Stoian et al., 2026), which part of Chapter 2 is based on.

## 5.6 Conclusions

In this Chapter, I have proposed the Disjunctive Refinement Layer (DRL), the first-ever neuro-symbolic AI layer able to automatically compile constraints expressed as QFLRA formulae into a neural layer and thus guarantee their satisfaction. This sort of work is needed in the tabular data synthesis field, as the experimental analysis shows that Deep Generative Models (DGMs) very frequently generate datapoints that are not aligned with the background knowledge, with some extreme cases where all the datapoints are violating the constraints. DRL presents many desirable properties: (i) it can be seamlessly integrated into the architecture of any neural network, (ii) it allows for the backpropagation of the gradients, (iii) it performs all the computations in a single forward pass (i.e., there are no cycles), (iv) it optimally refines the original predictions and, last but not least, (v) it improves the performance of all the tested DGMs in terms of machine learning efficacy. Indeed, in the experimental analysis DRL improved the performance for all datasets of up to 21.4%, 20.5%, and 20.9% in terms of F1-score, weighted F1-score, and Area Under the ROC Curve, respectively.

# Chapter 6

# PiShield: Learning with Real Constraints

In this Chapter, which is based on (Stoian et al., 2024b) and (Stoian et al., 2023), I introduce PiShield, the first PyTorch-based package designed for integrating background knowledge into neural networks via layer-based and loss-based methods.

PiShield is designed to be a versatile tool for applying neuro-symbolic approaches—whether layer- or loss-based—to practical problems. A key motivation for creating PiShield was to make my proposed neuro-symbolic methods easily accessible to researchers and practitioners working with real-world applications. To this end, PiShield implements my methods described so far in Chapters 3-5, but also my work on guiding neural networks via a logic-based loss function, which I present in detail in Section 6.2. In addition to the work described in this thesis, the package also incorporates the method from (Giunchiglia et al., 2024), a paper I have co-authored, but which I have not included in my thesis.

## 6.1 Motivation and Overview

Neuro-symbolic AI methods can be broadly classified into two categories. The first comprises methods able to integrate constraints in the loss function and penalise the neural network models when they violate the constraints (e.g., (Diligenti et al., 2012, 2017b; Donadello et al., 2017; Xu et al., 2018; Fischer et al., 2019; Nandwani et al., 2019; Badreddine et al., 2022; Li et al., 2023; Ahmed et al., 2022c; Stoian et al., 2023)). Emerging later, the second category consists of methods that incorporate constraints directly directly in the topology of the neural networks to guarantee their satisfaction (e.g., (Giunchiglia and Lukasiewicz, 2021; Hoernle et al., 2022; Ahmed et al., 2022b)).

While the previous Chapters of this thesis were dedicated to the latter category, the focus of this Chapter is on the loss-based approaches. Although these methods do not offer the formal guarantees of the latter category w.r.t. the constraints' satisfaction, they are a powerful tool for guiding models towards more reliable behaviour, especially in complex scenarios where architectural changes might be infeasible.

In this Chapter, I propose PiShield[1,2], a PyTorch-based package supporting both of these two categories of neuro-symbolic AI methods and allowing for seamlessly integrating domain constraints into neural networks by means of two core mechanisms:

1. Shield Layers: The focus of the previous Chapters, these layers can be built on top of any neural network. Adhering to the principles outlined in the previous Chapters, they advocate for a more-constraints-driven machine learning by bringing constraints into neural networks, and guarantee the satisfaction of the constraints regardless of the input.

2. Shield Losses: The focus of this Chapter, these losses can be added to the training objective of neural networks to penalise outputs that contradict the background knowledge constraints, thus guiding the learning process.

The initial version of PiShield, from (Stoian et al., 2024b), focused exclusively on the layer-based neuro-symbolic methods, such as the Linearly-constrained Layer (from Chapter 3) and the Disjunctive Refinement Layer (from Chapter 5). However, the latest version extends PiShield with Shield Losses, which are suitable for complex scenarios with a large number of constraints and which I will present here in Section 6.2, within the context of autonomous driving. Specifically, this extension is based on (Stoian et al., 2023), where I introduced a memory-efficient logic-based loss that guides the learning of neural networks using propositional logic constraints.

## 6.2 Shield Loss: Memory-efficient Logic-based Loss

A popular method to include background knowledge expressed as logical constraints into neural networks involves relaxing the constraints using T-norms and adding them in the loss function (e.g., (Diligenti et al., 2017b; Donadello et al., 2017; Diligenti et al., 2017a)). However, T-norm-based losses tend to have high memory requirements due to their need to: (i) compute the T-norm for each constraint and each prediction, where the constraints involve multiple variables that are to be instantiated to the

[1]Code: https://github.com/mihaela-stoian/PiShield
[2]Website: https://sites.google.com/view/pishield

corresponding values in the predictions, (ii) then aggregate the T-norm based results to obtain a single value that gives the degree of constraint satisfaction (across all constraints) for the set of predictions. Therefore, in scenarios where there is a large number of constraints, if not implemented in a way to minimise the computational costs, standard implementations of T-norm-based losses generate massive intermediate tensors (e.g., storing values for all labels across all constraints, even if those labels do not appear in all constraints). This dense representation forces the allocation of memory for all possible combinations (most of which are zeros), and, thus, may be infeasible (or even impossible) to apply when considering complex application domains like event detection in autonomous driving.

In this Section, I introduce Shield Losses, which are memory-efficient logic-based loss functions that allow for exploiting T-norms for such demanding tasks. Through an extensive experimental analysis on the ROAD-R dataset (Giunchiglia et al., 2023b), I show that my approach can be implemented and run on GPUs with less than 25 GiB of available memory, while standard T-norm-based losses are estimated to require more than 100 GiB, far exceeding the amount of memory normally available. Further, by training a state-of-the-art event detection model using different percentages of labelled training data (i.e., 10%, 20%, 50%, 75%, and 100%), I show that while logic-based losses can improve the performance of the models in all cases, they are particularly helpful when data are scarce. Indeed, the models yield an improvement of up to 1.85% and 3.95% when using, 10% and 20% of the labelled training data, respectively. Finally, I investigate the behaviour of the memory-efficient logic-based loss when only 10% annotated data are available, along with 10% unlabelled data, and find that applying the new loss on both labelled and unlabelled data after a warm-up training phase leads to further improvements, i.e., up to 2.75% w.r.t. the fully-supervised baseline.

### 6.2.1 Problem Statement

In the task of event detection in video, the goal is to identify and classify multiple objects within each frame, for a given dataset $\mathcal{X}$ that contains labelled datapoints. Specifically, each datapoint consists of an image $\mathbf{X}$ and of its ground truth $\mathcal{Y}$. The image $\mathbf{X} \in \mathbb{R}^{3\times W\times H}$ is a multi-dimensional array (i.e., tensor) representing a single video frame, where $W$ and $H$ are the frame's width and height, respectively, and 3 corresponds to the three different RGB colour channels. Furthermore, the ground truth $\mathcal{Y}$ for $\mathbf{X}$ is a set of pairs $(\mathbf{b}, \mathcal{P})$, each pair identifying an object in the frame, since a single frame may contain multiple objects. More precisely, each pair consists of

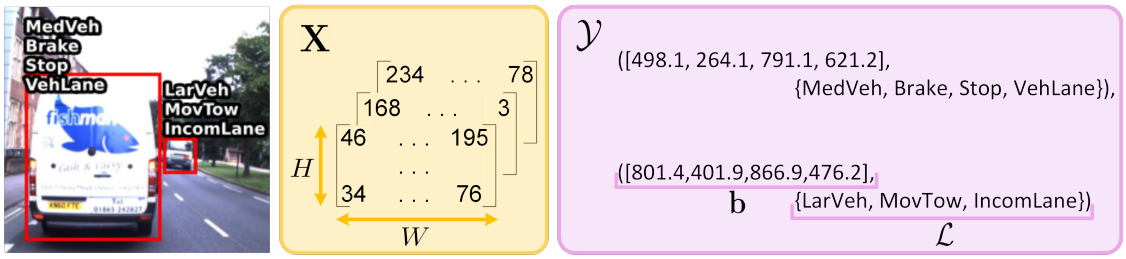



Figure 6.1: Visualisation of a datapoint consisting of a frame $\mathbf{X}$ and its ground truth $\mathcal{Y}$, where there are two objects in the frame.

a *bounding box* $\mathbf{b} \in \mathbb{R}^4$ and, importantly, a set of predicted labels $\mathcal{P}$ associated with that object, since there can be multiple labels that apply to the detected object. The bounding box $\mathbf{b}$ marks a rectangle around the detected object and, thus, identifies the position of the object within the frame. Figure 6.1 illustrates an example of a datapoint $\mathbf{X}$ and its ground truth $\mathcal{Y}$, given an image with two objects. Note that since there are two objects in the frame, $\mathcal{Y}$ contains two elements.

An event detection model processes a sequence of video frames, producing outputs for each frame $\mathbf{X}$. These outputs are pairs $(\hat{\mathbf{b}}, \hat{\mathbf{p}})$, where $\hat{\mathbf{b}} \in \mathbb{R}^4$ is a predicted bounding box and $\hat{\mathbf{p}} \in [0,1]^{|\mathcal{L}|}$ is a vector of confidence scores, for all possible event labels in the set $\mathcal{L}$, indicating which labels can be associated with the object identified by $\hat{\mathbf{b}}$.

In order to generate bounding box predictions, these models typically use a set of $D$ anchor boxes, which are predefined boxes of varying sizes and and aspect ratios positioned at different locations within the frame. Specifically, the model predicts the generated bounding box locations by calculating their offsets relative to these anchors. A final prediction is formed by associating a set of labels $\hat{\mathcal{P}}$ to each bounding box $\hat{\mathbf{b}}$. Note that a label $L \in \mathcal{L}$ is included in $\hat{\mathcal{P}}$ if its corresponding confidence score, $\hat{\mathbf{p}}_L$, is at least as large as a user-defined threshold $\theta$.

To ensure the model's outputs are compliant with the background knowledge, I first introduce a finite set of propositional logic constraints, denoted by $\Pi$, where each constraint $\psi$ is a clause expressed over the set $\mathcal{L}$ of all possible output labels. Specifically, each $\psi$ is a disjunction of literals $l_1, l_2, \ldots, l_{|\psi|}$, where a literal $l_i$ is either a label $L \in \mathcal{L}$ or its negation, $\neg L$, for all $i \in \{1, 2, \ldots, |\psi|\}$ and $|\psi|$ is the number of literals appearing in $\psi$. Example 6.2.1 illustrates a possible set of constraints $\Pi$ for a simple event detection problem in the context of autonomous driving, with only three labels available.

**Example 6.2.1.** *Let $\mathcal{L} = \{Car, Moving, Stopped\}$ be the set of labels for an event detection problem. And suppose that the background knowledge consists of the follow-*

*ing two constraints on the possible combinations of labels that can be assigned to an object: (i) if an object is labelled with Moving, then that object is a Car, and (ii) if an object is Moving, then it is not Stopped. Clearly, it is possible to express the two constraints as:*

$$\psi_1 = (\neg Moving \vee Car),$$
$$\psi_2 = (\neg Moving \vee \neg Stopped),$$

*and, thus, the set of constraints can be captured as* $\Pi = \{\psi_1, \psi_2\}$.

Intuitively, a clause $\psi$ means that for any predicted bounding box $\hat{\mathbf{b}}$ its associated set of predicted labels $\hat{\mathcal{P}}$ must make the statement of $\psi$ true. And for the clause to be satisfied, at least one of its literals must be true for that prediction. If at least one of the literals in $\psi$ is true, then $\psi$ is *satisfied*. Formally, a label $L \in \mathcal{L}$ *occurs positively* (resp., *negatively*) in the clause $\psi$ if there is a literal $l$ in $\psi$ such that $l = L$ (resp., $l = \neg L$). Further, $L$ *occurs* in $\psi$ if $L$ occurs either positively or negatively in $\psi$.

A positive literal $l$ is satisfied if its associated label $L$ is included in the final predicted set $\hat{\mathcal{P}}$. Similarly, a negative literal $\neg l$ is satisfied if its associated label $L$ is not included in the final predicted set $\hat{\mathcal{P}}$. I assume w.l.o.g. that in any given clause, a label can appear at most once, either in its positive or negative form.

If a set of predicted labels $\hat{\mathcal{P}}$ for a detected bounding box $\hat{\mathbf{b}}$ does not satisfy a clause $\psi$, then $\hat{\mathcal{P}}$ *violates* $\psi$. A set of clauses $\Pi$ is satisfied by $\hat{\mathcal{P}}$ if all its clauses are satisfied by $\hat{\mathcal{P}}$. The set $\Pi$ is satisfiable if there is at least one set of labels that satisfies $\Pi$. Finally, a model $m$ is compliant with $\Pi$ if all of its generated sets of labels satisfy $\Pi$.

### 6.2.2 Memory-efficient Logic-based Loss

Inspired by the semantic-based regularisation (SBR) framework (Diligenti et al., 2017a,b), I added a new regularisation term to the existing losses (i.e., to the localisation and classification losses) to express the degree of the logical constraints satisfaction. While the logic-based function I use is mathematically equivalent to the SBR formulation (i.e., relying on T-norms to relax logical constraints into a differentiable regularisation term), my contribution is a novel computational implementation. In particular, the standard SBR approach relies on tensor operations that do not scale to complex domains such as autonomous driving. For instance, applying the standard SBR implementation to the ROAD-R dataset (Giunchiglia et al., 2023b) would require over 100 GiB of GPU memory, which is infeasible in practice. My method relies

on a sparse matrix representation that exploits the fact that constraints typically involve only a small subset of labels, allowing T-norm-based losses to be applied to large-scale, resource-intensive tasks where the standard implementation would fail, as detailed in this Section.

Since all constraints in $\Pi$ are clauses (i.e., disjunctions of literals), it is easy to convert each of them into a form containing only negations and conjunctions (i.e., $l_1 \vee l_2 \vee \cdots \vee l_n \equiv \neg(\neg l_1 \wedge \neg l_2 \wedge \cdots \wedge \neg l_n)$). This step can be seen as relaxing:

1. the conjunction by using different triangular norms (T-norms) (Metcalfe, 2005), where a *T-norm* is a function $\top : [0,1]^2 \to [0,1]$ such that, for every $a, b, c \in [0,1]$, the following properties hold:

$$\begin{aligned}
\top(a,b) &= \top(b,a) && \text{(commutativity)}\\
\top(a,1) &= a && \text{(1 is the identity element)}\\
\top(a,0) &= 0 && \text{(0 is the absorbing element)}\\
\top(a,\top(b,c)) &= \top(\top(a,b),c) && \text{(associativity)}\\
a \leq b &\to (\top(a,c) \leq \top(b,c)) && \text{(monotonicity).}
\end{aligned}$$

2. the negation by using *strong negation*, which, given $a \in [0,1]$, is defined as $1-a$.

The above step is equivalent to directly relaxing the disjunction using the appropriate T-conorm. In general, a *T-conorm* is a function $\perp : [0,1]^2 \to [0,1]$ such that, for every $a, b \in [0,1]$, $\perp(a,b) = 1 - \top(1-a, 1-b)$. Thus, it is easy to see that $\perp$ has the following properties:

$$\begin{aligned}
\perp(a,b) &= \perp(b,a) && \text{(commutativity)}\\
\perp(a,0) &= a && \text{(0 is the identity element)}\\
\perp(a,1) &= 1 && \text{(1 is the absorbing element)}\\
\perp(a,\perp(b,c)) &= \perp(\perp(a,b),c) && \text{(associativity)}\\
a \leq b &\to (\perp(a,c) \leq \perp(b,c)) && \text{(monotonicity)}
\end{aligned}$$

for every $a, b \in [0,1]$. In the first two columns of Table 6.1, I summarise the most used T-norms together with their respective T-conorms.

To simplify the notation, I will extend the definitions of $\top$ and $\perp$ to support matrices and tensors (which is possible given the properties, in particular the commutativity and the associativity, of the T-norms and T-conorms). Specifically, given a tensor $\mathbf{Q}$ of size $p \times q \times r$, $\perp(\mathbf{Q})$ returns a matrix of size $p \times q$ whose element at

Table 6.1: The most common T-norms and their associated T-conorms.

| | T-norm | T-conorm |
|---|---|---|
| Gödel | $\min(a, b)$ | $\max(a, b)$ |
| Łukasiewicz | $\max(a + b - 1, 0)$ | $\min(a + b, 1)$ |
| Product | $a \cdot b$ | $1 - (1 - a)(1 - b)$ |

position $(i, j)$ is equal to the value of the T-conorm computed over the last dimension of $\mathbf{Q}$, i.e. over the vector[3] $\mathbf{Q}_{ij:}$. For instance, $\perp(\mathbf{Q}) = \max \mathbf{Q}_{ij:} = \max_{k=1}^{r} \mathbf{Q}_{ijk}$, when using the Gödel T-conorm. Furthermore, given two matrices $\mathbf{Q}$ and $\mathbf{R}$ of the same size, $\perp(\mathbf{Q}, \mathbf{R})$ returns a matrix of the same size whose element at position $(i, j)$ is $\perp(\mathbf{Q}, \mathbf{R})_{ij} = \perp(\mathbf{Q}_{ij}, \mathbf{R}_{ij})$. Analogously, the definition of $\top$ can be extended to tensors and matrices in a similar way.

I will show how to implement logic-based loss functions: (i) by the standard approach, and (ii) by my memory-efficient method, which relies on sparse tensors. Before presenting each solution, I will introduce the relevant terminology. Given an event detection problem with propositional logic constraints $\Pi$, it is possible to express $\Pi$ using two matrices $\mathbf{C}^+$ and $\mathbf{C}^-$, both of size $|\Pi| \times |\mathcal{L}|$, such that:

$$\mathbf{C}^+_{ij} = \begin{cases} 1, & \text{if the } j^{\text{th}} \text{ label appears positively in the } i^{\text{th}} \text{ constraint,} \\ 0, & \text{otherwise,} \end{cases}$$

and

$$\mathbf{C}^-_{ij} = \begin{cases} 1, & \text{if the } j^{\text{th}} \text{ label appears negatively in the } i^{\text{th}} \text{ constraint,} \\ 0, & \text{otherwise.} \end{cases}$$

Let $m$ be a model for this problem that uses $D$ anchor boxes for each frame. For each input frame, $m$ outputs the *prediction matrix* $\mathbf{P}$ of size $D \times |\mathcal{L}|$. Given $\mathbf{P}$, $\mathbf{C}^+$ and $\mathbf{C}^-$, the goal is to compute the degree of satisfaction of each constraint for each output, which can be compactly expressed as a matrix $\mathbf{G}$ of size $D \times |\Pi|$. Ultimately, $\mathbf{G}$ can be used to compute the frame-wise logic-based regularisation term in the loss as follows:

$$L_{logic}(\mathbf{G}) = 1 - \frac{1}{D} \frac{1}{|\Pi|} \sum_{ij} \mathbf{G}_{ij} \,. \tag{6.1}$$

[3]To ease the notation, I denote the vector $\mathbf{Q}_{i,j,1:r}$ by $\mathbf{Q}_{ij:}$.

**Standard approach.** Recall that $\mathbf{P}$ is a matrix of size $D \times |\mathcal{L}|$, and $\mathbf{C}^+$ and $\mathbf{C}^-$ are matrices of size $|\Pi| \times |\mathcal{L}|$. Starting from these three matrices, the standard approach constructs three tensors, each of size $D \times |\Pi| \times |\mathcal{L}|$, to align their dimensions for element-wise operations. To this end, let $\hat{\mathbf{P}}$ be the tensor obtained by repeating $\mathbf{P}$ along a new second dimension $|\Pi|$ times, and let $\hat{\mathbf{C}}^+$ and $\hat{\mathbf{C}}^-$ be the tensors obtained by repeating $\mathbf{C}^+$ and $\mathbf{C}^-$, respectively, along a new first dimension $D$ times. Then, the matrix $\mathbf{G}$ can be computed directly by using a chosen T-conorm along the last dimension:

$$\mathbf{G} = \perp(\hat{\mathbf{P}} \odot \hat{\mathbf{C}}^+ + \hat{\mathbf{C}}^- - \hat{\mathbf{P}} \odot \hat{\mathbf{C}}^-),$$

where $\odot$ is the Hadamard (element-wise) product.

**Example 6.2.2 (cont'd).** *Given the event detection problem, from Example 6.2.1, with propositional logic constraints* $\Pi = \{\neg \mathit{Moving} \vee \mathit{Car}, \neg \mathit{Moving} \vee \neg \mathit{Stopped}\}$ *written over the set of labels* $\mathcal{L} = \{\mathit{Car}, \mathit{Moving}, \mathit{Stopped}\}$*, I assume that the labels are numbered as listed in* $\mathcal{L}$*. Then, the matrices* $\mathbf{C}^+$ *and* $\mathbf{C}^-$ *are as follows:*

$$\mathbf{C}^+ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{C}^- = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

*Further, suppose the model* $m$ *for this problem uses 3 anchor boxes and produces a prediction matrix* $\mathbf{P}$*:*

$$\mathbf{P} = \begin{bmatrix} 0.1 & 0.7 & 0.3 \\ 0.9 & 0.9 & 0.2 \\ 0.2 & 0.9 & 0.9 \end{bmatrix}.$$

*Then, with the Gödel T-conorm, the goal matrix* $\mathbf{G}$ *is equal to:*

$$\mathbf{G} = \begin{bmatrix} 0.3 & 0.7 \\ 0.9 & 0.8 \\ 0.2 & 0.1 \end{bmatrix}.$$

The approach has a critical shortcoming, namely: it requires working with dense 3-dimensional tensors, inducing a large computational overhead. This makes the method unfeasible, especially for demanding application domains like autonomous driving.

For example, consider an event detection scenario with $|\Pi| = 200$ constraints, $|\mathcal{L}| = 50$ labels, and a model that processes 10-frame sequences with $D = 55$K anchor boxes per frame (a common number for event detection problems). This translates to processing 550K anchors per sequence, and storing just a single tensor of size $D \times |\Pi| \times |\mathcal{L}|$ requires approximately 20 GiB ($550,000 \times 200 \times 50 \times 4$ bytes).

Moreover, the standard approach works with five matrices of this size to compute the loss $L_{logic}$ in the forward pass and then backpropagate through it. Excluding any

Table 6.2: Operations used to update the goal matrix $\mathbf{G}$, on the grounds of the chosen T-conorm, when computing the memory-efficient logic-based loss. To simplify the notation, in the last two columns, I use 1 to refer to the matrix of ones of appropriate size.

| | Operation to update $\mathbf{G}_{s_L^+}$ | Operation to update $\mathbf{G}_{s_L^-}$ |
|---|---|---|
| Gödel | $\max(\mathbf{G}_{s_L^+}, \mathbf{P}_L \cdot \mathbb{1}^T_{\lvert s_L^+ \rvert})$ | $\max(\mathbf{G}_{s_L^-}, 1 - \mathbf{P}_L \cdot \mathbb{1}^T_{\lvert s_L^- \rvert})$ |
| Łukasiewicz | $\min(\mathbf{G}_{s_L^+} + \mathbf{P}_L \cdot \mathbb{1}^T_{\lvert s_L^+ \rvert}, 1)$ | $\min(\mathbf{G}_{s_L^-} + 1 - \mathbf{P}_L \cdot \mathbb{1}^T_{\lvert s_L^- \rvert}, 1)$ |
| Product | $1 - (1 - \mathbf{G}_{s_L^+}) \odot (1 - \mathbf{P}_L \cdot \mathbb{1}^T_{\lvert s_L^+ \rvert})$ | $1 - (1 - \mathbf{G}_{s_L^-}) \odot (\mathbf{P}_L \cdot \mathbb{1}^T_{\lvert s_L^- \rvert})$ |

other memory needed for storing the input and intermediate outputs, just computing the loss would take over 100 GiB. This exceeds the memory space limits of even high-end GPUs (e.g., the NVIDIA A100 Tensor Core GPU having 80 GiB RAM). Furthermore, notice that this computation is for a single sequence of only 10 frames. However, deep learning models for event detection are generally trained using batches of 4 or even 8 sequences, each comprising up to 32 frames in sequence. It is thus impossible to use the above standard dense representation to train event detection models, even with the largest GPU available to date.

**Memory-efficient approach.** In practice, most of the constraints are written over a subset of the available labels $\mathcal{L}$, and this subset is usually much smaller than $\mathcal{L}$. For example, in the experimental analysis that I will present later, although $\mathcal{L}$ consists of 41 labels, the longest constraint is written over just 15 labels. As a result, $\mathbf{C}^+$ and $\mathbf{C}^-$ contain mostly zeros. Hence, I designed a method to capture the logic-based loss that makes use of this sparsity property and ultimately avoids the high computational costs induced by the 3D tensors, operating only on 2D matrices.

Given $\Pi$, I associate with each constraint an index and then define two sets of sequences: (i) $\mathcal{S}^+ = \{s_L^+ : L \in \mathcal{L}\}$, where $s_L^+$ is the sequence of indices of the constraints in which $L$ occurs positively, and (ii) $\mathcal{S}^- = \{s_L^- : L \in \mathcal{L}\}$, where $s_L^-$ is the sequence of indexes of the constraints in which $L$ occurs negatively. These sets of sequences allow for immediately accessing the constraints which each label occurs in when needed.

To compute the goal matrix $\mathbf{G}$, I first instantiate it to the identity element of the disjunction. Then, I iterate through the labels in $\mathcal{L}$ and, for each label $L \in \mathcal{L}$, I update all the columns of $\mathbf{G}$ associated with constraints in which $L$ occurs, according to the values in $\mathbf{P}$. More precisely, I set $\mathbf{G} = \mathbf{0}_{D \times |\Pi|}$, where $\mathbf{0}_{D \times |\Pi|}$ is a matrix of zeros

of size $D \times |\Pi|$, and then, for each label $L \in \mathcal{L}$, I compute:

$$\mathbf{G}_{s_L^+} \longleftarrow \perp(\mathbf{G}_{s_L^+}, \mathbf{P}_L \cdot \mathbb{1}^T_{|s_L^+|})$$
$$\mathbf{G}_{s_L^-} \longleftarrow \perp(\mathbf{G}_{s_L^-}, 1 - \mathbf{P}_L \cdot \mathbb{1}^T_{|s_L^-|}),$$

where (i) $\mathbf{G}_{s_L^+}$ (resp., $\mathbf{G}_{s_L^-}$) selects the columns of $\mathbf{G}$ associated with constraints in which $L$ occurs positively (resp., negatively), (ii) $\mathbf{P}_L$ corresponds to the column of $\mathbf{P}$ associated with the label $L$, and (iii) $\mathbb{1}_n$ indicates the unit column vector with $n$ elements. In Table 6.2, I give the update operations[4] for three of the most common T-conorms: Gödel, Łukasiewicz, and Product T-conorms.

Finally, I compute $L_{logic}(\mathbf{G})$ as defined in Equation 6.1. Example 6.2.3 shows how my proposed memory-efficient approach computes the goal matrix $\mathbf{G}$ step by step.

**Example 6.2.3** (cont'd)**.** *Given the same event detection problem, from Example 6.2.2, assume that each constraint in* $\Pi$ *is assigned an index: constraint* $(\neg Moving \vee Car)$ *index 0, and constraint* $(\neg Moving \vee \neg Stopped)$ *index 1. Further, let* $\epsilon$ *denote the empty sequence. Then, the sequences associated with each label are:*

$$s^+_{Car} = (0), \qquad s^-_{Car} = \epsilon,$$
$$s^+_{Moving} = \epsilon, \qquad s^-_{Moving} = (0, 1),$$
$$s^+_{Stopped} = \epsilon, \qquad s^-_{Stopped} = (1).$$

*Recall that the prediction matrix* $\mathbf{P}$ *obtained from the model is the following:*

$$\mathbf{P} = \begin{bmatrix} 0.1 & 0.7 & 0.3 \\ 0.9 & 0.9 & 0.2 \\ 0.2 & 0.9 & 0.9 \end{bmatrix}.$$

*Supposing that the Gödel T-conorm is used,* $\mathbf{G}$ *is initialised with matrix* $\mathbf{0}_{3\times 2}$ *(as there are 3 bounding boxes and 2 constraints). Then, the first update of* $\mathbf{G}$ *is w.r.t. the label Car. Since Car appears only in one constraint, only the column of* $\mathbf{G}$ *associated with that constraint will be updated:*

$$\mathbf{G}_{j^+_{Car}} = \max\left( \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 \\ 0.9 \\ 0.2 \end{bmatrix} \right) = \begin{bmatrix} 0.1 \\ 0.9 \\ 0.2 \end{bmatrix},$$

*which changes the goal matrix to:*

$$\mathbf{G} = \begin{bmatrix} 0.1 & 0 \\ 0.9 & 0 \\ 0.2 & 0 \end{bmatrix}.$$

[4]Notice that, given two matrices, max (resp., min) represent the element-wise operation taking the maximum (resp., minimum) between two elements at the same position in the two input matrices.

*Since $j^-_{\text{Car}} = \epsilon$, there is no need to further update $\mathbf{G}$ for Car. So the next step is to consider the label Moving and update $\mathbf{G}$ according to $s^-_{Moving}$ (as $s^+_{Moving} = \epsilon$). This time all constraints involve Moving. Thus, the update affects all columns of the goal matrix:*

$$\mathbf{G}_{s^-_{Moving}} = \max\left(\begin{bmatrix}0.1 & 0\\ 0.9 & 0\\ 0.2 & 0\end{bmatrix},\ 1 - \begin{bmatrix}0.7 & 0.7\\ 0.9 & 0.9\\ 0.9 & 0.9\end{bmatrix}\right) = \begin{bmatrix}0.3 & 0.3\\ 0.9 & 0.1\\ 0.2 & 0.1\end{bmatrix},$$

*and, clearly, $\mathbf{G}$ is the same as $\mathbf{G}_{s^-_{Moving}}$ after this step:*

$$\mathbf{G} = \begin{bmatrix}0.3 & 0.3\\ 0.9 & 0.1\\ 0.2 & 0.1\end{bmatrix}.$$

*The last step is to update $\mathbf{G}$ w.r.t. the label Stopped:*

$$\mathbf{G}_{s^-_{Stopped}} = \max\left(\begin{bmatrix}0.3\\ 0.1\\ 0.1\end{bmatrix},\ 1 - \begin{bmatrix}0.3\\ 0.2\\ 0.9\end{bmatrix}\right).$$

*Thus, the goal matrix is:*

$$\mathbf{G} = \begin{bmatrix}0.3 & 0.7\\ 0.9 & 0.8\\ 0.2 & 0.1\end{bmatrix}.$$

As expected, the goal matrix $\mathbf{G}$ obtained in Example 6.2.3, contains low values in the last row, as the last prediction (i.e., the last row of $\mathbf{P}$) clearly violates both constraints from $\Pi$ w.r.t. any threshold larger than 0.5, above which a label is predicted as positive.

### 6.2.3 Experimental Analysis

To evaluate the efficiency of my proposed memory-efficient logic-based loss, I conduct experiments in the context of event detection for autonomous driving. The goal of this task is to assign to each detected bounding box in each video a subset of labels, typically containing one object (also referred to as agent) label, and a subset of the action (i.e., what the agent is doing) and location (i.e., where the agent is located) labels.

To this end, I use the ROAD-R dataset (Giunchiglia et al., 2023b) for autonomous driving, which extends the ROAD dataset (Singh et al., 2023) with 243 manually annotated constraints, provided in disjunctive normal form. Examples of these constraints can be found in Table H.4 from Appendix H.3. The dataset contains 22 videos, each approximatively 8 minutes long, annotated with tubelets (also referred to as tubes)

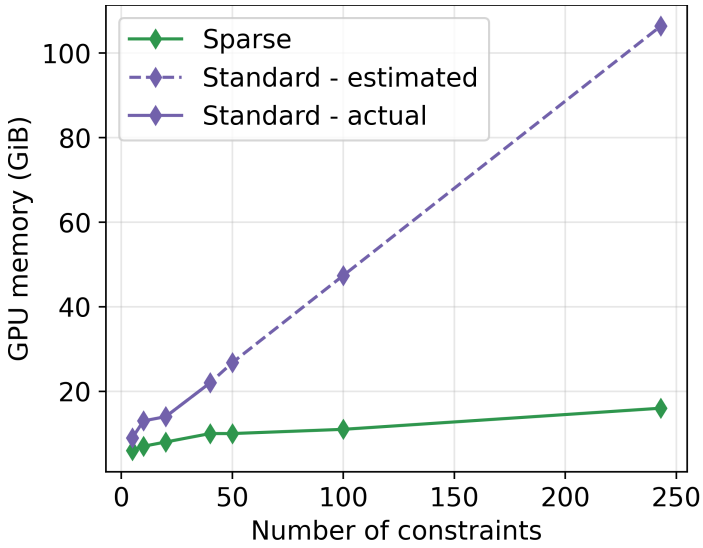



Figure 6.2: Comparison between the standard and the proposed approach in terms of GPU memory allocated when using different number of constraints. Each point on the continuous (resp., dashed) line corresponds to an actual observation (resp., estimate).

that link a sequence of bounding boxes in time. Each bounding box is annotated with a subset of the 41 labels available, which can be grouped into three categories, depending on which class of objects they represent: agents, actions, and locations. For these experiments, I use the dataset's training partition for training the models and, for reproducibility purposes, I report the results on the held-out validation dataset, as the test set is not publicly available.

The models are based on the 3D-RetinaNet (Singh et al., 2023) detector with a ResNet50 (He et al., 2016) backbone. For temporal feature learning, I combine this with a Random Connectivity Gated Recurrent Unit (RCGRU) (Hua et al., 2018), which I selected for its high performance in (Giunchiglia et al., 2023b). The models use sequences of 8 frames as input and a weight of 10 for the logic-based loss term.

**Memory assessment.** To assess the efficiency of my method for computing the logic-based loss w.r.t. how much GPU memory is allocated during training the models, I compared it to the standard implementation of the logic-based loss. I used a Titan RTX GPU with 24 GiB of RAM for training models for 50 iterations on ROAD-R, while using different numbers of constraints. Note that, most constraints in ROAD-R contain only two labels. However, to ensure a fair comparison with the standard implementation, I selected constraints with a variety of label counts. For reference, in each iteration, the number of anchors $D$ was of about 67K. Figure 6.2 shows that

my proposed method significantly reduced the memory costs, making it possible to use logic-based losses on the ROAD-R dataset, where the number of constraints and datapoints per batch are both large. Using the standard implementation supports at most 40 constraints—this is 203 constraints less than the ones in ROAD-R.

**Results.** I first investigate how the memory-efficient logic-based loss performs in a fully-supervised scenario. To this end, I test the methods w.r.t. three different T-norms (i.e., Gödel, Łukasiewicz, and Product, described in Table 6.1) using 10%, 20%, 50%, 75% and 100% of the available annotated ROAD-R data, and training for 110, 70, 45, 30, and 30 epochs, respectively. I use a learning rate of 0.0041 for all models, but for 100% labelled data, I drop it at epochs 18 and 25 by a factor of 10, as in (Giunchiglia et al., 2023b). Note that I always compute the logic-based losses w.r.t. all of the 243 constraints from ROAD-R. Finally, for evaluation, I use the frame-wise mean average precision (f-mAP) metric and report its result at the best training epoch. The f-mAP is a standard metric used in autonomous driving scenarios and it is computed by computing the average precision at a fixed intersection-over-union (IoU) threshold (often set to 0.5) over each frame for each class and then taking the mean of these per-class scores, as follows:

$$\text{f-mAP} = \frac{1}{|\mathcal{X}|} \sum_{(\mathbf{X},\mathcal{Y})\in\mathcal{X}} \Big( \frac{1}{|\mathcal{L}_{\mathbf{X}}|} \sum_{L\in\mathcal{L}_{\mathbf{X}}} \text{AP}_{0.5}(\mathbf{X}, L) \Big), \tag{6.2}$$

where $\mathcal{L}_{\mathbf{X}}$ represents the set of classes that occur in frame $\mathbf{X}$ (and which is assumed to be non-empty), and $\text{AP}_{0.5}(\mathbf{X}, L)$ represents the average precision (i.e., the area under the precision-recall curve) computed on frame $\mathbf{X}$ at class $L$, using an intersection-over-union threshold of 0.5. More specifically, $\text{AP}_{0.5}(\mathbf{X}, L)$ is computed by first ranking all the predicted bounding boxes by their confidence scores. The predictions are then matched to the ground truth bounding boxes using the IoU threshold of 0.5, such that a prediction is considered true positive if it has IoU of at least 0.5 with an unmatched ground truth object, and false positive otherwise. Finally, the precision and recall are computed at each rank position in the sorted list of predictions, to ultimately compute the average precision.

The results can be found in Table 6.3, which shows that the logic-based losses always improve the baseline performance, except when using 100% labelled data, where only the Łukasiewicz T-norm outperforms the baseline. Moreover, the Gödel and Łukasiewicz T-norms bring non-negligible improvements. Lastly, as expected, integrating background knowledge via the logic-based loss in neural network models

Table 6.3: Comparison between the constrained models with different logic-based losses and the baseline unconstrained models when varying the percentage of labelled data, as indicated on top of each column.

| | 10% | 20% | 50% | 75% | 100% |
|---|---|---|---|---|---|
| Baseline | 24.49 | 26.81 | 31.56 | 33.57 | 33.39 |
| Gödel | **26.34** (+1.85) | **30.76** (+3.95) | 32.71 (+1.15) | **34.39** (+0.82) | 32.16 (−1.23) |
| Łukasiewicz | 26.24 (+1.75) | 29.13 (+2.32) | **34.07** (+2.51) | 33.89 (+0.32) | **34.42** (+1.03) |
| Product | 24.53 (+0.04) | 26.96 (+0.15) | 31.66 (+0.10) | 34.06 (+0.49) | 33.34 (−0.05) |

Table 6.4: The **best**- and worst-performing models across fully-supervised models and models using unlabelled data, with or without warm-up. All models here used 10% labelled data for training. The models in the last two columns used also 10% unlabelled data during the training phase.

| | Fully-supervised | With unlabelled data | |
|---|---|---|---|
| | - | - | Warm-up |
| Gödel | 26.34 | 26.38 | **26.76** |
| Łukasiewicz | 26.24 | 25.48 | **26.75** |
| Product | 24.53 | 25.79 | **27.24** |

is more helpful when little data are available. Indeed, the proposed approach yields an improvement of up to 1.85% and 3.95% when using, 10% and 20% of the labelled training data, respectively.

This last observation led to an investigation into whether the known result—that background knowledge helps when unlabelled data are available (e.g., (Stewart and Ermon, 2017; Marra et al., 2019))—also holds in this setting. To this end, I trained models where I applied the logic-based losses on 10% labelled and 10% unlabelled data. As shown in the second column of Table 6.4, neither the Gödel nor the Łukasiewicz T-norm were particularly helpful relative to their fully-supervised performance. Surprisingly, the Product T-norm improved its performance instead, now surpassing the baseline. Since the added unlabelled data were not helpful in two out of three cases, I analysed the losses at the beginning of the training and hypothesised that it would be beneficial to introduce a warm-up training phase, during which the logic-based loss would be inactive, and after which the unlabelled data would be added and the logic-based loss activated. As expected, using this strategy, the results from the last column of Table 6.4 show a consistent improvement over the previous performances of all logic-based losses, with the Product T-norm giving the highest result (of 27.24 f-mAP) w.r.t. the baseline (of 24.49 f-mAP).

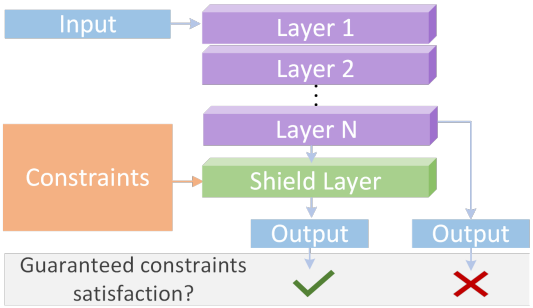



Figure 6.3: Visualisation of the Shield Layer integrated into a given DNN.

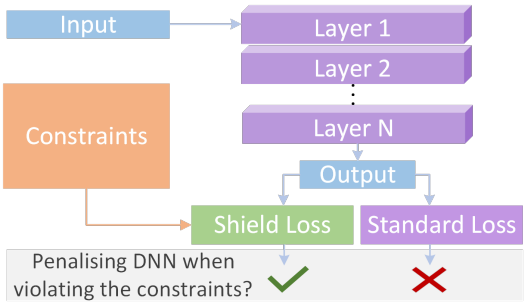



Figure 6.4: Visualisation of the Shield Loss for a given DNN.

## 6.3 PiShield

PiShield is built on top of PyTorch, allowing for a seamless integration of Shield Layers and Losses into neural networks.

The Shield Layers can be applied during inference and/or training, depending on the practitioners' needs. For users that have restricted access to a model but require that the outputs of their models are compliant with a set of rules, PiShield offers an easy-to-use interface to meet this need. To achieve this, it is enough to simply build a Shield Layer outside the neural network and apply it on the outputs to make them compliant with the constraints. Alternatively, Shield Layers can be used to constrain deep neural networks during training, suiting also practitioners accustomed to modifying their models. To this end, the architecture of the neural network needs to be changed by appending a Shield Layer immediately after the output layer, as shown in Figure 6.3.

On the other hand, the Shield Losses are designed to aid model learning by using logic-based loss functions to penalise outputs which are not compliant with the background knowledge. Indeed, they can be an option when the underlying models' architecture cannot be modified and only the existing weights can be updated based on signals from the constraints, as shown in Figure 6.4.

Regardless of its usage, any Shield Layer (and any Shield Loss) requires only two elements to be instantiated: (i) the dimension of the input to the layer (which typically corresponds to the dimension of the output of the network), and (ii) the path to the file which contains the constraints.

**Constraints.** The constraints can be either: (i) a set of disjunctive inequalities (e.g., QFLRA formulae) over a set of variables (e.g., capturing features in tabular

data generation tasks), or (ii) a set of propositional logic formulae over a set of atoms (e.g., capturing output classes in classification tasks).

Note that the first types of constraints is supported by the Shield Layer (e.g., for generative modelling), while the second type of constraints is supported by both the Shield Layer and Shield Loss (e.g., for multi-label classification). Regardless of the type, each line in the constraints file should contain a single constraint, which will thus be either: (i) a single disjunctive inequality (notice that this can also simply be a linear inequality), or (ii) a propositional logic formula in disjunctive normal form. Examples 6.3.1 and 6.3.2 illustrate scenarios for each constraints type and how to specify such constraints in PiShield.

**Example 6.3.1.** *Consider a tabular data generation problem, where the goal is to generate a synthetic dataset for a clinical trial. Additionally, suppose the following knowledge about the problem is available: (i) the maximum haemoglobin recorded per patient should be always higher or equal than the minimum, (ii) the maximum temperature recorded should be at least as high as the minimum, and (iii) if the minimum haemoglobin level recorded for a patient is of at least 2.1 (using standard metrics) and the minimum temperature is of at least 29 degrees Celsius, then the number of tests conducted since the start of the trial is at least 2. Assuming that the outputs of the neural network are ordered as MaxHaemoglobin, MinHaemoglobin, MaxTemp, MinTemp, NumTests, the input file should have the following format:*

$$y\_0 - y\_1 >= 0$$
$$y\_2 - y\_3 >= 0$$
$$\text{neg } y\_1 >= 2.1 \text{ or neg } y\_3 >= 29 \text{ or } y\_4 >= 2$$

Note that the package expects the linear inequalities to contain either $\geq$ or $>$. This is only a notational convention which does not restrict the constraints' expressivity, as any inequality written using $\leq$ or $<$ can be transformed into one of the forms above.

**Example 6.3.2.** *Consider a simple multi-label classification problem where the inputs are images taken from an autonomous vehicle, and the goal is to identify whether a traffic light appears in the image, and whether it is green, yellow, or red. For each image, a standard neural network for this task will output a 4-dimensional vector where each element corresponds to the confidence score assigned to each of the labels, which are ordered (for example) in the following way:* TrafficLight, Red, Yellow, Green. *Then the file, containing the knowledge that a traffic light is always associated*

Listing 1: Correcting predictions with a Shield Layer at inference time.

```
from pishield.shield_layer import build_shield_layer

def correct_predictions(predictions, constraints_path):
    num_variables = predictions.shape[-1]
    shield_layer = build_shield_layer(num_variables, constraints_path)
    corrected_predictions = shield_layer(predictions)
    return corrected_predictions
```

Listing 2: Building a Shield Layer into a deep neural network (DNN) to train with it.

```
from pishield.shield_layer import build_shield_layer

class DNN_with_Shield_Layer(torch.nn.Module):
    def __init__(self, num_dim, constraints_path, ..):
        self.model = torch.nn.Sequential(...)
        self.shield_layer = build_shield_layer(num_dim, constraints_path)
        ...

    def forward(self, input):
        output = self.model(input)
        corrected_output = self.shield_layer(output)
        return corrected_output
```

*with one of the colours and that the colours associated with a traffic light are mutually exclusive, should have the format below:*

$$\begin{aligned}
&\textit{not y_0 or y_1 or y_2 or y_3}\\
&\textit{not y_0 or not y_1 or not y_2}\\
&\textit{not y_0 or not y_1 or not y_3}\\
&\textit{not y_0 or not y_2 or not y_3}
\end{aligned}$$

**Usage.** As shown in Listings 1 and 2 (resp., Listing 3), only one call to the function `build_shield_layer` (resp., `build_shield_loss`) is needed to build a Shield Layer (resp., Shield Loss). More precisely, this function takes as input two parameters: (i) the dimension of the input to the Shield Layer (resp., Shield Loss), and (ii) the path to the constraints file. Optionally, a pre-defined ordering of the variables can be passed as

Listing 3: Adding a Shield Loss into a deep neural network (DNN) to train with it.

```
1  from pishield.shield_loss import build_shield_loss
2
3  class DNN_with_Shield_Loss(torch.nn.Module):
4      def __init__(self, num_dim, constraints_path, ..):
5          self.model = torch.nn.Sequential(...)
6          self.shield_loss = build_shield_loss(num_dim, constraints_path)
7          ...
8
9      def training_step(self, input, labels):
10         output = self.model(input)
11         logic_loss = self.shield_loss(output)
12         loss = self.model_loss(output, labels) + logic_loss
13         loss.backward()
```

input to the Shield Layer, and a chosen T-norm (either Gödel, Product or Łukasiewicz T-norm, which are supported in PiShield) can be specified for the Shield Loss.

Once instantiated, the Shield Layer receives as input a tensor of predictions $p$ (possibly violating the constraints) and returns a tensor $\hat{p}$ (of the same dimension as $p$), which is now guaranteed to be compliant with the constraints. Similarly, the Shield Loss receives as input $p$ and returns the value of the underlying logic-based loss, as described in Equation 6.1 of Section 6.2. The value returned by the Shield Loss should be added to the standard loss used by the model, as shown in Listing 3 (line 12).

PiShield benefits from an easy-to-use interface. Thus, correcting $p$ with a Shield Layer is a one-step operation consisting of a forward call of the Shield Layer on $p$. Listing 1 shows how to do all these steps at inference time and Listing 2 shows how to integrate the layer at training time for which the layer's forward call needs to be done before the backpropagation step. As shown in the latter Listing, the easiest way to match this condition is to instantiate the Shield Layer in the constructor of the deep neural network and then call the layer inside the usual forward method (as shown in line 11). Similarly, adding the memory-efficient logic-based loss to the standard loss function used on $p$ is also a one-step operation consisting of a forward call of the Shield Loss on $p$.

**Feature ordering.** As an additional input to the Shield Layer, the user has the option to provide an ordering: either as a separate argument to the layer or within the constraints file. As discussed in Chapter 4, the feature orderings can affect the

| Scenario | Baseline | PiShield | Shield type |
|---|---|---|---|
| Tabular data generation, linear constraints | 0.430 | **0.458** | Layer |
| Tabular data generation, QFLRA constraints | 0.450 | **0.503** | Layer |
| Autonomous driving, fully-supervised | 0.288 | **0.303** | Layer |
| Autonomous driving, 10% labelled & unlabelled data | 0.245 | **0.272** | Loss |

Table 6.5: Aggregated performance of PiShield and baselines across different scenarios. The results for tabular data generation and autonomous driving tasks are reported in F1-score for efficacy and f-mAP, respectively. The best results are in **bold**.

performance of the model. Depending on the scenario, different ordering might be more suitable. For example, in a tabular data generation task where there are known causal relations between the features, an intuitive approach would be to provide an ordering that respects the topological order of the underlying directed acyclic graph capturing the causal relations. Alternatively, the user can set the ordering to be randomly generated.

## 6.4 Example Scenarios

The need for a package like PiShield naturally raises in many application domains. Here, I discuss different example scenarios where I have applied it and compare the performance between the baseline unconstrained models and the models equipped with PiShield. Nevertheless, its applications extend to performing checks on the outputs of the models mentioned above, and can also help make tasks more systematic, for example by helping with creating new datasets by checking annotations.

**Tabular Data Generation.** Applying Shield Layers to standard deep generative models (DGMs) for synthesising tabular data produces more realistic outputs that are compliant with the background knowledge constraints, such as linear or disjunctive inequalities. As demonstrated in Chapters 3 and 5, this approach significantly improves machine learning efficacy, a standard benchmark for tabular data generation. Table 6.5 highlights this effect, comparing the F1-score of baseline models against their constrained counterparts. Specifically, the table presents performance averaged over five DGM types for two distinct scenarios: (i) the first row shows results from Chapter 3, where models were constrained with up to 31 linear inequalities across five classification datasets, and (ii) the second row shows results from Chapter 5, where models were constrained with up to 18 disjunctive inequalities across four clas-

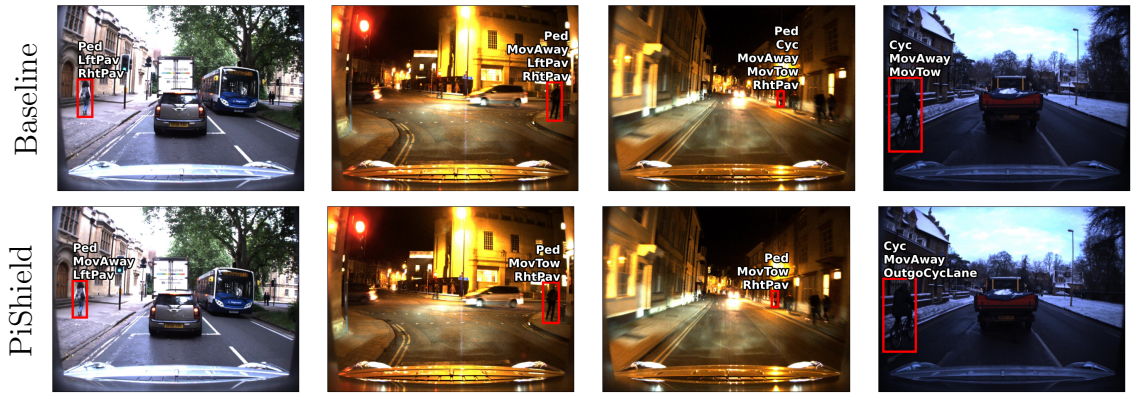



Figure 6.5: PiShield case scenario: autonomous driving. The unconstrained models (top row) violate simple background knowledge rules, predicting (i) that a pedestrian is both on the right and on the left pavement, and (ii) that a person is both a pedestrian and a cyclist, and is moving towards and away from the self-driving vehicle at the same time. On the other hand, the predictions made using PiShield (bottom row) do not violate the background knowledge.

sification datasets. In both scenarios, using the constraints during training provides significant improvements over the baselines.

**Road Events Detection.** For tasks like road event detection, where predictions must obey strict constraints, PiShield can integrate domain knowledge into the neural network models in two ways: (i) as a Shield Loss to guide model training or (ii) as a Shield Layer to guarantee compliant outputs. To this end, consider the scenario I presented in Section 6.2, specifically: the task of multi-label classification using the ROAD-R autonomous driving dataset (Giunchiglia et al., 2023b), which is annotated with 243 propositional logic constraints. The final two rows of Table 6.5 summarise the results, showing that both approaches improve predictive performance as measured by frame-wise mean average precision (f-mAP). Specifically, the third row details these gains[5] in a fully-supervised setting, while the fourth row shows the results in a challenging low-resource setting, where the model was trained with only 10% of the labelled data, supplemented by an additional 10% of unlabelled data used (along with the labelled data) only for the Shield Loss after a warm-up phase. Finally, in Figure 6.5 I provide visual examples of how enforcing these constraints improves prediction quality, which is intrinsically coupled with the safety guarantees.

[5]I showed this result as part of my contribution in (Giunchiglia et al., 2024), a paper I have co-authored, but which I have not included in this thesis. Table 2 of this paper shows the full results over five different models, from which I obtain the average f-mAP performance reported here.

Ultimately, these experiments demonstrate that PiShield is a powerful tool for safety-critical applications. It is capable of ensuring that predictions adhere to crucial domain knowledge during training and inference, but also of leveraging those constraints to guide the learning process towards more accurate and reliable models.

## 6.5 Related Work

**Neuro-symbolic AI methods.** Neuro-symbolic works have proposed ways to embed available background knowledge by either embedding it into the topology and/or into the loss. In the former category, directly relating to the Linearly-constrained Layer and the Disjunctive Refinement Layer I introduced in Chapters 3 and 5, respectively, we find works that build a constrained layer on top of a neural network, such as Coherent-by-Construction Network (CCN) (Giunchiglia and Lukasiewicz, 2021), MultiplexNet (Hoernle et al., 2022), and Semantic Probabilistic Layer (SPL) (Ahmed et al., 2022b), all of these approaches being able to guarantee the satisfaction of hard constraints under certain conditions. Having a similar goal, NESTER (Dragone et al., 2021) proposes another end-to-end approach by imposing soft and hard constraints via a program applied on the outputs. Yet another recent work is Iterative Local Refinement (ILR) (Daniele et al., 2023b), which proposed an analytic way of integrating T-norm-based functions as neural network layers to refine the predictions in a differentiable manner.

The other main line of work comprises methods that relax the constraints and integrate them into the neural networks' loss, directly relating to the memory-efficient logic-based loss I introduced in Section 6.2. Early work on semantic based regularisation (SBR) (Diligenti et al., 2012) on kernel machines led to the development of ways to map the constraints into the neural networks' loss according to the T-norm operations (Diligenti et al., 2017b; Marra et al., 2019; Serafini and d'Avila Garcez, 2016; Badreddine et al., 2022). However, among other issues highlighted in (van Krieken et al., 2020, 2022), one problem occurring in these approaches is that they are syntax-dependent. To address this, Semantic Loss (Xu et al., 2018) and DL2 (Fischer et al., 2019) introduced syntax-independent loss functions. Another work, by Ahmed et al. (2022c), integrates logic into the standard entropy regularisation (Grandvalet and Bengio, 2004) term of the loss. The recent work by Li et al. (2023) explores another problem with the previous approaches, namely, that models tend to settle on the easier solutions that satisfy the constraints, and proposes a way to enforce the model

to fully explore the available knowledge. While many such works proved to be particularly helpful when little annotated data are available (Li and Srikumar, 2019; Fischer et al., 2019; Marra et al., 2019; Hu et al., 2016a,b; Stewart and Ermon, 2017), they have been designed for small and/or synthetic datasets and would not scale to complex scenarios. For a complete survey on how to incorporate logical constraints in deep learning, see, e.g., (Giunchiglia et al., 2022).

**Packages.** A closely related work is Pylon (Ahmed et al., 2022a), a framework built on PyTorch which allows users to integrate constraints into a loss function. Similarly, the LTN package (Badreddine et al., 2022) provides a TensorFlow implementation of the Logic Tensor Networks (LTNs), while LTNTorch (Carraro, 2022) provides its PyTorch implementation. However, unlike PiShield, all the methods described above cannot guarantee the satisfaction of the constraints.

## 6.6 Conclusions

In this Chapter, I introduced PiShield, a package designed to bridge the gap between research and practice in constraint-based machine learning. It provides two core mechanisms for integrating background knowledge into neural networks: (i) Shield Layers, which are built into a model's architecture to guarantee that all outputs comply with a given set of rules, and (ii) Shield Losses, which are memory-efficient T-norm-based loss functions that guide the learning process, proving effective even in resource-intensive scenarios like autonomous driving.

Built on PyTorch, PiShield is a versatile tool that supports multiple constraint types and is designed for easy integration with state-of-the-art models across diverse tasks, from tabular data synthesis to road event detection.

# Chapter 7

# Conclusions

In this thesis, I investigated how to integrate background knowledge constraints into deep neural networks in order to refine their outputs so that they are compliant with the constraints. In the context of this work, *background knowledge* refers to: (i) a set of propositional logic formulae written over a set of labels, (ii) a set of linear inequalities, or (iii) a set of disjunctions over linear inequalities, which capture the inherent relations between data features. Focusing on two real-world applications, I have developed, implemented, and evaluated methods that belong to two primary neuro-symbolic approaches: layer-based methods that guarantee constraint satisfaction for deep generative models in the context of tabular data synthesis, and loss-based methods that guide the learning process for multi-label classifiers in the context of autonomous driving. Further, to evaluate my proposed methods, I utilised datasets which span a wide range of complexities to accurately reflect real-world challenges. For tabular data, this includes binary and multi-class classification datasets, regression datasets, data-scarce settings, large-scale data, and background knowledge constraints with varied expressivity. Similarly, in the context of autonomous driving, I evaluate my approach on a highly complex dataset annotated with an extensive set of constraints. The work presented demonstrates that integrating background knowledge constraints and, thus, creating *knowledge-refined* neural networks, not only prevents models from making predictions which are inconsistent with the constraints but can also significantly improve their performance and reliability.

I presented the contributions of this thesis across four core Chapters, as summarised below. In Chapter 3, I introduced the Linearly-constrained Layer (LL), the first framework to directly incorporate background knowledge constraints expressed as linear inequalities into the architecture of Deep Generative Models (DGMs). Able to automatically parse such user-defined constraints, LL was designed as a differentiable layer that guarantees all generated samples satisfy the given constraints.

Further, through an in-depth experimental analysis, I showed that integrating constraints via LL into DGMs provides significant overall improvements in the quality of synthetic data, measured by machine learning efficacy and detection, all without incurring additional runtime costs at inference.

In Chapter 4, I proposed and evaluated five different methods for providing an order in which the features' values are corrected by LL. As different variable orderings may lead to different LL outputs, a core question of this Chapter was whether varying the ordering affects the sample quality. The extensive experimental analysis revealed that while no single ordering is universally optimal, there are reliable ordering choices that consistently improve the quality of the generated data over the unconstrained models.

Two future work directions could be (i) to further investigate these patterns, aiming to uncover why certain models and datasets exhibit preferences for specific orderings, and (ii) to ultimately design a method that automatically learns the feature ordering, which could further boost the improvements already provided by LL.

In order to address the limitations of linear inequalities, in Chapter 5, I proposed the Disjunctive Refinement Layer (DRL), which is a novel layer that extends the core idea of LL to handle far more expressive constraints in the form of quantifier-free linear real arithmetic (QFLRA) formulae. This allows for the enforcement of constraints that define non-convex and even disconnected output spaces, which are common in real-world data. Designed to be compatible with any deep generative architectures (e.g., GANs, VAEs, graph-based models etc.) and applicable across various downstream tasks (e.g., classification and regression), DRL is a differentiable layer that guarantees constraint satisfaction while finding an optimal correction to the original sample. Further, through an in-depth experimental analysis, I showed that DRL consistently improves the machine learning efficacy of the underlying generative models, while also providing inference runtimes comparable to the unconstrained models.

Importantly, through extensive evaluations conducted across Chapters 3 to 5, I show that there is no trade-off between ensuring that the samples generated using my methods satisfy the background knowledge and the sample quality. In fact, the resulting models (i.e., models equipped with LL or DRL) significantly improve sample quality, as measured by machine learning efficacy. Moreover, these approaches are superior to the standard technique of rejection sampling, which is slower, can fail completely when violation rates are high, and often degrades data quality. The experiments also validate the strong need for these new methods, showing that existing

generative models frequently violate constraints, with over half the data being invalid in many real-world scenarios.

The development and application of synthetic data generation techniques, particularly in tabular data, have the potential to significantly impact a wide range of sectors, including healthcare, finance, and social sciences. While the LL and DRL frameworks that I proposed for synthesising tabular data improve the quality of generated data by ensuring alignment with user-specified constraints, there are ethical implications of synthetic data use. Firstly, there is the potential for misuse. Synthetic data may be seen as a substitute for real-world data, but it should not be viewed as a perfect replacement. Secondly, the use of synthetic data in automated decision-making systems poses risks for fairness and bias. While both LL and DRL allow for the specification of constraints that align with real-world domain knowledge, it is important to ensure that these user-specified constraints do not embed existing biases or discrimination. Provided this condition is met, injecting domain knowledge into generative models does not pose any direct negative impact. Regarding privacy, when the constraints are derived from public domain knowledge (e.g., physical laws or general business rules), my proposed methods (LL and DRL) preserve the privacy guarantees of the underlying base model. However, additional care is required if constraints are derived directly from sensitive datasets, as this could introduce new leakage channels not present in the unconstrained model.

As future work, one possible direction is to further expand the expressivity of the background knowledge constraints that can be injected into DGMs, while ensuring that they are satisfied during training and inference. Secondly, another level of automatisation could be explored in the future. For instance, it would be interesting to investigate whether combining a module that automatically discovers constraints between the features would help the layer-based methods presented in this thesis.

Finally, in Chapter 6, I proposed PiShield, a PyTorch-based package designed for seamlessly integrating background knowledge into neural networks via layer-based and loss-based methods. For the former type of methods, the package provides Shield Layers, which are implementations of the layer-based methods (i.e., the LL and DRL layers) from previous Chapters. For the latter type of methods, this Chapter expanded the scope of the thesis to the second category of neuro-symbolic AI by contributing a novel, memory-efficient, logic-based loss function, which I refer to as the Shield Loss. Empirically, I demonstrated that this loss function is highly effective for guiding the training of multi-label classifiers in complex settings with a large number of constraints, such as autonomous driving, especially in data-scarce scenarios.

For Shield Losses, as future work, a possible research direction is conducting a study into how the loss can help in a semi-supervised scenario, with varying amounts of unlabelled data added during the training and using the warm-up phase that I found helpful for reducing the runtime strain brought by integrating background knowledge into neural networks. Another research direction could be a study on the effect of using different intervals of warm-up, before introducing the T-norm loss and applying it on the unlabelled data.

I encapsulated the methods developed in this thesis in PiShield to make them accessible to researchers and practitioners, thus, creating a versatile tool for applying neuro-symbolic approaches—whether layer- or loss-based—to practical problems. Based on the theoretical and empirical evidence presented in this thesis, I offer the following recommendations for practitioners regarding the selection of the proposed methods. The decision process should prioritise the type of constraint enforcement required (soft vs. hard) and the complexity of the domain knowledge. This choice corresponds to the distinction between Shield Losses and Shield Layers. If the primary goal is to guide the training of the model using the background knowledge (expressed as propositional logic) without requiring strict guarantees for its satisfaction, then Shield Losses should be used, as shown in Chapter 6 in the context of autonomous driving. Conversely, Shield Layers should be used when mathematical guarantees for constraint satisfaction are required, as shown in Chapters 3-5 in the context of tabular data synthesis.

If a Shield Layer is selected, it can be deployed in two ways: it can be either integrated into a deep generative model at training time (e.g., if practitioners have access to the underlying model), or it can simply be used as a post-processing step at inference time (e.g., if the goal is solely compliance with background knowledge, without requiring the performance gains associated with the LL integration at training time). Further, the specific Shield Layer variant depends on the complexity of the constraints. Specifically, if the background knowledge consists of linear inequalities capturing the relations between the tabular data features, then the Linearly-constrained Layer (LL) should be used. If LL is used during training, it is crucial to note that the ordering of features is an important factor influencing downstream performance and should be chosen based on the underlying model architecture and dataset characteristics, as discussed in Chapter 4. Finally, if the background knowledge involves non-convex constraints that can be captured using QFLRA formulae (including disjunctions over linear inequalities), then the Disjunctive Refinement Layer (DRL) should be used.

While a significant portion of this thesis focuses on synthetic tabular data, the ability to inherently incorporate constraints into neural networks has implications beyond generative modelling. Purely data-driven models are prone to producing physically impossible, unsafe, or logically inconsistent outputs when faced with out-of-distribution inputs. Incorporating constraints (whether through Shield Layers or Shield Losses) acts as a powerful inductive bias. This is especially important for predictive and decision-making systems in safety-critical domains. For instance, in healthcare, predictive models must obey biological constraints (e.g., predicting drug dosages that do not exceed dangerous limits); in physics-informed machine learning, models must conserve mass and energy; and in robotics or autonomous driving, reinforcement learning agents must strictly adhere to safety requirements and traffic laws. Thus, the knowledge-refined neural networks presented in this thesis serve as a step towards generally safe, reliable, and trustworthy AI.

As I have shown, the proposed methods in this thesis have the capability to improve AI systems by incorporating constraints over their purely data-driven counterparts. Moreover, learning from domain knowledge brings little annotation overhead, as constraints are often easier to obtain than large annotated datasets. For these reasons, my belief is that the interest in neuro-symbolic AI will continue to grow, further paving the way for the creation of more reliable and trustworthy systems.

# Appendix A

# Hyperparameter Search for Chapter 3

For reproducibility, I present the hyperparameter search I conducted as part of the experimental analysis in Section 3.4 of Chapter 3 in order to ensure that the comparisons between the standard DGMs and the DGM+LL models are meaningful. I always chose the best settings according to the machine learning efficacy performance measured either (i) by the average over the F1-score, weighted F1-score, and Area Under the ROC Curve for the binary and multi-class classification datasets or (ii) by the Mean Absolute Error for the regression dataset.

For GOGGLE, I used the same optimiser and learning rate set as Liu et al. (2022). Specifically, I used Adam (Kingma and Ba, 2015) with a set of three different learning rates: $\{1\times10^{-3}, 5\times10^{-3}, 1\times10^{-2}\}$. For TVAE, I used Adam with a set of five different learning rates: $\{5 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$. And for each of the other DGM models, I used three different optimisers, Adam, RMSProp (Hinton, 2014), SGD (Ruder, 2016), each with a different set of learning rates:

- for WGAN $\{1\times10^{-4}, 1\times10^{-3}\}$, $\{5\times10^{-5}, 1\times10^{-4}, 1\times10^{-3}\}$, and $\{1\times10^{-4}, 1\times 10^{-3}\}$, respectively.
- for TableGAN, $\{5\times10^{-5}, 1\times10^{-4}, 2\times10^{-4}, 1\times10^{-3}\}$, $\{1\times10^{-4}, 2\times10^{-4}, 1\times10^{-3}\}$ and $\{1 \times 10^{-4}, 1 \times 10^{-3}\}$, respectively.
- for CTGAN, $\{5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}\}$, $\{1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$ and $\{1 \times 10^{-4}, 1 \times 10^{-3}\}$, respectively.

Then, for each of the above optimiser-learning rate pairs, I tested different batch sizes, depending on the DGM model: $\{64, 128, 256\}$ for WGAN, $\{128, 256, 512\}$ for TableGAN, $\{70, 280, 500\}$ for CTGAN and TVAE, and $\{64, 128\}$ for GOGGLE. The

Table A.1: Best hyperparameter configurations used for the DGMs (and DGM+LL models) in the experiments.

| Model/Dataset | Hyperparameter | URL | WiDS | LCLD | Heloc | FSP | News |
|---|---|---|---|---|---|---|---|
| WGAN | Batch size | 64 | 128 | 128 | 64 | 128 | 128 |
| | Optimiser | Adam | RMSProp | SGD | Adam | RMSProp | RMSProp |
| | Learning rate | 0.0001 | 0.001 | 0.001 | 0.0001 | 0.001 | 0.001 |
| | Epochs | 300 | 80 | 30 | 200 | 20 | 50 |
| | Discriminator iters | 10 | 10 | 5 | 10 | 10 | 10 |
| | Ordering | Rnd | KDE | KDE | KDE | KDE | KDE |
| TableGAN | Batch size | 128 | 128 | 128 | 128 | 128 | 128 |
| | Optimiser | Adam | RMSProp | Adam | Adam | Adam | RMSProp |
| | Learning rate | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| | Epochs | 300 | 50 | 25 | 200 | 200 | 50 |
| | Ordering | Corr | Corr | KDE | Corr | KDE | KDE |
| CTGAN | Batch size | 500 | 500 | 500 | 500 | 500 | 500 |
| | Optimiser | Adam | RMSProp | Adam | Adam | Adam | Adam |
| | Learning rate | 0.0002 | 0.0005 | 0.0002 | 0.0002 | 0.0002 | 0.0002 |
| | Epochs | 150 | 50 | 20 | 500 | 300 | 100 |
| | Ordering | KDE | Corr | Rnd | Corr | Corr | Corr |
| TVAE | Batch size | 70 | 70 | 500 | 500 | 70 | 70 |
| | Optimiser | Adam | Adam | Adam | Adam | Adam | Adam |
| | Learning rate | 0.0002 | 0.000005 | 0.00001 | 0.000005 | 0.00001 | 0.00001 |
| | Epochs | 150 | 50 | 40 | 150 | 500 | 50 |
| | Loss factor | 2 | 2 | 4 | 2 | 1 | 3 |
| | Ordering | KDE | KDE | Corr | Corr | Corr | Corr |
| GOGGLE | Batch size | 128 | 128 | 128 | 64 | 128 | 64 |
| | Optimiser | Adam | Adam | Adam | Adam | Adam | Adam |
| | Learning rate | 0.005 | 0.01 | 0.001 | 0.001 | 0.005 | 0.001 |
| | Epochs | 1000 | 200 | 200 | 1000 | 1000 | 250 |
| | Patience | 50 | 50 | 50 | 50 | 50 | 50 |
| | Ordering | Random | Corr | Corr | Random | Corr | KDE |

batch sizes for CTGAN are multiples of 10, to allow for using the CTGAN's recommended PAC (Lin et al., 2018) value of 10, among other values.

The initial phase of the hyperparameter search allowed to narrow down the space of configurations and focus on further investigating the most promising one. For the second phase, I varied the following parameters for each DGM, keeping fixed the rest from the best model of the initial phase:

- for WGAN, I varied initially PAC values within $\{4, 8, 16\}$ and then discriminator iterations within $\{1, 2, 10\}$.
- for TableGAN, I changed separately the generator layers' dimensions to 128 (from 100 initially) and the embedding dimensions by doubling the value.

- for CTGAN, I varied the value of PAC within $\{1, 5, 15\}$.
- for TVAE, I varied the multiplier of the loss within $\{1, 2, 3, 4\}$.

In Table A.1, I provide the hyperparameter configurations determined after the search conducted. Note that for the DGM+LL models I report the best ordering among random, correlation-based and KDE-based methods (detailed in Chapter 4).

# Appendix B

# Linearly-constrained Layer: Full Results

## B.1 Context of Collaboration

The experimental analysis in (Stoian et al., 2024a) and (Stoian et al., 2025), both of which are included in my thesis, is the result of a joint research effort where the experimental work was divided equally between my collaborator, Salijona Dyrmishi, and me. Some of these results are presented in Sections 3.4 and 4.3, and Appendix B. Below I list the individual contributions that led to these results:

- I designed and implemented the Linearly-constrained Layer.
- I designed and implemented all feature orderings methods for the Linearly-constrained Layer.
- Equally-split experimental analysis (including Table F.1): I conducted the experiments using WGAN, GOGGLE, and half of the TableGAN experiments, and Salijona Dyrmishi conducted the experiments using CTGAN, TVAE, and the other half of the TableGAN experiments.
- Salijona Dyrmishi converted the data transforms into differentiable operations for the CTGAN and TVAE models.

## B.2 DGMs vs. DGMs+LL

In Tables B.1-B.7, I present the full machine learning efficacy and detection results. More specifically, in each table I report results for each dataset, each DGM and each DGM+LL. Additionally, for each result in the table, I report the standard deviation

from the mean. As a reminder, the results are obtained by running each experiment with 5 different seeds.

**Machine learning efficacy.** In Tables B.1-B.3, I present the machine learning efficacy results on all classification datasets using three metrics: F1-score, weighted F1-score, and Area Under the ROC Curve, one per table (in this order). Separately, in Table B.4, I report the performance of real and synthetic data for the News dataset (which is a regression dataset), using two different metrics: Explained Variance (XV) and Mean Absolute Error (MAE).

Below, I provide a discussion of the machine learning efficacy results for each DGM and its corresponding DMG+LL model:

1. Comparing WGAN with WGAN+LL shows that the only dataset for which no improvement in efficacy is registered w.r.t. any of the three metrics is LCLD. However, in this case, the gap between the WGAN and the best WGAN+LL (i.e., using the KDE-based ordering) is small for all the 3 different metrics: 0.8% and 0.6% for F1-score and Area Under the ROC Curve, respectively, with a tie for the weighted F1-score. In most cases, WGAN+LL brings significant machine learning efficacy improvements, e.g. 7.7%, 5% and 3.8% for Heloc in F1-score, weighted F1-score and Area Under the ROC Curve, respectively.

2. For TableGAN, the results show that TableGAN+LL outperforms the unconstrained models for all binary and multi-class classification datasets in terms of machine learning efficacy on all three metrics. The only dataset where this is not the case is the regression dataset, News. It is worth noticing that out of all the DGM+LL models, TableGAN+LL brings major improvements overall. For example, it outperforms the unconstrained model by at least 4.5% according to the F1-score in 4 datasets, with the highest improvement, of 7.5%, recorded for WiDS.

3. As opposed to TableGAN+LL, CTGAN+LL does not show the same trend, with most models giving either moderate improvements or performing close to the unconstrained CTGAN. However, it is still the case that most of the datasets (four out of six) show improvements over all three metrics (with the remaining two datasets showing improvements on two out of three metrics). Notably, CTGAN+LL yields a large improvement of 64.9 points in the mean absolute error for machine learning efficacy when training on the News dataset, which is the second highest improvement observed (i.e., after WGAN+LL).

4. Similarly, to the WGAN+LL case, there is only one dataset for which TVAE+LL does not improve the machine learning efficacy performance in any of the three metrics, namely the Heloc dataset. Nevertheless, the difference between the overall best TVAE+LL model (i.e. using the correlation-based ordering) and TVAE is small for all the 3 different metrics: 0.2%, 0.6%, 0.3% for F1-score, weighted F1-score and Area Under the ROC Curve, respectively. In most cases, TVAE+LL yields major machine learning efficacy improvements, e.g. 4.0%, 3.6% and 1.5% for WiDS in F1-score, weighted F1-score and Area Under the ROC Curve, respectively, and 2.6%, 2.9% and 1.4% for FSP in F1-score, weighted F1-score and Area Under the ROC Curve, respectively.

5. The GOGGLE+LL models outperform the standard GOGGLE model in five out of six datasets on all metrics, giving major improvements (with the largest improvement being of 17.1% in the F1-score for machine learning efficacy when training on the URL dataset). Only the FSP dataset does not show any improvements on any of the orderings when using the GOGGLE+LL version.

**Detection.** In Tables B.5-B.7, I present the detection results on all datasets using three metrics in the following order: F1-score, weighted F1-score and Area Under the ROC Curve. The results show that:

1. The best WGAN+LL models bring an improvement over five out of six datasets in all metrics. Further, WGAN+LL improves two out of three metrics for the remaining dataset (i.e., the URL dataset).

2. The best TableGAN+LL models yield improvements over the baseline across all three metrics for four datasets (i.e., WiDS, LCLD, FSP, and News).

3. The CTGAN+LL models show improvement in all six datasets, across all three metrics.

4. The best TVAE+LL models register improvements (or do not change the performance, as it happens in one case, i.e., for News) across all three metrics for three datasets (i.e., LCLD, Heloc, and News). Further, TVAE+LL improves two of the three metrics for the URL dataset.

5. Unlike the previous models, the best GOGGLE+LL models bring improvements in only half of the datasets according to at least one out of three metrics for three datasets. In particular, for WiDS and FSP, GOGGLE+LL yields improvements across all three metrics.

Table B.1: Binary (macro) F1-score machine learning efficacy results with their corresponding stddevs for binary (multiclass) classification datasets.

| | | URL | WiDS | LCLD | Heloc | FSP |
|---|---|---|---|---|---|---|
| WGAN | - | 0.756±0.030 | **0.329**±0.059 | **0.239**±0.026 | 0.634±0.099 | 0.357±0.020 |
| | Rnd | 0.792±0.021 | 0.303±0.075 | 0.196±0.026 | 0.643±0.060 | 0.334±0.028 |
| | Corr | 0.790±0.026 | 0.284±0.040 | 0.231±0.050 | 0.704±0.039 | **0.367**±0.027 |
| WGAN+LL | KDE | **0.801**±0.004 | 0.316±0.044 | 0.231±0.050 | **0.711**±0.030 | **0.367**±0.027 |
| | WD | 0.771±0.033 | 0.273±0.070 | 0.157±0.062 | 0.667±0.029 | 0.346±0.013 |
| | Caus | 0.786±0.026 | 0.275±0.072 | — | 0.702±0.030 | 0.346±0.013 |
| TableGAN | - | 0.562±0.051 | 0.171±0.037 | 0.123±0.041 | 0.593±0.058 | 0.199±0.044 |
| | Rnd | 0.535±0.096 | 0.213±0.027 | 0.149±0.058 | 0.635±0.073 | 0.199±0.031 |
| | Corr | **0.612**±0.111 | **0.246**±0.047 | 0.131±0.036 | 0.638±0.061 | 0.179±0.036 |
| TableGAN+LL | KDE | 0.577±0.074 | 0.208±0.043 | **0.174**±0.063 | 0.581±0.114 | 0.207±0.053 |
| | WD | 0.515±0.086 | 0.242±0.021 | 0.173±0.038 | 0.646±0.070 | **0.226**±0.036 |
| | Caus | 0.486±0.075 | 0.225±0.028 | **0.174**±0.063 | **0.664**±0.041 | 0.159±0.012 |
| CTGAN | - | 0.822±0.017 | 0.362±0.033 | 0.247±0.087 | 0.736±0.035 | 0.374±0.034 |
| | Rnd | 0.832±0.006 | 0.364±0.019 | 0.235±0.057 | **0.743**±0.010 | 0.371±0.006 |
| | Corr | 0.830±0.009 | 0.365±0.020 | 0.260±0.081 | 0.729±0.027 | 0.383±0.019 |
| CTGAN+LL | KDE | **0.836**±0.002 | 0.360±0.022 | **0.265**±0.040 | 0.736±0.010 | 0.381±0.030 |
| | WD | 0.831±0.006 | **0.368**±0.013 | 0.224±0.060 | 0.732±0.026 | **0.391**±0.018 |
| | Caus | 0.835±0.007 | 0.365±0.010 | 0.222±0.034 | 0.733±0.010 | 0.374±0.035 |
| TVAE | - | 0.810±0.008 | 0.282±0.029 | **0.185**±0.021 | **0.735**±0.010 | 0.473±0.016 |
| | Rnd | 0.824±0.004 | 0.248±0.038 | 0.142±0.018 | 0.720±0.014 | **0.501**±0.012 |
| | Corr | 0.826±0.007 | 0.305±0.021 | 0.158±0.011 | 0.733±0.017 | 0.496±0.018 |
| TVAE+LL | KDE | 0.824±0.004 | **0.321**±0.041 | 0.146±0.023 | 0.731±0.008 | 0.498±0.014 |
| | WD | **0.829**±0.003 | 0.297±0.033 | 0.150±0.042 | 0.731±0.010 | 0.430±0.027 |
| | Caus | 0.821±0.009 | 0.316±0.019 | 0.163±0.037 | 0.729±0.009 | 0.499±0.012 |
| GOGGLE | - | 0.622±0.094 | 0.189±0.038 | 0.163±0.119 | 0.596±0.072 | **0.152**±0.003 |
| | Rnd | 0.782±0.035 | 0.139±0.068 | 0.157±0.085 | **0.723**±0.018 | 0.117±0.008 |
| | Corr | **0.793**±0.013 | 0.185±0.108 | **0.219**±0.023 | 0.714±0.013 | 0.136±0.024 |
| GOGGLE+LL | KDE | 0.787±0.014 | 0.171±0.036 | 0.167±0.082 | 0.709±0.025 | 0.134±0.016 |
| | WD | 0.722±0.019 | **0.207**±0.133 | — | 0.706±0.020 | 0.142±0.013 |
| | Caus | 0.640±0.057 | 0.175±0.051 | — | 0.707±0.013 | 0.129±0.020 |

Table B.2: Weighted F1-score machine learning efficacy results with their corresponding stddevs for binary and multiclass classification datasets.

| | | URL | WiDS | LCLD | Heloc | FSP |
|---|---|---|---|---|---|---|
| WGAN | - | 0.764±0.018 | **0.381**±0.057 | **0.359**±0.019 | 0.599±0.050 | 0.339±0.021 |
| | Rnd | 0.782±0.025 | 0.360±0.068 | 0.340±0.020 | 0.585±0.079 | 0.317±0.029 |
| | Corr | 0.784±0.011 | 0.343±0.036 | **0.359**±0.037 | 0.648±0.022 | **0.350**±0.028 |
| WGAN+LL | KDE | **0.790**±0.007 | 0.372±0.040 | **0.359**±0.037 | **0.649**±0.011 | **0.350**±0.028 |
| | WD | 0.775±0.018 | 0.331±0.064 | 0.312±0.046 | 0.646±0.017 | 0.327±0.015 |
| | Caus | 0.784±0.022 | 0.334±0.067 | — | 0.643±0.039 | 0.327±0.015 |
| TableGAN | - | 0.659±0.035 | 0.240±0.034 | 0.286±0.029 | 0.615±0.030 | 0.199±0.043 |
| | Rnd | 0.644±0.061 | 0.279±0.025 | 0.307±0.043 | 0.618±0.042 | 0.198±0.030 |
| | Corr | **0.695**±0.071 | **0.309**±0.042 | 0.292±0.029 | 0.633±0.036 | 0.180±0.034 |
| TableGAN+LL | KDE | 0.670±0.052 | 0.274±0.039 | 0.314±0.029 | 0.596±0.048 | 0.209±0.051 |
| | WD | 0.627±0.052 | 0.305±0.019 | **0.322**±0.031 | **0.634**±0.032 | **0.224**±0.033 |
| | Caus | 0.615±0.047 | 0.289±0.026 | 0.314±0.029 | 0.629±0.025 | 0.160±0.012 |
| CTGAN | - | 0.799±0.033 | 0.405±0.034 | 0.379±0.061 | 0.675±0.015 | 0.372±0.034 |
| | Rnd | 0.819±0.008 | 0.408±0.019 | 0.369±0.044 | 0.684±0.005 | 0.369±0.007 |
| | Corr | 0.816±0.012 | 0.409±0.019 | 0.388±0.060 | 0.688±0.010 | 0.379±0.020 |
| CTGAN+LL | KDE | 0.820±0.008 | 0.403±0.023 | **0.392**±0.030 | **0.690**±0.010 | 0.381±0.032 |
| | WD | **0.821**±0.003 | **0.411**±0.013 | 0.361±0.041 | 0.682±0.010 | **0.387**±0.018 |
| | Caus | 0.816±0.019 | 0.409±0.010 | 0.356±0.026 | 0.688±0.009 | 0.372±0.033 |
| TVAE | - | 0.802±0.012 | 0.342±0.027 | **0.330**±0.016 | **0.696**±0.006 | 0.463±0.017 |
| | Rnd | 0.818±0.007 | 0.311±0.034 | 0.300±0.012 | 0.687±0.006 | **0.491**±0.013 |
| | Corr | 0.817±0.007 | 0.363±0.019 | 0.310±0.011 | 0.690±0.009 | 0.488±0.019 |
| TVAE+LL | KDE | 0.816±0.008 | **0.378**±0.038 | 0.304±0.016 | 0.694±0.003 | 0.489±0.014 |
| | WD | **0.821**±0.007 | 0.356±0.030 | 0.306±0.027 | 0.693±0.004 | 0.419±0.028 |
| | Caus | 0.814±0.013 | 0.373±0.018 | 0.313±0.027 | 0.691±0.002 | 0.490±0.014 |
| GOGGLE | - | 0.648±0.074 | 0.198±0.067 | 0.315±0.086 | 0.566±0.050 | **0.139**±0.002 |
| | Rnd | 0.742±0.032 | 0.210±0.061 | 0.314±0.065 | **0.663**±0.012 | 0.103±0.008 |
| | Corr | **0.752**±0.024 | 0.253±0.098 | **0.357**±0.020 | 0.637±0.023 | 0.121±0.024 |
| GOGGLE+LL | KDE | 0.749±0.029 | 0.239±0.032 | 0.318±0.060 | 0.656±0.014 | 0.119±0.017 |
| | WD | 0.675±0.046 | **0.273**±0.121 | — | 0.654±0.023 | 0.126±0.012 |
| | Caus | 0.668±0.031 | 0.244±0.046 | — | 0.637±0.033 | 0.113±0.021 |

Table B.3: Area under the ROC curve machine learning efficacy results with their corresponding stddevs for binary and multiclass classification datasets.

| | | URL | WiDS | LCLD | Heloc | FSP |
|---|---|---|---|---|---|---|
| WGAN | - | 0.839±0.016 | 0.775±0.038 | **0.618**±0.011 | 0.677±0.033 | **0.742**±0.007 |
| | Rnd | 0.860±0.018 | 0.797±0.028 | 0.617±0.009 | 0.672±0.037 | 0.728±0.010 |
| | Corr | **0.864**±0.011 | 0.796±0.017 | 0.612±0.016 | **0.715**±0.017 | **0.742**±0.022 |
| WGAN+LL | KDE | 0.858±0.011 | **0.815**±0.015 | 0.612±0.016 | **0.715**±0.011 | **0.742**±0.022 |
| | WD | 0.852±0.012 | 0.770±0.019 | 0.594±0.032 | 0.712±0.020 | 0.726±0.008 |
| | Caus | 0.856±0.014 | 0.775±0.024 | — | 0.705±0.024 | 0.726±0.008 |
| TableGAN | - | 0.843±0.020 | 0.740±0.021 | 0.587±0.027 | 0.707±0.007 | 0.642±0.026 |
| | Rnd | 0.853±0.018 | **0.777**±0.017 | 0.593±0.019 | 0.692±0.029 | 0.636±0.022 |
| | Corr | **0.868**±0.007 | 0.775±0.015 | 0.605±0.016 | 0.704±0.030 | 0.630±0.037 |
| TableGAN+LL | KDE | 0.865±0.010 | 0.767±0.016 | 0.587±0.017 | 0.676±0.037 | 0.635±0.028 |
| | WD | 0.830±0.019 | 0.770±0.013 | **0.608**±0.010 | **0.710**±0.020 | **0.651**±0.034 |
| | Caus | 0.844±0.019 | 0.770±0.011 | 0.587±0.017 | 0.691±0.035 | 0.615±0.024 |
| CTGAN | - | 0.859±0.040 | 0.835±0.012 | **0.651**±0.020 | 0.744±0.009 | 0.760±0.014 |
| | Rnd | **0.880**±0.004 | 0.836±0.003 | 0.627±0.028 | 0.749±0.008 | 0.757±0.008 |
| | Corr | 0.877±0.010 | 0.826±0.013 | 0.644±0.015 | **0.755**±0.007 | 0.761±0.009 |
| CTGAN+LL | KDE | **0.880**±0.007 | 0.832±0.007 | 0.641±0.015 | 0.751±0.011 | 0.760±0.012 |
| | WD | 0.878±0.006 | **0.842**±0.004 | 0.645±0.016 | 0.754±0.002 | **0.766**±0.010 |
| | Caus | 0.870±0.013 | 0.838±0.007 | 0.635±0.016 | 0.749±0.005 | 0.753±0.014 |
| TVAE | - | 0.863±0.011 | 0.800±0.016 | 0.631±0.004 | **0.752**±0.003 | 0.789±0.007 |
| | Rnd | 0.875±0.008 | 0.773±0.036 | 0.630±0.002 | 0.750±0.005 | **0.803**±0.009 |
| | Corr | 0.878±0.005 | 0.804±0.014 | 0.633±0.003 | 0.749±0.005 | 0.791±0.015 |
| TVAE+LL | KDE | 0.878±0.007 | **0.816**±0.028 | 0.632±0.003 | 0.751±0.002 | 0.793±0.009 |
| | WD | **0.883**±0.005 | 0.794±0.022 | **0.637**±0.005 | 0.751±0.001 | 0.778±0.017 |
| | Caus | 0.873±0.012 | 0.808±0.012 | 0.630±0.006 | **0.752**±0.002 | 0.795±0.008 |
| GOGGLE | - | 0.742±0.071 | 0.656±0.049 | 0.543±0.039 | 0.600±0.056 | **0.577**±0.010 |
| | Rnd | 0.800±0.029 | 0.643±0.088 | 0.569±0.055 | **0.719**±0.005 | 0.534±0.016 |
| | Corr | 0.794±0.030 | 0.675±0.051 | **0.593**±0.046 | 0.696±0.022 | 0.545±0.011 |
| GOGGLE+LL | KDE | **0.802**±0.016 | 0.678±0.029 | 0.571±0.051 | 0.713±0.021 | 0.538±0.020 |
| | WD | 0.759±0.034 | **0.705**±0.040 | — | 0.709±0.022 | 0.543±0.020 |
| | Caus | 0.741±0.030 | 0.681±0.039 | — | 0.694±0.023 | 0.535±0.019 |

Table B.4: Machine learning efficacy performance comparison between DGM and DGM+LL models trained on News, using XV and MAE. In the first row, I report the real data performance.

| | | XV | MAE |
|---|---|---|---|
| Real data | - | -0.001±0.001 | 3001.1±55.0 |
| WGAN | - | -0.006±0.003 | 3133.6±242.1 |
| | Rnd | -0.008±0.012 | 3093.0±202.3 |
| | Corr | **-0.002**±0.002 | 3014.0±79.1 |
| WGAN+LL | KDE | -0.004±0.001 | 3070.0±98.4 |
| | WD | -0.012±0.010 | 3042.2±100.5 |
| | Caus | -0.012±0.011 | **2968.5**±70.6 |
| TableGAN | - | **-0.001**±0.001 | **2992.2**±35.8 |
| | Rnd | **-0.001**±0.001 | 3033.1±53.9 |
| | Corr | -0.002±0.002 | 3015.9±40.0 |
| TableGAN+LL | KDE | -0.002±0.002 | 3016.8±70.8 |
| | WD | **-0.001**±0.001 | 3009.8±62.4 |
| | Caus | -0.002±0.001 | 3030.4±39.0 |
| CTGAN | - | **-0.002**±0.001 | 3043.8±37.8 |
| | Rnd | **-0.002**±0.001 | 3034.6±29.4 |
| | Corr | **-0.002**±0.001 | **2978.9**±28.7 |
| CTGAN+LL | KDE | -0.003±0.001 | 3029.1±72.7 |
| | WD | -0.003±0.001 | 3007.4±72 |
| | Caus | **-0.002**±0.001 | 3035.2±60.8 |
| TVAE | - | **-0.001**±0.000 | 3021.3±10.0 |
| | Rnd | -0.002±0.001 | 3067.7±71.5 |
| | Corr | **-0.001**±0.001 | 3005.0±56.2 |
| TVAE+LL | KDE | -0.002±0.001 | 3011.7±44.2 |
| | WD | **-0.001**±0.001 | **2995.0**±35.7 |
| | Caus | -0.002±0.001 | 2996.9±22.8 |
| GOGGLE | - | -0.004±0.003 | 3054.5±44.7 |
| | Rnd | -0.001±0.001 | 3042.6±54.1 |
| | Corr | -0.001±0.001 | 3026.0±34.6 |
| GOGGLE+LL | KDE | -0.001±0.000 | **2999.0**±27.3 |
| | WD | **0.000**±0.000 | 3000.9±28.5 |
| | Caus | -0.001±0.001 | 3028.4±39.4 |

Table B.5: Binary F1-score detection results with their corresponding stddevs for all datasets.

| | | URL | WiDS | LCLD | Heloc | FSP | News |
|---|---|---|---|---|---|---|---|
| WGAN | - | 0.865±0.035 | 0.975±0.002 | 0.999±0.001 | 0.964±0.004 | 0.914±0.018 | 0.954±0.022 |
| | Rnd | **0.864**±0.046 | 0.995±0.003 | **0.916**±0.013 | 0.970±0.024 | 0.899±0.078 | 0.969±0.015 |
| | Corr | 0.879±0.027 | 0.975±0.006 | 0.923±0.005 | 0.964±0.007 | **0.843**±0.037 | 0.929±0.023 |
| WGAN+LL | KDE | 0.877±0.016 | 0.975±0.002 | 0.923±0.005 | **0.958**±0.018 | **0.843**±0.037 | **0.927**±0.011 |
| | WD | 0.887±0.018 | **0.932**±0.014 | 0.940±0.014 | 0.968±0.015 | 0.864±0.022 | 0.937±0.011 |
| | Caus | 0.892±0.012 | 0.944±0.005 | — | 0.959±0.013 | 0.864±0.022 | 0.941±0.023 |
| TableGAN | - | 0.831±0.029 | 0.963±0.006 | 0.895±0.024 | **0.923**±0.011 | 0.909±0.021 | 0.927±0.009 |
| | Rnd | 0.840±0.020 | 0.984±0.002 | **0.870**±0.016 | 0.963±0.021 | 0.839±0.061 | 0.953±0.010 |
| | Corr | 0.848±0.025 | **0.956**±0.004 | 0.874±0.011 | 0.952±0.010 | 0.818±0.012 | 0.933±0.011 |
| TableGAN+LL | KDE | **0.827**±0.031 | 0.962±0.007 | **0.870**±0.014 | 0.948±0.018 | 0.830±0.015 | 0.933±0.011 |
| | WD | 0.848±0.025 | 0.961±0.006 | 0.872±0.034 | 0.955±0.014 | **0.808**±0.040 | **0.918**±0.008 |
| | Caus | 0.833±0.025 | 0.962±0.003 | **0.870**±0.014 | 0.957±0.011 | 0.830±0.014 | 0.924±0.007 |
| CTGAN | - | 0.850±0.027 | 0.990±0.002 | 0.848±0.016 | 0.914±0.024 | 0.926±0.011 | 0.901±0.018 |
| | Rnd | 0.834±0.026 | 0.990±0.001 | 0.864±0.022 | 0.910±0.013 | 0.859±0.022 | 0.909±0.016 |
| | Corr | 0.820±0.025 | 0.988±0.003 | 0.845±0.025 | 0.903±0.012 | 0.861±0.021 | 0.905±0.019 |
| CTGAN+LL | KDE | 0.838±0.033 | 0.986±0.002 | 0.848±0.017 | 0.897±0.023 | 0.857±0.022 | 0.902±0.014 |
| | WD | 0.839±0.019 | **0.953**±0.003 | **0.829**±0.030 | **0.888**±0.026 | 0.860±0.025 | 0.901±0.018 |
| | Caus | **0.811**±0.046 | 0.954±0.003 | 0.837±0.034 | 0.900±0.012 | **0.853**±0.039 | **0.882**±0.021 |
| TVAE | - | 0.813±0.037 | **0.926**±0.001 | 0.842±0.019 | 0.914±0.011 | **0.843**±0.027 | 0.877±0.013 |
| | Rnd | 0.825±0.037 | 0.965±0.001 | **0.796**±0.030 | 0.905±0.022 | 0.873±0.015 | 0.875±0.018 |
| | Corr | 0.814±0.037 | 0.959±0.003 | 0.808±0.030 | 0.905±0.014 | 0.855±0.020 | **0.864**±0.008 |
| TVAE+LL | KDE | 0.813±0.030 | 0.961±0.002 | 0.799±0.020 | 0.907±0.021 | 0.879±0.008 | 0.891±0.019 |
| | WD | **0.801**±0.037 | 0.958±0.001 | 0.815±0.020 | 0.911±0.015 | 0.885±0.011 | 0.890±0.019 |
| | Caus | 0.828±0.042 | 0.958±0.001 | 0.798±0.015 | **0.896**±0.022 | 0.868±0.009 | 0.905±0.008 |
| GOGGLE | - | 0.892±0.019 | 0.987±0.018 | **0.890**±0.017 | **0.924**±0.006 | 0.912±0.010 | **0.949**±0.023 |
| | Rnd | 0.890±0.044 | **0.965**±0.006 | 0.903±0.011 | 0.939±0.008 | **0.866**±0.020 | 0.967±0.009 |
| | Corr | 0.898±0.021 | 0.972±0.005 | 0.910±0.017 | 0.938±0.010 | 0.884±0.009 | 0.968±0.015 |
| GOGGLE+LL | KDE | 0.903±0.019 | 0.975±0.013 | 0.911±0.017 | 0.937±0.008 | 0.887±0.014 | 0.957±0.020 |
| | WD | 0.922±0.009 | 0.980±0.015 | — | 0.937±0.011 | 0.881±0.010 | 0.976±0.007 |
| | Caus | **0.885**±0.019 | 0.966±0.008 | — | 0.938±0.009 | 0.881±0.012 | 0.971±0.011 |

Table B.6: Weighted F1-score detection results with their corresponding stddevs for all datasets.

| | | URL | WiDS | LCLD | Heloc | FSP | News |
|---|---|---|---|---|---|---|---|
| WGAN | - | 0.856±0.008 | 0.976±0.002 | 0.999±0.001 | 0.964±0.004 | 0.910±0.027 | 0.954±0.020 |
| | Rnd | **0.850**±0.017 | 0.996±0.003 | **0.911**±0.013 | 0.970±0.024 | 0.900±0.072 | 0.969±0.016 |
| | Corr | 0.860±0.016 | 0.976±0.006 | 0.917±0.007 | 0.964±0.007 | **0.844**±0.022 | 0.929±0.022 |
| WGAN+LL | KDE | 0.870±0.010 | 0.975±0.002 | 0.917±0.007 | **0.957**±0.019 | **0.844**±0.022 | **0.925**±0.013 |
| | WD | 0.879±0.009 | **0.925**±0.016 | 0.939±0.014 | 0.967±0.016 | 0.849±0.031 | 0.935±0.012 |
| | Caus | 0.861±0.009 | 0.929±0.014 | — | 0.958±0.013 | 0.849±0.031 | 0.941±0.023 |
| TableGAN | - | **0.822**±0.007 | 0.964±0.006 | 0.895±0.022 | **0.925**±0.011 | 0.906±0.028 | 0.929±0.013 |
| | Rnd | 0.829±0.011 | 0.984±0.002 | **0.859**±0.015 | 0.962±0.021 | 0.834±0.054 | 0.953±0.011 |
| | Corr | 0.835±0.011 | **0.957**±0.003 | 0.866±0.022 | 0.952±0.010 | 0.813±0.020 | 0.936±0.010 |
| TableGAN+LL | KDE | 0.826±0.006 | 0.963±0.007 | 0.871±0.013 | 0.947±0.019 | 0.818±0.021 | 0.934±0.011 |
| | WD | 0.828±0.013 | 0.962±0.005 | 0.867±0.038 | 0.955±0.015 | **0.802**±0.015 | **0.920**±0.012 |
| | Caus | 0.835±0.014 | 0.963±0.003 | 0.871±0.013 | 0.956±0.012 | 0.816±0.019 | 0.930±0.008 |
| CTGAN | - | 0.862±0.014 | 0.990±0.002 | 0.850±0.019 | 0.915±0.023 | 0.927±0.016 | 0.896±0.023 |
| | Rnd | 0.843±0.017 | 0.990±0.001 | 0.868±0.015 | 0.911±0.013 | **0.861**±0.022 | 0.904±0.018 |
| | Corr | 0.839±0.015 | 0.988±0.002 | 0.847±0.024 | 0.904±0.013 | 0.865±0.028 | 0.902±0.023 |
| CTGAN+LL | KDE | 0.849±0.020 | 0.986±0.002 | 0.838±0.014 | 0.897±0.023 | 0.868±0.009 | 0.895±0.018 |
| | WD | 0.843±0.010 | **0.955**±0.003 | **0.822**±0.025 | **0.890**±0.024 | 0.865±0.018 | 0.899±0.017 |
| | Caus | **0.836**±0.018 | 0.956±0.003 | 0.837±0.020 | 0.901±0.013 | 0.863±0.017 | **0.883**±0.017 |
| TVAE | - | 0.831±0.013 | **0.927**±0.001 | 0.832±0.010 | 0.916±0.010 | **0.847**±0.006 | **0.855**±0.019 |
| | Rnd | 0.829±0.014 | 0.965±0.001 | 0.796±0.025 | 0.908±0.020 | 0.867±0.018 | 0.856±0.012 |
| | Corr | 0.833±0.011 | 0.960±0.003 | **0.795**±0.014 | 0.908±0.013 | 0.857±0.011 | **0.855**±0.010 |
| TVAE+LL | KDE | 0.829±0.014 | 0.962±0.002 | 0.797±0.019 | 0.909±0.020 | 0.873±0.014 | 0.883±0.024 |
| | WD | 0.828±0.012 | 0.959±0.001 | 0.811±0.013 | 0.914±0.013 | 0.876±0.016 | 0.880±0.021 |
| | Caus | **0.827**±0.010 | 0.959±0.001 | 0.801±0.016 | **0.899**±0.020 | 0.870±0.014 | 0.904±0.009 |
| GOGGLE | - | **0.880**±0.012 | 0.987±0.018 | **0.892**±0.014 | **0.926**±0.005 | 0.916±0.011 | **0.955**±0.019 |
| | Rnd | 0.892±0.030 | **0.965**±0.005 | 0.905±0.010 | 0.940±0.008 | 0.823±0.030 | 0.970±0.008 |
| | Corr | 0.899±0.014 | 0.971±0.005 | 0.912±0.017 | 0.939±0.009 | **0.817**±0.004 | 0.972±0.012 |
| GOGGLE+LL | KDE | 0.903±0.017 | 0.975±0.013 | 0.912±0.018 | 0.939±0.008 | 0.865±0.033 | 0.963±0.015 |
| | WD | 0.906±0.030 | 0.980±0.015 | — | 0.939±0.010 | 0.847±0.039 | 0.978±0.006 |
| | Caus | 0.887±0.018 | **0.965**±0.009 | — | 0.939±0.009 | 0.842±0.032 | 0.974±0.009 |

Table B.7: Area under the ROC curve detection results with their corresponding stddevs for all datasets.

| | | URL | WiDS | LCLD | Heloc | FSP | News |
|---|---|---|---|---|---|---|---|
| WGAN | - | **0.872**±0.008 | 0.989±0.002 | 1.000±0.000 | 0.983±0.004 | 0.916±0.016 | 0.964±0.021 |
| | Rnd | 0.877±0.017 | 0.999±0.001 | **0.941**±0.009 | 0.983±0.018 | 0.918±0.060 | 0.981±0.010 |
| | Corr | 0.879±0.009 | 0.989±0.002 | 0.946±0.006 | 0.982±0.007 | **0.875**±0.020 | 0.945±0.020 |
| WGAN+LL | KDE | 0.883±0.009 | 0.989±0.001 | 0.946±0.006 | **0.975**±0.018 | **0.875**±0.020 | **0.941**±0.011 |
| | WD | 0.897±0.009 | **0.948**±0.013 | 0.972±0.011 | 0.983±0.013 | 0.879±0.029 | 0.948±0.018 |
| | Caus | 0.885±0.008 | 0.956±0.006 | — | 0.977±0.009 | 0.879±0.029 | 0.956±0.018 |
| TableGAN | - | **0.850**±0.007 | 0.980±0.004 | 0.926±0.020 | **0.953**±0.009 | 0.907±0.022 | 0.940±0.011 |
| | Rnd | 0.851±0.006 | 0.996±0.001 | **0.900**±0.012 | 0.978±0.016 | 0.866±0.048 | 0.963±0.012 |
| | Corr | 0.856±0.004 | **0.974**±0.002 | 0.904±0.019 | 0.969±0.008 | 0.845±0.005 | 0.950±0.010 |
| TableGAN+LL | KDE | **0.850**±0.005 | 0.979±0.005 | 0.909±0.012 | 0.964±0.017 | 0.849±0.012 | 0.947±0.010 |
| | WD | 0.853±0.004 | 0.977±0.004 | 0.905±0.026 | 0.974±0.010 | **0.840**±0.005 | **0.934**±0.013 |
| | Caus | 0.854±0.012 | 0.979±0.003 | 0.909±0.012 | 0.974±0.011 | 0.847±0.010 | 0.938±0.009 |
| CTGAN | - | 0.879±0.008 | 0.996±0.001 | 0.896±0.011 | 0.953±0.016 | 0.932±0.007 | 0.912±0.021 |
| | Rnd | 0.865±0.013 | 0.997±0.000 | 0.908±0.011 | 0.950±0.008 | 0.897±0.016 | 0.925±0.021 |
| | Corr | **0.864**±0.009 | 0.995±0.001 | 0.900±0.020 | 0.946±0.004 | 0.894±0.025 | 0.922±0.019 |
| CTGAN+LL | KDE | 0.867±0.015 | 0.994±0.001 | 0.897±0.012 | 0.943±0.018 | **0.893**±0.007 | 0.919±0.022 |
| | WD | **0.864**±0.007 | **0.970**±0.002 | **0.877**±0.021 | **0.937**±0.017 | 0.895±0.015 | 0.917±0.016 |
| | Caus | 0.867±0.010 | **0.970**±0.002 | 0.891±0.016 | 0.946±0.010 | 0.894±0.019 | **0.904**±0.016 |
| TVAE | - | 0.854±0.007 | **0.935**±0.002 | 0.861±0.009 | 0.947±0.005 | **0.872**±0.008 | 0.881±0.019 |
| | Rnd | 0.855±0.008 | 0.982±0.001 | 0.843±0.013 | 0.942±0.015 | 0.904±0.015 | 0.882±0.012 |
| | Corr | 0.854±0.009 | 0.977±0.001 | **0.840**±0.011 | 0.943±0.011 | 0.896±0.008 | **0.880**±0.014 |
| TVAE+LL | KDE | **0.850**±0.013 | 0.979±0.001 | 0.842±0.012 | 0.944±0.014 | 0.904±0.013 | 0.908±0.030 |
| | WD | 0.854±0.007 | 0.976±0.001 | 0.850±0.008 | 0.947±0.010 | 0.906±0.011 | 0.907±0.024 |
| | Caus | 0.852±0.003 | 0.976±0.001 | 0.846±0.018 | **0.933**±0.017 | 0.905±0.012 | 0.930±0.008 |
| GOGGLE | - | **0.891**±0.009 | 0.993±0.013 | **0.928**±0.010 | **0.949**±0.003 | 0.920±0.006 | **0.973**±0.013 |
| | Rnd | 0.897±0.028 | **0.979**±0.007 | 0.938±0.010 | 0.957±0.008 | 0.871±0.009 | 0.977±0.007 |
| | Corr | 0.906±0.020 | 0.984±0.005 | 0.944±0.013 | 0.959±0.007 | **0.865**±0.011 | 0.980±0.012 |
| GOGGLE+LL | KDE | 0.906±0.018 | 0.984±0.009 | 0.945±0.013 | 0.959±0.007 | 0.880±0.014 | **0.973**±0.009 |
| | WD | 0.914±0.012 | 0.990±0.008 | — | 0.958±0.009 | 0.878±0.022 | 0.984±0.006 |
| | Caus | 0.893±0.019 | **0.979**±0.009 | — | 0.956±0.008 | 0.873±0.023 | 0.982±0.006 |

## B.3 DGMs vs. P-DGMs

The Linearly-constrained Layer uses the constraints to correct the predictions both at training and inference time. However, the constraints can simply be used at inference time to correct the predictions only once. This latter approach, which results in P-DGM models, is a way of putting guardrails on the output space of the DGMs and could be the preferred option for users who do not want to make any modifications to their models or retrain them. And, while both methods guarantee that the constraints are satisfied by the predictions, their impact on the models' performance might differ. To see this, I compare DGMs with P-DGMs whose predictions have been corrected according to the five different label ordering methods detailed in Chapter 4: random-, correlation-, KDE-, Wasserstein distance-, and causal-based methods.

For completeness, I also report the mean performance and standard deviation from the mean for:

1. Machine learning efficacy: In Tables B.8, B.9 and B.10, I compare the performance of the DGM and P-DGM models in terms of machine learning efficacy and report the results w.r.t. 3 different metrics: F1-score, weighted F1-score, and Area Under the ROC Curve. Separately, in Table B.11, I report the performance of the synthetic data for the News dataset, using two metrics: XV and MAE. As the results show, putting guardrails on the output space of the models can help increase the performance. However, the overall gains are not as high as those obtained by the DGM+LL models, which correct the outputs at training time.

2. Detection: In Tables B.12, B.13 and B.14, I compare the performance of the DGM and P-DGM models in terms of detection. Similarly to the machine learning efficacy case, post-processing the outputs of the unconstrained models can only slightly help increase the overall performance.

Table B.8: Binary (macro) F1-score machine learning efficacy results with their corresponding stddevs for binary (multiclass) classification datasets when postprocessing the unconstrained predictions.

| | | URL | WiDS | LCLD | Heloc | FSP |
|---|---|---|---|---|---|---|
| WGAN | - | 0.756±0.030 | 0.329±0.059 | **0.239**±0.026 | 0.634±0.099 | 0.357±0.020 |
| | Rnd | **0.767**±0.034 | 0.319±0.053 | 0.200±0.030 | 0.641±0.113 | 0.356±0.020 |
| | Corr | **0.767**±0.036 | 0.332±0.050 | 0.214±0.021 | 0.641±0.115 | **0.359**±0.024 |
| P-WGAN | KDE | **0.767**±0.035 | **0.334**±0.059 | 0.214±0.021 | 0.641±0.112 | **0.359**±0.024 |
| | WD | 0.763±0.035 | 0.332±0.052 | 0.210±0.019 | 0.640±0.107 | 0.353±0.019 |
| | Caus | 0.762±0.034 | 0.332±0.049 | 0.210±0.019 | **0.643**±0.101 | 0.353±0.019 |
| TableGAN | - | 0.562±0.051 | 0.171±0.037 | 0.123±0.041 | 0.593±0.058 | **0.199**±0.044 |
| | Rnd | **0.565**±0.048 | 0.173±0.049 | 0.115±0.028 | 0.594±0.058 | 0.194±0.046 |
| | Corr | 0.553±0.050 | 0.174±0.045 | 0.125±0.032 | 0.594±0.060 | 0.194±0.037 |
| P-TableGAN | KDE | 0.556±0.060 | 0.171±0.042 | 0.120±0.025 | 0.592±0.060 | 0.193±0.042 |
| | WD | 0.562±0.070 | **0.181**±0.045 | **0.126**±0.030 | **0.595**±0.058 | 0.188±0.043 |
| | Caus | 0.563±0.060 | 0.171±0.040 | 0.124±0.030 | **0.595**±0.056 | 0.191±0.050 |
| CTGAN | - | 0.822±0.017 | 0.362±0.033 | 0.247±0.087 | 0.736±0.035 | 0.374±0.034 |
| | Rnd | 0.818±0.014 | 0.364±0.032 | 0.255±0.089 | 0.735±0.038 | 0.365±0.021 |
| | Corr | 0.813±0.008 | 0.365±0.038 | 0.246±0.087 | **0.743**±0.032 | 0.372±0.033 |
| P-CTGAN | KDE | **0.823**±0.017 | 0.357±0.040 | 0.248±0.077 | 0.735±0.031 | 0.371±0.035 |
| | WD | **0.823**±0.017 | **0.375**±0.033 | **0.263**±0.076 | 0.738±0.032 | **0.375**±0.027 |
| | Caus | **0.823**±0.012 | 0.363±0.036 | 0.249±0.092 | 0.738±0.033 | 0.369±0.021 |
| TVAE | - | 0.810±0.008 | 0.282±0.029 | **0.185**±0.021 | 0.735±0.010 | **0.473**±0.016 |
| | Rnd | 0.810±0.011 | 0.266±0.020 | 0.172±0.011 | **0.739**±0.004 | 0.464±0.019 |
| | Corr | **0.815**±0.009 | 0.282±0.026 | 0.176±0.028 | 0.730±0.006 | 0.462±0.018 |
| P-TVAE | KDE | **0.815**±0.009 | **0.285**±0.034 | 0.171±0.030 | 0.735±0.007 | 0.463±0.021 |
| | WD | 0.811±0.012 | 0.280±0.028 | 0.179±0.028 | 0.735±0.007 | 0.462±0.012 |
| | Caus | 0.809±0.011 | 0.269±0.051 | 0.169±0.018 | 0.734±0.007 | 0.463±0.019 |
| GOGGLE | - | 0.622±0.094 | 0.189±0.038 | 0.163±0.119 | 0.596±0.072 | 0.152±0.003 |
| | Rnd | **0.626**±0.098 | **0.192**±0.042 | 0.164±0.121 | **0.601**±0.060 | 0.151±0.007 |
| | Corr | 0.623±0.089 | 0.191±0.038 | **0.169**±0.126 | **0.601**±0.063 | **0.155**±0.010 |
| P-GOGGLE | KDE | **0.626**±0.096 | 0.189±0.039 | **0.169**±0.126 | 0.597±0.066 | **0.155**±0.011 |
| | WD | 0.610±0.102 | 0.191±0.042 | 0.166±0.118 | **0.601**±0.063 | 0.154±0.006 |
| | Caus | 0.624±0.100 | 0.191±0.041 | 0.166±0.118 | 0.598±0.069 | 0.154±0.009 |

Table B.9: Weighted F1-score machine learning efficacy results with their corresponding stddevs for binary and multiclass classification datasets when post-processing the unconstrained predictions.

| | | URL | WiDS | LCLD | Heloc | FSP |
|---|---|---|---|---|---|---|
| WGAN | - | 0.764±0.018 | 0.381±0.057 | **0.359**±0.019 | 0.599±0.050 | 0.339±0.021 |
| | Rnd | 0.768±0.028 | 0.372±0.052 | 0.342±0.019 | 0.600±0.057 | 0.337±0.022 |
| | Corr | 0.769±0.027 | 0.384±0.048 | 0.353±0.014 | 0.598±0.054 | **0.341**±0.026 |
| P-WGAN | KDE | 0.761±0.028 | **0.386**±0.057 | 0.353±0.014 | **0.603**±0.059 | **0.341**±0.026 |
| | WD | 0.765±0.024 | 0.384±0.051 | 0.349±0.014 | 0.600±0.056 | 0.335±0.021 |
| | Caus | **0.772**±0.021 | 0.383±0.048 | 0.349±0.014 | 0.601±0.050 | 0.335±0.021 |
| TableGAN | - | 0.659±0.035 | 0.240±0.034 | 0.286±0.029 | **0.615**±0.030 | **0.199**±0.043 |
| | Rnd | **0.660**±0.035 | 0.242±0.045 | 0.281±0.022 | 0.614±0.028 | 0.195±0.046 |
| | Corr | 0.656±0.035 | 0.242±0.041 | 0.286±0.023 | 0.614±0.030 | 0.195±0.036 |
| P-TableGAN | KDE | 0.658±0.040 | 0.240±0.039 | 0.285±0.018 | 0.613±0.029 | 0.193±0.042 |
| | WD | 0.659±0.045 | **0.250**±0.041 | **0.287**±0.023 | 0.614±0.029 | 0.189±0.041 |
| | Caus | **0.660**±0.041 | 0.240±0.036 | 0.286±0.021 | **0.615**±0.028 | 0.191±0.050 |
| CTGAN | - | 0.799±0.033 | 0.405±0.034 | 0.379±0.061 | 0.675±0.015 | 0.372±0.034 |
| | Rnd | 0.794±0.031 | 0.407±0.032 | 0.383±0.070 | 0.671±0.015 | 0.363±0.019 |
| | Corr | 0.801±0.008 | 0.407±0.040 | 0.379±0.068 | 0.678±0.010 | 0.370±0.033 |
| P-CTGAN | KDE | **0.802**±0.032 | 0.400±0.042 | 0.380±0.058 | 0.671±0.010 | 0.369±0.033 |
| | WD | **0.802**±0.032 | **0.419**±0.034 | **0.386**±0.056 | **0.679**±0.012 | **0.373**±0.025 |
| | Caus | 0.799±0.039 | 0.406±0.037 | 0.376±0.065 | 0.673±0.011 | 0.366±0.021 |
| TVAE | - | 0.802±0.012 | 0.342±0.027 | **0.330**±0.016 | **0.696**±0.006 | **0.463**±0.017 |
| | Rnd | 0.802±0.015 | 0.327±0.018 | 0.323±0.009 | **0.696**±0.004 | 0.454±0.020 |
| | Corr | 0.807±0.008 | 0.342±0.024 | 0.325±0.018 | 0.691±0.004 | 0.451±0.019 |
| P-TVAE | KDE | 0.807±0.008 | **0.345**±0.031 | 0.322±0.020 | 0.694±0.004 | 0.452±0.022 |
| | WD | **0.809**±0.005 | 0.340±0.026 | 0.323±0.020 | 0.694±0.004 | 0.452±0.013 |
| | Caus | 0.801±0.015 | 0.329±0.046 | 0.321±0.012 | 0.693±0.005 | 0.452±0.020 |
| GOGGLE | - | 0.648±0.074 | 0.198±0.067 | 0.315±0.086 | **0.566**±0.050 | 0.139±0.002 |
| | Rnd | 0.646±0.071 | **0.202**±0.070 | 0.315±0.086 | **0.566**±0.047 | 0.138±0.006 |
| | Corr | 0.646±0.070 | 0.201±0.067 | **0.318**±0.090 | **0.566**±0.047 | **0.142**±0.009 |
| P-GOGGLE | KDE | **0.650**±0.072 | 0.197±0.069 | **0.318**±0.090 | 0.565±0.048 | 0.141±0.011 |
| | WD | 0.636±0.073 | 0.200±0.072 | 0.317±0.084 | **0.566**±0.049 | 0.140±0.005 |
| | Caus | 0.645±0.072 | 0.200±0.070 | 0.317±0.084 | 0.564±0.052 | 0.141±0.009 |

Table B.10: Area under the ROC curve machine learning efficacy results with their corresponding stddevs for binary and multiclass classification datasets when post-processing the unconstrained predictions.

| | | URL | WiDS | LCLD | Heloc | FSP |
|---|---|---|---|---|---|---|
| WGAN | - | 0.839±0.016 | 0.775±0.038 | **0.618**±0.011 | 0.677±0.033 | **0.742**±0.007 |
| | Rnd | 0.839±0.019 | 0.777±0.037 | **0.618**±0.010 | 0.682±0.042 | 0.736±0.011 |
| | Corr | 0.839±0.020 | 0.781±0.035 | 0.617±0.008 | **0.683**±0.038 | 0.740±0.010 |
| P-WGAN | KDE | 0.833±0.021 | **0.783**±0.039 | 0.617±0.008 | **0.683**±0.041 | 0.740±0.010 |
| | WD | 0.839±0.017 | 0.776±0.037 | 0.612±0.010 | 0.682±0.039 | 0.741±0.012 |
| | Caus | **0.846**±0.012 | 0.774±0.033 | 0.612±0.010 | 0.675±0.034 | 0.741±0.012 |
| TableGAN | - | 0.843±0.020 | 0.740±0.021 | 0.587±0.027 | **0.707**±0.007 | 0.642±0.026 |
| | Rnd | 0.848±0.019 | **0.749**±0.019 | 0.585±0.027 | 0.703±0.008 | 0.641±0.030 |
| | Corr | 0.846±0.020 | 0.731±0.022 | **0.588**±0.028 | **0.707**±0.011 | **0.645**±0.031 |
| P-TableGAN | KDE | **0.854**±0.016 | 0.735±0.016 | 0.587±0.027 | **0.707**±0.011 | 0.644±0.035 |
| | WD | 0.845±0.021 | 0.738±0.022 | 0.585±0.030 | 0.702±0.006 | 0.637±0.034 |
| | Caus | 0.847±0.016 | 0.741±0.015 | **0.588**±0.027 | **0.707**±0.009 | 0.636±0.033 |
| CTGAN | - | 0.859±0.040 | 0.835±0.012 | 0.651±0.020 | 0.744±0.009 | **0.760**±0.014 |
| | Rnd | 0.859±0.038 | **0.837**±0.012 | 0.652±0.024 | 0.743±0.008 | 0.754±0.008 |
| | Corr | **0.866**±0.007 | 0.835±0.012 | 0.651±0.023 | 0.745±0.008 | 0.754±0.011 |
| P-CTGAN | KDE | 0.863±0.035 | 0.835±0.012 | 0.651±0.018 | 0.743±0.009 | 0.754±0.009 |
| | WD | 0.863±0.035 | 0.836±0.012 | **0.653**±0.017 | **0.747**±0.009 | 0.759±0.012 |
| | Caus | 0.856±0.039 | 0.835±0.013 | 0.646±0.021 | 0.743±0.008 | 0.754±0.008 |
| TVAE | - | 0.863±0.011 | **0.800**±0.016 | **0.631**±0.004 | **0.752**±0.003 | 0.789±0.007 |
| | Rnd | 0.865±0.013 | 0.785±0.017 | 0.630±0.007 | 0.751±0.003 | 0.787±0.011 |
| | Corr | **0.870**±0.007 | 0.787±0.009 | 0.629±0.004 | **0.752**±0.004 | 0.779±0.014 |
| P-TVAE | KDE | **0.870**±0.007 | 0.797±0.016 | 0.630±0.002 | 0.750±0.001 | 0.788±0.010 |
| | WD | **0.870**±0.007 | 0.778±0.019 | 0.627±0.005 | 0.750±0.001 | 0.779±0.012 |
| | Caus | 0.864±0.014 | 0.793±0.015 | 0.629±0.006 | 0.751±0.001 | **0.792**±0.009 |
| GOGGLE | - | **0.742**±0.071 | 0.656±0.049 | 0.543±0.039 | 0.600±0.056 | 0.577±0.010 |
| | Rnd | 0.738±0.067 | 0.665±0.043 | **0.548**±0.035 | 0.601±0.056 | 0.575±0.008 |
| | Corr | 0.740±0.063 | 0.667±0.044 | 0.542±0.036 | 0.601±0.058 | 0.575±0.010 |
| P-GOGGLE | KDE | 0.741±0.067 | 0.667±0.060 | 0.542±0.036 | **0.603**±0.058 | 0.578±0.014 |
| | WD | 0.738±0.070 | 0.663±0.054 | 0.542±0.036 | 0.601±0.057 | 0.578±0.012 |
| | Caus | 0.739±0.069 | **0.669**±0.044 | 0.542±0.036 | 0.601±0.060 | **0.579**±0.013 |

Table B.11: Machine learning efficacy performance comparison between DGM and P-DGM models trained on News.

| | | XV | MAE |
|---|---|---|---|
| WGAN | - | -0.006±0.003 | 3133.6±242.1 |
| | Rnd | **-0.005**±0.002 | 3151.3±219.8 |
| | Corr | -0.012±0.011 | **3109.2**±220.4 |
| P-WGAN | KDE | -0.006±0.003 | 3159.7±210.7 |
| | WD | -0.013±0.011 | 3119.8±230.6 |
| | Caus | -0.015±0.013 | 3076.4±211.5 |
| TableGAN | - | **-0.001**±0.001 | 2992.2±35.8 |
| | Rnd | **-0.001**±0.001 | **2981.9**±32.3 |
| | Corr | **-0.001**±0.001 | 3015.3±43.9 |
| P-TableGAN | KDE | **-0.001**±0.001 | 3022.7±23.6 |
| | WD | **-0.001**±0.001 | 3006.6±20.3 |
| | Caus | -0.002±0.001 | 3006.1±62.1 |
| CTGAN | - | -0.002±0.001 | 3043.8±37.8 |
| | Rnd | -0.003±0.004 | 3013.4±36.2 |
| | Corr | -0.006±0.002 | **2999.3**±27.3 |
| P-CTGAN | KDE | -0.003±0.003 | 3004.8±47.4 |
| | WD | **-0.001**±0.000 | 3040.9±32.3 |
| | Caus | -0.003±0.002 | 3090.7±60.6 |
| TVAE | - | **-0.001**±0.000 | 3021.3±10.0 |
| | Rnd | **-0.001**±0.001 | 3040.5±60.4 |
| | Corr | -0.002±0.001 | 3003.8±22.7 |
| P-TVAE | KDE | **-0.001**±0.001 | 3032.2±39.3 |
| | WD | **-0.001**±0.000 | 3013.1±47.1 |
| | Caus | -0.002±0.001 | **3003.0**±61.4 |
| GOGGLE | - | **-0.004**±0.003 | 3054.5±44.7 |
| | Rnd | -0.006±0.005 | 3024.1±37.2 |
| | Corr | -0.005±0.002 | 3077.9±54.1 |
| P-GOGGLE | KDE | **-0.004**±0.003 | 3012.3±36.6 |
| | WD | -0.010±0.010 | 3030±17.5 |
| | Caus | -0.006±0.004 | **3012.1**±41.6 |

Table B.12: Binary F1-score detection results with their corresponding stddevs for all datasets when post-processing the unconstrained predictions.

| | | URL | WiDS | LCLD | Heloc | FSP | News |
|---|---|---|---|---|---|---|---|
| WGAN | - | 0.865±0.035 | **0.975**±0.002 | 0.999±0.001 | 0.964±0.004 | 0.914±0.018 | **0.954**±0.022 |
| | Rnd | **0.856**±0.037 | 0.979±0.003 | 0.916±0.008 | 0.965±0.004 | **0.910**±0.015 | 0.955±0.022 |
| | Corr | 0.857±0.033 | 0.976±0.002 | 0.921±0.004 | 0.968±0.005 | 0.911±0.018 | **0.954**±0.026 |
| P-WGAN | KDE | 0.862±0.037 | 0.978±0.003 | 0.921±0.004 | 0.967±0.004 | 0.911±0.018 | 0.957±0.019 |
| | WD | 0.863±0.028 | 0.979±0.003 | **0.910**±0.014 | 0.967±0.004 | 0.911±0.018 | 0.955±0.025 |
| | Caus | 0.868±0.032 | 0.980±0.002 | **0.910**±0.014 | **0.963**±0.005 | 0.911±0.018 | **0.954**±0.023 |
| TableGAN | - | 0.831±0.029 | 0.963±0.006 | 0.895±0.024 | 0.923±0.011 | 0.909±0.021 | 0.927±0.009 |
| | Rnd | 0.842±0.034 | **0.962**±0.006 | 0.896±0.026 | 0.923±0.010 | 0.858±0.031 | 0.931±0.010 |
| | Corr | 0.833±0.035 | 0.963±0.005 | 0.900±0.024 | 0.922±0.011 | 0.860±0.029 | **0.920**±0.009 |
| P-TableGAN | KDE | 0.834±0.041 | **0.962**±0.006 | 0.901±0.019 | **0.921**±0.011 | **0.856**±0.029 | 0.928±0.004 |
| | WD | 0.839±0.038 | **0.962**±0.006 | **0.889**±0.019 | 0.924±0.010 | 0.861±0.027 | 0.925±0.006 |
| | Caus | **0.826**±0.036 | 0.963±0.006 | 0.899±0.027 | 0.922±0.011 | 0.861±0.027 | 0.925±0.006 |
| CTGAN | - | 0.850±0.027 | 0.990±0.002 | 0.848±0.016 | 0.914±0.024 | 0.926±0.011 | 0.901±0.018 |
| | Rnd | 0.845±0.028 | 0.991±0.002 | **0.839**±0.015 | 0.916±0.028 | 0.902±0.029 | 0.898±0.012 |
| | Corr | 0.855±0.014 | **0.989**±0.003 | 0.845±0.019 | **0.913**±0.028 | **0.895**±0.030 | 0.898±0.006 |
| P-CTGAN | KDE | 0.850±0.023 | 0.991±0.002 | 0.841±0.019 | **0.913**±0.027 | **0.895**±0.032 | 0.902±0.010 |
| | WD | 0.850±0.023 | 0.990±0.002 | 0.862±0.018 | **0.913**±0.028 | 0.904±0.024 | 0.901±0.012 |
| | Caus | **0.837**±0.033 | 0.990±0.002 | 0.854±0.014 | 0.914±0.026 | 0.905±0.023 | **0.891**±0.018 |
| TVAE | - | 0.813±0.037 | **0.926**±0.001 | 0.842±0.019 | 0.914±0.011 | **0.843**±0.027 | 0.877±0.013 |
| | Rnd | 0.806±0.039 | 0.962±0.002 | 0.835±0.018 | 0.913±0.007 | 0.881±0.007 | 0.873±0.012 |
| | Corr | 0.805±0.044 | 0.962±0.001 | 0.829±0.022 | 0.912±0.011 | 0.881±0.011 | 0.867±0.014 |
| P-TVAE | KDE | 0.805±0.044 | 0.963±0.001 | 0.837±0.010 | **0.908**±0.011 | 0.874±0.012 | 0.868±0.012 |
| | WD | **0.800**±0.041 | 0.962±0.002 | 0.834±0.019 | **0.908**±0.011 | 0.884±0.010 | 0.871±0.009 |
| | Caus | 0.809±0.031 | 0.962±0.001 | **0.826**±0.016 | 0.910±0.011 | 0.880±0.009 | **0.866**±0.016 |
| GOGGLE | - | 0.892±0.019 | **0.987**±0.018 | **0.890**±0.017 | **0.924**±0.006 | 0.912±0.010 | 0.949±0.023 |
| | Rnd | **0.885**±0.025 | 0.988±0.017 | 0.892±0.012 | 0.927±0.006 | 0.907±0.005 | 0.951±0.021 |
| | Corr | 0.897±0.010 | 0.988±0.016 | 0.902±0.005 | 0.925±0.006 | 0.909±0.006 | 0.950±0.021 |
| P-GOGGLE | KDE | 0.898±0.015 | **0.987**±0.017 | 0.902±0.005 | 0.927±0.007 | 0.907±0.009 | 0.950±0.020 |
| | WD | 0.898±0.015 | 0.988±0.017 | 0.895±0.008 | 0.927±0.006 | 0.913±0.013 | 0.949±0.019 |
| | Caus | 0.911±0.007 | 0.988±0.016 | 0.895±0.008 | 0.927±0.006 | **0.903**±0.009 | — |

Table B.13: Weighted F1-score detection results with their corresponding stddevs for all datasets when post-processing the unconstrained predictions.

| | | URL | WiDS | LCLD | Heloc | FSP | News |
|---|---|---|---|---|---|---|---|
| WGAN | - | 0.856±0.008 | **0.976**±0.002 | 0.999±0.001 | 0.964±0.004 | 0.910±0.027 | **0.954**±0.020 |
| | Rnd | **0.852**±0.011 | 0.980±0.003 | 0.907±0.010 | 0.964±0.004 | **0.906**±0.009 | 0.956±0.020 |
| | Corr | **0.852**±0.008 | 0.977±0.002 | 0.914±0.002 | 0.968±0.004 | 0.912±0.018 | **0.954**±0.025 |
| P-WGAN | KDE | 0.855±0.010 | 0.978±0.003 | 0.914±0.002 | 0.967±0.004 | 0.912±0.018 | 0.959±0.017 |
| | WD | 0.856±0.007 | 0.979±0.003 | **0.905**±0.010 | 0.967±0.004 | 0.911±0.020 | 0.956±0.023 |
| | Caus | 0.858±0.011 | 0.981±0.002 | **0.905**±0.010 | **0.963**±0.005 | 0.911±0.020 | **0.954**±0.022 |
| TableGAN | - | 0.822±0.007 | 0.964±0.006 | 0.895±0.022 | 0.925±0.011 | 0.906±0.028 | 0.929±0.013 |
| | Rnd | 0.823±0.005 | **0.963**±0.006 | **0.888**±0.031 | 0.924±0.011 | 0.873±0.018 | 0.934±0.012 |
| | Corr | 0.826±0.006 | 0.964±0.005 | 0.898±0.025 | **0.923**±0.011 | 0.868±0.016 | **0.923**±0.009 |
| P-TableGAN | KDE | 0.824±0.005 | **0.963**±0.006 | 0.896±0.021 | **0.923**±0.011 | 0.874±0.022 | 0.929±0.006 |
| | WD | 0.826±0.008 | **0.963**±0.006 | 0.894±0.017 | 0.925±0.010 | 0.874±0.029 | 0.924±0.010 |
| | Caus | **0.819**±0.009 | 0.964±0.007 | 0.898±0.023 | 0.924±0.010 | **0.865**±0.018 | 0.925±0.011 |
| CTGAN | - | 0.862±0.014 | 0.990±0.002 | 0.850±0.019 | 0.915±0.023 | 0.927±0.016 | 0.896±0.023 |
| | Rnd | 0.859±0.017 | 0.991±0.002 | **0.840**±0.014 | 0.917±0.027 | 0.905±0.026 | 0.894±0.006 |
| | Corr | 0.860±0.012 | **0.989**±0.003 | 0.851±0.009 | **0.914**±0.027 | 0.902±0.017 | 0.894±0.014 |
| P-CTGAN | KDE | 0.861±0.017 | 0.991±0.002 | 0.848±0.006 | **0.914**±0.027 | 0.904±0.025 | 0.899±0.014 |
| | WD | 0.861±0.017 | 0.990±0.002 | 0.858±0.018 | **0.914**±0.027 | **0.897**±0.028 | 0.897±0.011 |
| | Caus | **0.854**±0.022 | 0.990±0.002 | 0.852±0.015 | **0.914**±0.026 | 0.902±0.023 | **0.888**±0.018 |
| TVAE | - | 0.831±0.013 | **0.927**±0.001 | 0.832±0.010 | 0.916±0.010 | **0.847**±0.006 | 0.855±0.019 |
| | Rnd | 0.828±0.012 | 0.963±0.002 | 0.832±0.010 | 0.915±0.007 | 0.877±0.010 | 0.856±0.017 |
| | Corr | 0.829±0.012 | 0.962±0.001 | **0.825**±0.011 | 0.914±0.011 | 0.882±0.012 | 0.860±0.015 |
| P-TVAE | KDE | 0.829±0.012 | 0.964±0.001 | 0.831±0.014 | **0.910**±0.011 | 0.870±0.016 | **0.848**±0.014 |
| | WD | **0.823**±0.012 | 0.962±0.002 | 0.832±0.010 | **0.910**±0.011 | 0.885±0.015 | 0.861±0.013 |
| | Caus | 0.828±0.010 | 0.962±0.001 | 0.827±0.008 | 0.912±0.011 | 0.884±0.016 | 0.852±0.014 |
| GOGGLE | - | 0.880±0.012 | **0.987**±0.018 | 0.892±0.014 | **0.926**±0.005 | 0.916±0.011 | 0.955±0.019 |
| | Rnd | 0.878±0.018 | 0.988±0.017 | **0.889**±0.015 | 0.928±0.005 | 0.911±0.007 | 0.957±0.017 |
| | Corr | **0.875**±0.021 | 0.988±0.016 | 0.901±0.006 | **0.926**±0.005 | 0.907±0.012 | 0.957±0.017 |
| P-GOGGLE | KDE | 0.883±0.011 | **0.987**±0.017 | 0.901±0.006 | 0.928±0.006 | **0.906**±0.011 | 0.956±0.017 |
| | WD | 0.884±0.007 | 0.988±0.017 | 0.893±0.005 | 0.928±0.005 | 0.909±0.020 | 0.954±0.016 |
| | Caus | 0.882±0.021 | 0.988±0.016 | 0.893±0.005 | 0.928±0.005 | 0.908±0.009 | — |

Table B.14: Area under the ROC curve detection results with their corresponding stddevs for all datasets when post-processing the unconstrained predictions.

| | | URL | WiDS | LCLD | Heloc | FSP | News |
|---|---|---|---|---|---|---|---|
| WGAN | - | 0.872±0.008 | **0.989**±0.002 | 1.000±0.000 | **0.983**±0.004 | **0.916**±0.016 | **0.964**±0.021 |
| | Rnd | **0.867**±0.003 | 0.992±0.002 | **0.936**±0.010 | 0.985±0.002 | 0.927±0.006 | 0.966±0.020 |
| | Corr | 0.869±0.004 | 0.991±0.002 | 0.942±0.001 | 0.986±0.003 | 0.932±0.013 | 0.966±0.024 |
| P-WGAN | KDE | 0.870±0.003 | 0.991±0.002 | 0.942±0.001 | 0.986±0.002 | 0.932±0.013 | 0.969±0.017 |
| | WD | 0.873±0.005 | 0.992±0.002 | 0.937±0.006 | 0.986±0.002 | 0.930±0.015 | 0.965±0.024 |
| | Caus | 0.873±0.004 | 0.992±0.001 | 0.937±0.006 | **0.983**±0.003 | 0.930±0.015 | 0.965±0.022 |
| TableGAN | - | 0.850±0.007 | 0.980±0.004 | 0.926±0.020 | 0.953±0.009 | 0.907±0.022 | 0.940±0.011 |
| | Rnd | 0.850±0.006 | 0.980±0.004 | **0.925**±0.024 | **0.952**±0.009 | 0.894±0.016 | 0.945±0.012 |
| | Corr | 0.848±0.007 | 0.981±0.003 | 0.929±0.023 | 0.953±0.008 | **0.886**±0.018 | **0.937**±0.011 |
| P-TableGAN | KDE | 0.853±0.009 | 0.980±0.003 | 0.934±0.016 | **0.952**±0.008 | 0.890±0.022 | 0.939±0.006 |
| | WD | 0.850±0.007 | **0.979**±0.005 | 0.926±0.016 | 0.953±0.008 | 0.891±0.026 | 0.941±0.011 |
| | Caus | **0.845**±0.005 | 0.980±0.004 | 0.930±0.022 | **0.952**±0.009 | 0.889±0.013 | 0.941±0.006 |
| CTGAN | - | 0.879±0.008 | **0.996**±0.001 | 0.896±0.011 | 0.953±0.016 | 0.932±0.007 | 0.912±0.021 |
| | Rnd | 0.876±0.012 | 0.997±0.001 | **0.892**±0.008 | 0.953±0.021 | 0.926±0.019 | 0.914±0.012 |
| | Corr | 0.878±0.004 | 0.997±0.001 | 0.896±0.005 | 0.952±0.018 | 0.922±0.013 | 0.921±0.017 |
| P-CTGAN | KDE | 0.876±0.013 | 0.997±0.001 | 0.893±0.004 | 0.952±0.020 | 0.926±0.017 | 0.919±0.021 |
| | WD | 0.876±0.013 | 0.997±0.001 | 0.899±0.009 | 0.952±0.018 | **0.920**±0.020 | 0.914±0.012 |
| | Caus | **0.875**±0.013 | 0.997±0.001 | 0.893±0.010 | **0.951**±0.020 | 0.924±0.018 | **0.910**±0.015 |
| TVAE | - | 0.854±0.007 | **0.935**±0.002 | 0.861±0.009 | 0.947±0.005 | **0.872**±0.008 | 0.881±0.019 |
| | Rnd | **0.850**±0.010 | 0.978±0.002 | 0.862±0.008 | 0.947±0.004 | 0.908±0.008 | 0.886±0.021 |
| | Corr | 0.851±0.009 | 0.979±0.001 | **0.857**±0.007 | 0.946±0.007 | 0.910±0.011 | 0.883±0.014 |
| P-TVAE | KDE | 0.851±0.009 | 0.979±0.001 | 0.861±0.005 | **0.943**±0.006 | 0.896±0.013 | **0.874**±0.018 |
| | WD | **0.850**±0.007 | 0.979±0.001 | 0.862±0.007 | **0.943**±0.006 | 0.912±0.011 | 0.886±0.012 |
| | Caus | 0.852±0.007 | 0.979±0.001 | 0.858±0.006 | 0.945±0.006 | 0.911±0.013 | 0.876±0.016 |
| GOGGLE | - | 0.891±0.009 | **0.993**±0.013 | 0.928±0.010 | **0.949**±0.003 | 0.920±0.006 | 0.973±0.013 |
| | Rnd | **0.889**±0.017 | **0.993**±0.013 | **0.924**±0.008 | 0.952±0.004 | 0.923±0.011 | 0.976±0.012 |
| | Corr | 0.895±0.010 | **0.993**±0.012 | 0.932±0.005 | 0.951±0.003 | 0.918±0.011 | 0.977±0.011 |
| P-GOGGLE | KDE | 0.894±0.017 | **0.993**±0.012 | 0.932±0.005 | 0.952±0.003 | **0.917**±0.010 | 0.976±0.012 |
| | WD | 0.897±0.008 | **0.993**±0.012 | 0.927±0.006 | 0.952±0.003 | 0.920±0.019 | 0.972±0.010 |
| | Caus | 0.901±0.009 | 0.994±0.011 | 0.927±0.006 | 0.951±0.003 | 0.921±0.011 | — |

# Appendix C

# Hyperparameter Search for Chapter 5

I carried out an extensive hyperparameter search to identify the optimal configurations for each DGM. I selected these configurations based on the efficacy performance: for the classification datasets (i.e., URL, CCS, LCLD, and Heloc), I used the average of the F1-score, weighted F1-score, and AUC, whereas for the regression dataset (i.e., House), I used the average of the Mean Absolute Error and Root Mean Square Error.

For clarity, I describe the hyperparameter search space below, for each of the considered models. For the GOGGLE model, I adopted the same optimiser and learning rate settings as in (Liu et al., 2022). Specifically, I used the Adam optimiser (Kingma and Ba, 2015) with three learning rates: $\{1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}\}$. Additionally, I experimented with a set of values for the threshold parameter: $\{1 \times 10^{-1}, 2 \times 10^{-1}\}$. For the TVAE model, Adam was used again, but with a different set of learning rates: $\{5 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$. For the other three models (i.e., WGAN, TableGAN, and CTGAN), I tested three different optimisers: Adam, RMSProp (Hinton, 2014), and SGD (Ruder, 2016), each paired with its own set of learning rates, as follows:

- WGAN: Adam: $\{1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$, RMSProp: $\{5 \times 10^{-5}, 1 \times 10^{-4}, 1 \times 10^{-3}\}$, SGD: $\{1 \times 10^{-4}, 1 \times 10^{-3}\}$
- TableGAN: Adam: $\{5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}\}$, RMSProp: $\{1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$, SGD: $\{1 \times 10^{-4}, 1 \times 10^{-3}\}$
- CTGAN: Adam: $\{5 \times 10^{-5}, 1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$, RMSProp: $\{1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}\}$, SGD: $\{1 \times 10^{-4}, 1 \times 10^{-3}\}$

Table C.1: Best hyperparameter settings used for DGMs (and also for DGMs+LL and DGMs+DRL) in the experiments.

| Model/Dataset | Hyperparameter | URL | CCS | LCLD | Heloc | House |
|---|---|---|---|---|---|---|
| WGAN | Batch size | 510 | 256 | 510 | 510 | 510 |
| | Optimiser | Adam | Adam | Adam | Adam | Adam |
| | Learning rate | 0.001 | 0.001 | 0.001 | 0.001 | 0.0002 |
| | Epochs | 150 | 1250 | 15 | 150 | 100 |
| | Discriminator iters | 5 | 5 | 5 | 5 | 1 |
| | LL Ordering | Corr | KDE | Rnd | Corr | Rnd |
| | DRL Ordering | Rnd | Rnd | KDE | Rnd | KDE |
| TableGAN | Batch size | 128 | 128 | 510 | 128 | 256 |
| | Optimiser | Adam | Adam | Adam | Adam | Adam |
| | Learning rate | 0.001 | 0.0002 | 0.01 | 0.001 | 0.0001 |
| | Epochs | 300 | 2000 | 20 | 200 | 50 |
| | LL Ordering | Corr | KDE | KDE | Corr | KDE |
| | DRL Ordering | KDE | KDE | KDE | Corr | Rnd |
| CTGAN | Batch size | 500 | 70 | 500 | 500 | 500 |
| | Optimiser | Adam | Adam | Adam | Adam | Adam |
| | Learning rate | 0.0002 | 0.001 | 0.0002 | 0.0002 | 0.0002 |
| | Epochs | 150 | 1000 | 20 | 500 | 150 |
| | LL Ordering | KDE | KDE | KDE | Corr | Rnd |
| | DRL Ordering | KDE | Corr | Corr | Corr | Rnd |
| TVAE | Batch size | 70 | 70 | 500 | 500 | 70 |
| | Optimiser | Adam | Adam | Adam | Adam | Adam |
| | Learning rate | 0.0002 | 0.0001 | 0.00001 | 0.000005 | 0.0002 |
| | Epochs | 150 | 1500 | 40 | 150 | 150 |
| | Loss factor | 2 | 2 | 4 | 2 | 2 |
| | LL Ordering | KDE | Rnd | Corr | KDE | Rnd |
| | DRL Ordering | Rnd | Rnd | Corr | Corr | KDE |
| GOGGLE | Batch size | 128 | 32 | 128 | 64 | 64 |
| | Optimiser | Adam | Adam | Adam | Adam | Adam |
| | Learning rate | 0.005 | 0.01 | 0.001 | 0.001 | 0.01 |
| | Epochs | 1000 | 500 | 60 | 1000 | 400 |
| | Threshold | 0.1 | 0.2 | 0.2 | 0.1 | 0.2 |
| | Patience | 50 | 50 | 50 | 50 | 50 |
| | LL Ordering | KDE | Rnd | Rnd | Rnd | Rnd |
| | DRL Ordering | Rnd | Rnd | Rnd | Rnd | Rnd |

Additionally, I explored various batch sizes for each model:

- WGAN: $\{64, 128, 256, 510\}$
- TableGAN: $\{128, 256, 510\}$
- CTGAN and TVAE: $\{70, 280, 500\}$

- GOGGLE: $\{32, 64, 128\}$

I did not separately tune the DGM+DRL models, nor the DGM+LL models. However, for each of the DGM+DRL and DGM+LL models, I ran three versions corresponding to three different ways of ordering the variables that decided the order in which the layers (LL and DRL) change the values of the features. More precisely, I tested the following ordering methods: (i) a random (Rnd) ordering, (ii) the correlation (Corr)-based ordering, (iii) the Kernel Density Estimation (KDE)-based ordering, all of which I introduced in Chapter 4. I then selected the best ordering separately for each DGM+DRL and DGM+LL model.

In Table C.1, I report the optimal hyperparameter configurations, which I use in all experiments that involve the DGM, DGM+LL and DGM+DRL models.

# Appendix D

# Linear vs. QFLRA Constraints: Full Results

For completeness, I provide the full results of the quantitative analysis from Section 5.4.3 of Chapter 5. Specifically, I compare the DGM+DRL models (using QFLRA constraints) against the DGM+LL models (using linear constraints) along two axes: (i) background knowledge alignment, and (ii) machine learning efficacy. Recall that the goal of the analysis in Section 5.4.3 was to determine whether the more complex QFLRA constraints help increase the quality of the generated data relative to using only linear constraints.

**Background knowledge alignment.** In Tables D.1, D.2, and D.3, I provide the CVR, sCVC, and CVC, respectively, for the DGM+LL models, according to the two possible types: linear and disjunctive constraints.

**Efficacy.** Tables D.4, D.5, D.6 show the efficacy, along with the standard deviations from the mean, for each DGM+LL model and the corresponding DGM+DRL models on every classification dataset, using the F1, $w$F1, and AUC metrics, respectively. Similarly, in Table D.7, I report MAE and RMSE for each unconstrained DGM+LL model and the corresponding DGM+DRL models on the regression dataset (i.e., House).

Table D.1: Constraint violation rate (CVR) for each DGM+LL model and dataset.

| Constraint Type | Model/Dataset | URL | CCS | LCLD | Heloc | House |
|---|---|---|---|---|---|---|
| Linear | All models + LL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |
| Disjunctive | WGAN+LL | 8.9±3.2 | 51.5±11.2 | 27.0±3.6 | 20.6±6.3 | 100.0±0.0 |
| | TableGAN+LL | 3.6±0.8 | 54.0±17.8 | 11.3±0.9 | 26.6±7.7 | 23.9±2.7 |
| | CTGAN+LL | 7.0±2.6 | 55.7±16.3 | 2.6±1.1 | 2.6±2.4 | 10.8±7.8 |
| | TVAE+LL | 6.8±0.6 | 8.4±2.0 | 5.8±0.8 | 0.0±0.0 | 13.0±12.6 |
| | GOGGLE+LL | 6.5±7.0 | 23.0±10.7 | 81.9±6.5 | 11.5±7.1 | 2.6±2.6 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |

Table D.2: Samplewise constraints violation coverage (sCVC) for each model and dataset.

| Constraint Type | Model/Dataset | URL | CCS | LCLD | Heloc | House |
|---|---|---|---|---|---|---|
| Linear | All models + LL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |
| Disjunctive | WGAN+LL | 0.9±0.4 | 6.9±1.1 | 2.2±0.4 | 4.2±1.4 | 9.1±0.0 |
| | TableGAN+LL | 0.4±0.1 | 6.5±2.3 | 1.0±0.1 | 5.9±1.7 | 2.2±0.3 |
| | CTGAN+LL | 0.7±0.3 | 7.0±2.1 | 0.2±0.1 | 0.5±0.5 | 1.0±0.7 |
| | TVAE+LL | 0.7±0.1 | 0.9±0.2 | 0.4±0.1 | 0.0±0.0 | 1.2±1.2 |
| | GOGGLE+LL | 0.7±0.7 | 2.6±1.4 | 11.2±2.1 | 2.3±1.4 | 0.2±0.2 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |

Table D.3: Constraints violation coverage (CVC) for each model and dataset.

| Constraint Type | Model/Dataset | URL | CCS | LCLD | Heloc | House |
|---|---|---|---|---|---|---|
| Linear | All models + LL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |
| Disjunctive | WGAN+LL | 50.0±0.0 | 40.0±0.0 | 28.0±1.3 | 79.2±1.8 | 13.1±4.1 |
| | TableGAN+LL | 54.0±4.7 | 30.0±0.0 | 27.3±0.0 | 84.8±11.1 | 36.7±0.8 |
| | CTGAN+LL | 52.8±2.3 | 30.0±0.0 | 25.4±2.7 | 41.6±12.8 | 32.7±5.0 |
| | TVAE+LL | 51.2±1.1 | 29.2±1.8 | 22.6±0.6 | 20.8±14.3 | 32.0±4.6 |
| | GOGGLE+LL | 33.6±12.4 | 27.5±5.0 | 46.2±0.0 | 28.0±16.7 | 17.6±1.0 |
| | All models + DRL | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0**±0.0 | **0.0** ±0.0 |

Table D.4: Machine learning efficacy comparison between the DGM+LL models and their DRL counterparts in terms of F1. The performance, along with the standard deviation, is reported for each classification dataset.

| | URL | CCS | LCLD | Heloc |
|---|---|---|---|---|
| WGAN+LL | **0.803**±0.038 | **0.359**±0.096 | 0.183±0.094 | 0.694±0.033 |
| WGAN+DRL | 0.800±0.011 | 0.313±0.127 | **0.197**±0.060 | **0.721**±0.027 |
| TableGAN+LL | 0.612±0.111 | **0.169**±0.044 | 0.232±0.026 | **0.638**±0.061 |
| TableGAN+DRL | **0.619**±0.046 | 0.163±0.079 | **0.269**±0.025 | 0.628±0.083 |
| CTGAN+LL | **0.836**±0.002 | 0.250±0.081 | 0.265±0.040 | 0.729±0.027 |
| CTGAN+DRL | **0.836**±0.004 | **0.288**±0.116 | **0.288**±0.013 | **0.744**±0.020 |
| TVAE+LL | 0.824±0.004 | 0.413±0.057 | 0.158±0.011 | 0.730±0.009 |
| TVAE+DRL | **0.835**±0.009 | **0.467**±0.100 | **0.189**±0.022 | **0.731**±0.009 |
| GOGGLE+LL | **0.787**±0.014 | 0.233±0.180 | 0.284±0.123 | **0.723**±0.018 |
| GOGGLE+DRL | 0.720±0.086 | **0.253**±0.144 | **0.298**±0.153 | 0.698±0.023 |

Table D.5: Machine learning efficacy comparison between the DGM+LL models and their DRL counterparts in terms of $w$F1. The performance, along with the standard deviation, is reported for each classification dataset.

| | URL | CCS | LCLD | Heloc |
|---|---|---|---|---|
| WGAN+LL | 0.799±0.022 | **0.383**±0.092 | 0.330±0.068 | **0.662**±0.021 |
| WGAN+DRL | **0.801**±0.014 | 0.340±0.122 | **0.339**±0.049 | 0.652±0.036 |
| TableGAN+LL | **0.695**±0.071 | **0.203**±0.042 | 0.373±0.017 | **0.633**±0.036 |
| TableGAN+DRL | 0.693±0.028 | 0.196±0.076 | **0.401**±0.018 | 0.628±0.038 |
| CTGAN+LL | **0.820**±0.008 | 0.271±0.083 | 0.392±0.030 | **0.688**±0.010 |
| CTGAN+DRL | 0.815±0.011 | **0.308**±0.118 | **0.409**±0.007 | 0.680±0.011 |
| TVAE+LL | 0.816±0.008 | 0.436±0.055 | 0.310±0.011 | 0.691±0.007 |
| TVAE+DRL | **0.832**±0.014 | **0.487**±0.096 | **0.330**±0.014 | **0.694**±0.006 |
| GOGGLE+LL | **0.749**±0.029 | 0.262±0.173 | **0.310**±0.039 | **0.663**±0.012 |
| GOGGLE+DRL | 0.673±0.039 | **0.281**±0.139 | **0.310**±0.057 | 0.636±0.020 |

Table D.6: Machine learning efficacy comparison between the DGM+LL models and their DRL counterparts in terms of AUC. The performance, along with the standard deviation, is reported for each classification dataset.

| | URL | CCS | LCLD | Heloc |
|---|---|---|---|---|
| WGAN+LL | 0.869±0.014 | 0.857±0.058 | 0.608±0.021 | **0.732**±0.013 |
| WGAN+DRL | **0.875**±0.007 | **0.885**±0.050 | **0.623**±0.023 | 0.717±0.029 |
| TableGAN+LL | **0.868**±0.007 | **0.794**±0.015 | 0.640±0.005 | 0.704±0.030 |
| TableGAN+DRL | 0.865±0.022 | 0.742±0.096 | **0.657**±0.007 | **0.709**±0.011 |
| CTGAN+LL | 0.880±0.007 | **0.959**±0.027 | 0.641±0.015 | **0.755**±0.007 |
| CTGAN+DRL | **0.883**±0.009 | 0.955±0.022 | **0.643**±0.019 | 0.745±0.008 |
| TVAE+LL | 0.878±0.007 | **0.933**±0.036 | 0.633±0.003 | 0.747±0.007 |
| TVAE+DRL | **0.893**±0.010 | 0.926±0.039 | **0.635**±0.002 | **0.752**±0.003 |
| GOGGLE+LL | **0.802**±0.016 | **0.765**±0.084 | 0.554±0.039 | **0.719**±0.005 |
| GOGGLE+DRL | 0.747±0.029 | 0.758±0.091 | **0.563**±0.027 | 0.691±0.039 |

Table D.7: Machine learning efficacy performance comparison between DGM+LL and DGM+DRL models trained on House, using MAE and RMSE.

| | MAE | RMSE |
|---|---|---|
| WGAN+LL | 547638.5±11.4 | 688118.2±11.0 |
| WGAN+DRL | **547637.5**±17.4 | **688118.0**±14.9 |
| TableGAN+LL | **547638.0**±18.9 | **688118.3**±17.9 |
| TableGAN+DRL | 547653.6±22.8 | 688131.4±18.7 |
| CTGAN+LL | **547642.3**±14.6 | **688121.4**±14.5 |
| CTGAN+DRL | 547642.9±18.1 | 688122.1±14.0 |
| TVAE+LL | 547658.1±9.8 | 688137.4±9.4 |
| TVAE+DRL | **547645.4**±31.8 | **688124.6**±27.8 |
| GOGGLE+LL | 547651.9±9.9 | 688129.6±8.4 |
| GOGGLE+DRL | **547633.9**±16.0 | **688115.7**±11.3 |

# Appendix E

# Background Knowledge Alignment: A Qualitative Analysis for DRL

Figure E.1 accompanies Figure 5.6 from Chapter 5 and shows the relevant sample space for the same two constraints from the House dataset: *if the Zipcode is 98004 or 98005 then the Price is greater than 400K USD* and *if the Zipcode is between 98006 and 98008 then the Price exceeds 225K USD*. As we can see, in all cases, the unconstrained DGMs and the DGMs+LL fail to comply with the constraints. Unlike the synthetic data from the unconstrained DGMs, the samples generated using DRL never cross into the areas that mark regions where datapoints violate the constraints and, in addition, their distribution resembles more closely the real data in all five cases.

In addition, I provide a similar comparison but for a different dataset and different constraints. Specifically, I consider the following two constraints from the URL dataset: *if the Number of Subdomains is less than 2 then the Hostname length is less than 30* and *if the Number of Subdomains is less than 3 then the Hostname length is less than 55*. The two constraints capture the relation between the two features (i.e., *Number of Subdomains* and *Hostname length*) and, differently from the two constraints from the House dataset mentioned above, their respective violation space intersects, as shown in red in Figure E.2. Nevertheless, the DGM+DRL models never violate any of the constraints, unlike the unconstrained and DGM+LL models.

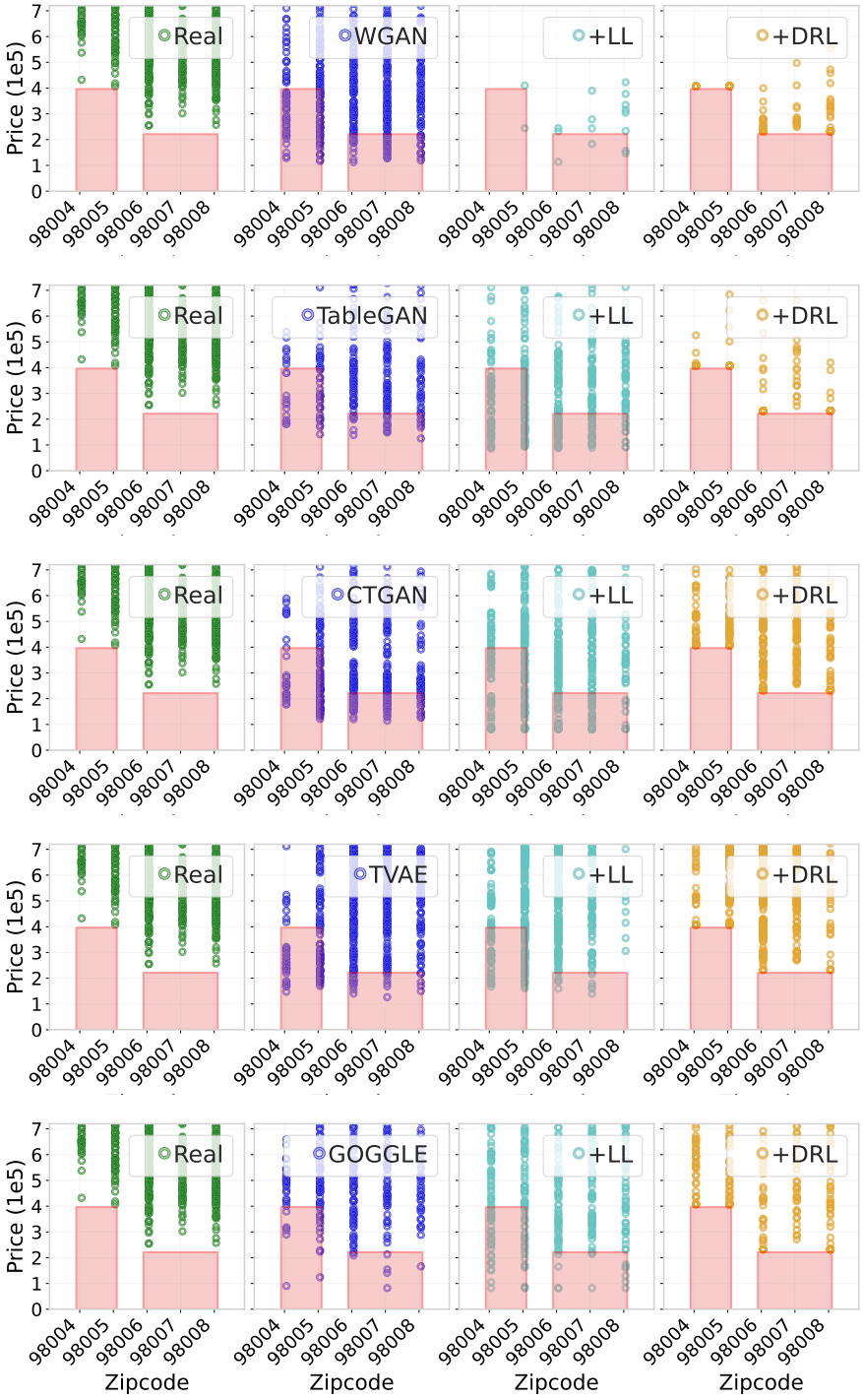



Figure E.1: Comparison of sample distributions between real data and synthetic data generated by the unconstrained DGM models and their corresponding DGM+LL and DGM+DRL models, using the *Zipcode* and *Price* features from the House dataset. In order (from the top to the bottom row), the DGM models used in the plots are: WGAN, TableGAN, CTGAN, TVAE, and GOGGLE. The areas in red indicate regions where samples violate the constraints on the given features.

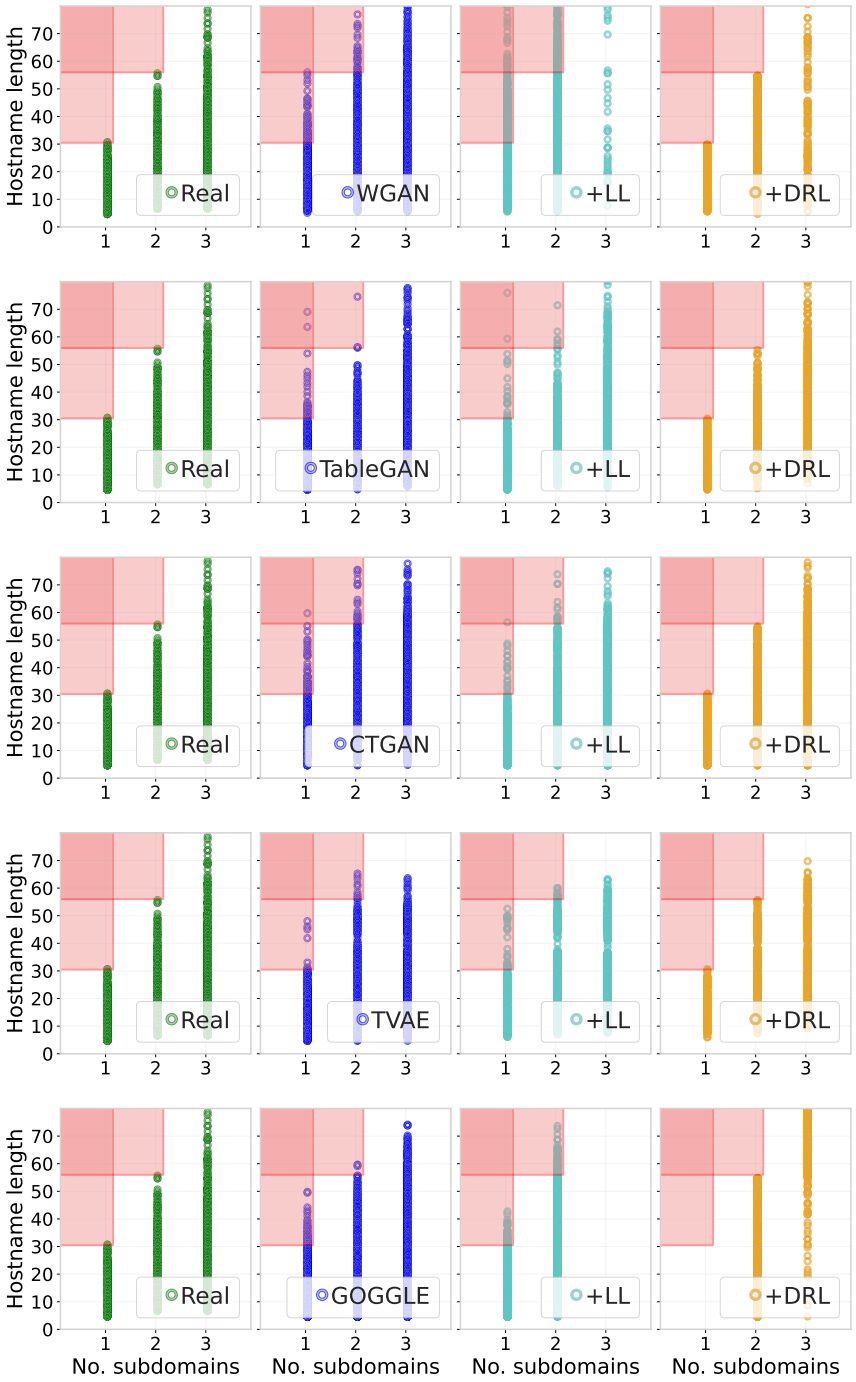



Figure E.2: Comparison of sample distributions between real data and synthetic data generated by the unconstrained DGM models and their corresponding DGM+LL and DGM+DRL models, using the *No. subdomains* and *Hostname length* features from the URL dataset. In order (from the top to the bottom row), the DGM models used in the plots are: WGAN, TableGAN, CTGAN, TVAE, and GOGGLE. The areas in red indicate regions where samples violate the constraints on the given features.

# Appendix F

# Performance on Real Data

Table F.1: Machine learning efficacy scores calculated on the real datasets: URL, WiDS, FSP, LCLD, Heloc, and News.

| | F1 | *w*F1 | AUC | XV | MAE | |
|---|---|---|---|---|---|---|
| URL | 0.884$\pm$0.007 | 0.875$\pm$0.014 | 0.903$\pm$0.009 | - | - | |
| WiDS | 0.383$\pm$0.021 | 0.434$\pm$0.020 | 0.832$\pm$0.009 | - | - | |
| FSP | 0.662$\pm$0.011 | 0.659$\pm$0.009 | 0.848$\pm$0.010 | - | - | |
| LCLD | 0.171$\pm$0.030 | 0.316$\pm$0.013 | 0.645$\pm$0.007 | - | - | |
| Heloc | 0.772$\pm$0.003 | 0.662$\pm$0.011 | 0.707$\pm$0.008 | - | - | |
| News | | - | - | - | -0.001$\pm$0.001 | 3001.1$\pm$55.0 |

Here I present the efficacy of the real data on the datasets used in different Chapters. Such results are important as they provide a reference point and allow for comparing the machine learning efficacy of the synthetic data with the one of the real data.

## F.1 Datasets from Chapters 3 and 4

In Table F.1, I train the same classifiers/regressors (following the protocol described in Sections 3.4.1) on the real data and report (i) the F1-score, weighted F1-score, and AUC for the classification datasets, and (ii) the Explained Variance and the Mean Absolute Error for the regression dataset (i.e., News).

I also compare the machine learning efficacy performance of the synthetic data in Tables B.1-B.4 with that of the real data, from Table F.1:

1. First, the WGAN+LL and CTGAN+LL models often produce synthetic data that leads to strong machine learning efficacy, in some cases approaching the performance of real data, particularly on the WiDS and Heloc datasets.

Table F.2: Machine learning efficacy scores calculated on the real datasets: URL, CCS, LCLD, Heloc, and House.

| | F1 | *w*F1 | AUC | MAE | RMSE |
|---|---|---|---|---|---|
| URL | $0.884_{\pm 0.007}$ | $0.875_{\pm 0.014}$ | $0.903_{\pm 0.009}$ | - | - |
| CCS | $0.529_{\pm 0.011}$ | $0.547_{\pm 0.011}$ | $0.948_{\pm 0.010}$ | | |
| LCLD | $0.171_{\pm 0.030}$ | $0.316_{\pm 0.013}$ | $0.645_{\pm 0.007}$ | - | - |
| Heloc | $0.772_{\pm 0.003}$ | $0.662_{\pm 0.011}$ | $0.707_{\pm 0.008}$ | - | - |
| House | - | - | - | $547655.7_{\pm 38.2}$ | $688133.1_{\pm 30.8}$ |

2. For TableGAN+LL and TVAE+LL, there are several cases where LL helps in bringing the performance of the synthetic data closer to the real data. In particular, for LCLD, the real data get an F1-score 4.8% higher than the unconstrained TableGAN, but the TableGAN+LL model is able to match the real data performance (scoring slightly better than it, i.e., by 0.3%).

3. It is also worth noticing that the gap between real and synthetic performance is reduced the most for the LCLD and WiDS datasets, both of which are highly unbalanced. For instance, CTGAN+LL (with all orderings), CTGAN, WGAN+LL (with most orderings), and WGAN models yield a higher F1-score than the real LCLD data. Similar trends emerge for the other metrics.

4. On the other hand, none of the DGM or the DGM+LL models get as close to the real FSP data. One possible reason is the dataset's small size and multi-class nature, which makes it harder for the DGM models to capture patterns that lead to the correct different targets.

5. Finally, for the News dataset, several models get very close or even yield a better MAE performance (e.g., WGAN+LL using the causal-based ordering).

Overall, integrating LL into the DGMs can improve synthetic data quality to match or exceed real-world machine learning efficacy.

## F.2 Datasets from Chapter 5

I train six classifiers (resp. regressors, as described in Section 5.4.1 of Chapter 5) on the real classification datasets (resp. real regression dataset). Then, in Table F.2, for the classification datasets, I report the average F1-score, weighted F1-score and AUC. For the regression dataset, House, I report the average MAE and RMSE.

# Appendix G

# Evaluation Metrics for Tabular Data Requirements

In this Appendix, I provide the formulae for each of the metrics presented in Table 2.1. I denote by $N$ and $D$ the sampled population size (i.e., the total number of rows in a dataset) and the number of features (i.e., the total number of columns in a dataset), respectively. Moreover, $\mathbb{I}(\cdot)$ is the indicator function, which returns 1 if the condition inside the parentheses is true, and 0 otherwise, I also denote the *true positives*, *true negatives*, *false positives*, and *false negatives* by $TP$, $TN$, $FP$, and $FN$, respectively. The sets $\mathcal{R}$ and $\mathcal{S}$ represent the real and synthetic datasets, respectively.

## Utility

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{G.1}$$

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{G.2}$$

where $\text{Precision} = \frac{TP}{TP+FP}$ and $\text{Recall} = \frac{TP}{TP+FN}$.

$$\text{Weighted F1-score} = \sum_{i=1}^{C} w_i \cdot \text{F1}_i, \tag{G.3}$$

where $C$ is the number of classes (assuming a categorical target feature in a tabular dataset), $\text{F1}_i$ is the F1-score for class $i$, and $w_i$ is the frequency of class $i$ relative to the other classes.

$$\text{Root mean square error (RMSE)} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}, \tag{G.4}$$

where $y_i$ and $\hat{y}_i$ denote the actual (i.e., ground-truth) target value and the predicted target value for the $i$-th sample, respectively.

$$\text{Mean absolute error (MAE)} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|, \tag{G.5}$$

where $y_i$ and $\hat{y}_i$ denote the actual (i.e., ground-truth) target value and the predicted target value for the $i$-th sample, respectively.

$$\text{Explained variance} = 1 - \frac{\text{Var}(\mathbf{y} - \hat{\mathbf{y}})}{\text{Var}(\mathbf{y})}, \tag{G.6}$$

where $\mathbf{y}$ and $\hat{\mathbf{y}}$ represent arrays of actual and predicted values, respectively, and $\text{Var}(\cdot)$ represents the variance of a set of values.

**Alignment**

$$\text{CVR} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(\text{sample } i \text{ violates any constraint}) \tag{G.7}$$

$$\text{CVC} = \frac{\#\text{constraints violated by any of the N samples}}{\#\text{constraints}} \tag{G.8}$$

$$\text{sCVC} = \frac{1}{N}\sum_{i=1}^{N}\frac{\#\text{constraints violated by sample } i}{\#\text{constraints}} \tag{G.9}$$

**Fidelity** In what follows, (i) $F$ represents the cumulative distribution function of a given feature in a dataset $\mathcal{D}$, (ii) $D_{KL}(\cdot||\cdot)$ denotes the Kullback-Leibler (KL) Divergence, measuring how one probability distribution diverges from a second, while $P$ and $Q$ denote the probability distributions of the real and synthetic data, respectively, (iii) $M$ is the mixture (average) distribution between $P$ and $Q$, and (iv) $\theta$ represents the parameters of the model trained to synthesise data.

Further, $\sup_x$ is the supremum (i.e., the least upper bound of a set of values over all possible $x$), and $I_{\mathcal{S}}(i,j)$ represents the mutual information between features $i$ and $j$ in dataset $\mathcal{D}$. Finally, $\rho_{\mathcal{S},ij}$ is the correlation coefficient (e.g., Pearson) between features i and j in dataset $\mathcal{D}$.

1. Feature-wise evaluation metrics:

$$\text{Wasserstein distance} = \int_{-\infty}^{\infty}|F_R(x) - F_S(x)|dx \tag{G.10}$$

$$\text{Kolmogorov-Smirnov test statistic} = \sup_x |F_R(x) - F_S(x)| \tag{G.11}$$

$$\text{Jensen-Shannon divergence} = \frac{D_{KL}(P||M) + D_{KL}(Q||M)}{2} \tag{G.12}$$

$$\text{Total variation distance} = \sum_x \frac{|P(x) - Q(x)|}{2} \tag{G.13}$$

2. Pair-wise evaluation metrics:

$$\text{RMSE differences in mutual information} = \sqrt{\frac{1}{D^2}\sum_{i,j}(I_{\mathcal{R}}(i,j) - I_{\mathcal{S}}(i,j))^2} \tag{G.14}$$

$$\text{MAE differences in mutual information} = \frac{1}{D^2}\sum_{i,j}|I_{\mathcal{R}}(i,j) - I_{\mathcal{S}}(i,j)| \tag{G.15}$$

$$\text{Correlation difference} = \frac{2}{D(D-1)}\sum_{i<j}|\rho_{\mathcal{R},ij} - \rho_{\mathcal{S},ij}| \tag{G.16}$$

3. Joint evaluation metrics:

$$\text{Likelihood fitness score} = \frac{1}{|\mathcal{R}|}\sum_{\mathbf{x}\in\mathcal{R}}\log p_\theta(\mathbf{x}) \tag{G.17}$$

**Privacy**

$$\text{AUCROC (for membership attacks)} = \mathbb{P}(\text{score(member)} > \text{score(non-member)}) \tag{G.18}$$

$$\text{Precision (for attribute disclosure attacks)} = \frac{TP}{TP + FP} \tag{G.19}$$

$$\text{Recall/Sensitivity (for attribute disclosure attacks)} = \frac{TP}{TP + FN} \tag{G.20}$$

$$\text{Distance to the closest record (DCR)} = \min_{\mathbf{x}_r\in\mathcal{R}} \text{dist}(\mathbf{x}_s, \mathbf{x}_r), \text{ for } \mathbf{x}_s \in \mathcal{S} \tag{G.21}$$

$$\text{Nearest neighbour distance ratio (NNDR)} = \frac{\min_{\mathbf{x}_r \in \mathcal{R}} \text{dist}(\mathbf{x}_s, \mathbf{x}_r)}{\min_{\mathbf{x}'_r \in \mathcal{R} \setminus \{\mathbf{x}_r\}} \text{dist}(\mathbf{x}_r, \mathbf{x}'_r)}, \text{ for } \mathbf{x}_s \in \mathcal{S} \tag{G.22}$$

$$\text{k-anonymity} = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{x}_r \in \mathcal{R}} \mathbb{I}(|\{\mathbf{x}' : v_{QI}^{\mathbf{x}'} = v_{QI}^{\mathbf{x}_r}\}| \geq k), \tag{G.23}$$

where $v_{QI}^{\mathbf{x}}$ represents the values of the quasi-identifier attributes (i.e., attributes that can be combined to uniquely identify an individual) for a given sample $\mathbf{x}$.

# Appendix H

# Examples of Constraints

Table H.1: Examples of constraints, expressed as linear inequalities, and their natural language explanations for the URL dataset.

| **Logical constraints** | **Descriptions in natural language** |
|---|---|
| $\text{length\_url} - \text{length\_hostname} \geq 0$ | The total length of the URL must be greater than or equal to the length of its hostname. |
| $\text{length\_url} - 3 \cdot \text{nb\_space} - \text{nb\_dots} - \text{nb\_hyphens} - \text{nb\_at} - \text{nb\_qm} - \text{nb\_and} - \text{nb\_or} - \text{nb\_eq} - \text{nb\_underscore} - \text{nb\_tilde} - \text{nb\_percent} - \text{nb\_slash} - \text{nb\_star} - \text{nb\_colon} - \text{nb\_comma} - \text{nb\_semicolumn} \geq 0$ | The URL's length must be greater than or equal to the sum of the counts of various special characters (., -, @, ? etc.) plus three times the number of spaces (with a single space encoded as "%20"). |
| $\text{length\_url} - 3 \cdot \text{nb\_www} - 4 \cdot \text{nb\_com} - 2 \cdot \text{nb\_dslash} - 4 \cdot \text{http\_in\_path} \geq 0$ | The URL's length must be greater than or equal to a weighted sum of the counts of "www", "com", double slashes ("//"), and "http" in the path. |
| $\text{random\_domain} - \text{shortening\_service} \geq 0$ | This implies that if a URL is identified as a shortening service, it must also be classified as having a random domain. |

## H.1 Linear Constraints

Tables H.1 and H.2 show examples of constraints expressed as linear inequalities for the URL and WiDS datasets, respectively.

## H.2 QFLRA Constraints

Table H.3 show examples of constraints expressed as disjunctive inequalities (i.e., QFLRA formulae) for the House dataset.

Table H.2: Examples of constraints, expressed as linear inequalities, and their natural language explanations for patient vital signs from the WiDS dataset.

| **Logical constraints** | **Descriptions in natural language** |
|---|---|
| $\text{map\_apache} - \text{resprate\_apache} \geq 0$ | The APACHE score component for mean arterial pressure must be greater than or equal to the score component for respiratory rate. |
| $\text{d1\_heartrate\_max} - \text{d1\_heartrate\_min} \geq 0$ | Within the first 24 hours of ICU stay, the patient's maximum heart rate cannot be less than the minimum. |
| $\text{d1\_mbp\_max} - \text{d1\_mbp\_min} \geq 0$ | Within the first 24 hours of ICU stay, the patient's maximum mean blood pressure cannot be less than the minimum. |
| $\text{d1\_temp\_max} - \text{d1\_temp\_min} \geq 0$ | Within the first 24 hours of ICU stay, the patient's maximum temperature cannot be less than the minimum. |
| $\text{h1\_heartrate\_max} - \text{h1\_heartrate\_min} \geq 0$ | During the first hour of ICU stay, the patient's maximum heart rate cannot be less than the minimum. |
| $\text{h1\_temp\_max} - \text{h1\_temp\_min} \geq 0$ | During the first hour of ICU stay, the patient's maximum temperature cannot be less than the minimum. |
| $\text{d1\_calcium\_max} - \text{d1\_calcium\_min} \geq 0$ | Within the first 24 hours of ICU stay, the patient's maximum serum calcium level cannot be less than the minimum. |
| $\text{d1\_creatinine\_max} - \text{d1\_creatinine\_min} \geq 0$ | Within the first 24 hours of ICU stay, the patient's maximum serum creatinine level cannot be less than the minimum. |
| $\text{d1\_glucose\_max} - \text{d1\_glucose\_min} \geq 0$ | Within the first 24 hours of ICU stay, the patient's maximum serum glucose level cannot be less than the minimum. |
| $\text{d1\_haemaglobin\_max} - \text{d1\_haemaglobin\_min} \geq 0$ | Within the first 24 hours of ICU stay, the patient's maximum haemoglobin level cannot be less than the minimum. |
| $\text{d1\_haematocrit\_max} - \text{d1\_haematocrit\_min} \geq 0$ | Within the first 24 hours of ICU stay, the patient's maximum haematocrit level cannot be less than the minimum. |

## H.3 Propositional Logic Constraints

Table H.4 show examples of constraints expressed as propositional logic formulae from the ROAD-R dataset.

Table H.3: Examples of constraints, expressed as QFLRA formulae, and their natural language explanations for the House dataset.

| **Logical constraints** | **Descriptions in natural language** |
|---|---|
| $\text{sqft\_lv} - \text{sqft\_abv} - \text{sqft\_bsmt} \geq 0$<br>$-\text{sqft\_lv} + \text{sqft\_abv} + \text{sqft\_bsmt} \geq 0$ | The living area must equal the sum of the area above ground and the area of the basement, where area is measured in sq. ft. |
| $\neg(\text{yr\_rnv} \geq 0) \vee (\text{yr\_rnv} - \text{yr\_blt} > 0)$ | If a house was renovated, the year of renovation must be after the year it was built. |
| $\neg(\text{prc} \geq 5000000) \vee (\text{sqft\_lv} \geq 2000)$ | If the price of a house is greater than or equal to \$5,000,000, its living area must be at least 2,000 sq. ft. |
| $(\text{prc} \geq 100000) \vee \neg(\text{beds} \geq 1) \vee (\text{beds} > 1) \vee (-\text{sqft\_lv} > -800)$ | If the price of a house is less than \$100,000 and it has exactly one bedroom, the living area must be less than 800 sq. ft. |
| $\neg(\text{zip} \geq 98004) \vee (\text{zip} > 98005) \vee (\text{prc} \geq 400000)$ | If the zip code is between 98004 and 98005 (inclusive), the price must be at least \$400,000. |
| $\neg(\text{zip} \geq 98006) \vee (\text{zip} > 98008) \vee (\text{prc} \geq 225000)$ | If the zip code is between 98006 and 98008 (inclusive), the price must be at least \$225,000. |
| $\neg(\text{sqft\_lv} \geq 4000) \vee \neg(\text{baths} \geq 2) \vee (\text{beds} \geq 3)$ | If the living area is at least 4,000 sq. ft. and there are at least 2 bathrooms, there must be at least 3 bedrooms. |

Table H.4: Examples of logical constraints and their natural language explanations from Tables 9-11 of (Giunchiglia et al., 2023b).

| **Logical constraints** | **Descriptions in natural language** |
|---|---|
| {not Mobike, not Bus} | A motorbike cannot be a bus |
| {not TL, not TurLft} | A traffic light cannot turn left |
| {not Wait2X, not Ovtak} | An agent cannot wait to cross and overtake |
| {Ped, not PushObj} | If an agent pushes an object then it is a pedestrian |
| {PushObj, not Ped, MovAway, MovTow, Mov, Stop, TurLft, TurRht, Wait2X, XingFmLft, XingFmRht, Xing} | A pedestrian can only push objects, move away, etc. |
| {VehLane, OutgoLane, OutgoCycLane, IncomLane, IncomCycLane, Pav, LftPav, RhtPav, Jun, XingLoc, BusStop, Parking, TL, OthTL} | Every agent but traffic lights must have a position |

# Bibliography

Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2016.

Kareem Ahmed, Tao Li, Thy Ton, Quan Guo, Kai-Wei Chang, Parisa Kordjamshidi, Vivek Srikumar, Guy Van den Broeck, and Sameer Singh. Pylon: A PyTorch framework for learning with constraints. In *Advances in Neural Information Processing Systems, Competitions and Demonstrations Track*. PMLR, 2022a.

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

Kareem Ahmed, Eric Wang, Kai-Wei Chang, and Guy Van den Broeck. Neuro-symbolic entropy regularization. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, 2022c.

Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *CoRR*, abs/1701.07875, 2017.

Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic Tensor Networks. *Artificial Intelligence*, 303, 2022.

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

Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *Transactions on Neural Networks and Learning Systems*, 35, 2024.

Massimo Buscema. MetaNet*: The theory of independent judges. *Substance Use & Misuse*, 33, 1998.

Diego Calanzone, Stefano Teso, and Antonio Vergari. Logically consistent language models via neuro-symbolic integration. In *Proceedings of International Conference on Learning Representations*, 2025.

Ramiro Camino, Christian A. Hammerschmidt, and Radu State. Generating multi-categorical samples with generative adversarial networks. *CoRR*, abs/1807.01202, 2018.

Tommaso Carraro. LTNtorch: PyTorch implementation of Logic Tensor Networks, 2022.

Zhengping Che, Yu Cheng, Shuangfei Zhai, Zhaonan Sun, and Yan Liu. Boosting deep learning risk prediction with generative adversarial networks for electronic health records. In *Proceedings of International Conference on Data Mining*, 2017.

Dingfan Chen, Ning Yu, Yang Zhang, and Mario Fritz. GAN-Leaks: A taxonomy of membership inference attacks against generative models. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2020.

Haipeng Chen, Sushil Jajodia, Jing Liu, Noseong Park, Vadim Sokolov, and V. S. Subrahmanian. FakeTables: Using GANs to generate functional dependency preserving tables with bounded real data. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2019.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F. Stewart, and Jimeng Sun. Generating multi-label discrete patient records using generative adversarial networks. In *Proceedings of Proceedings of Machine Learning for Health Care Conference*, 2017.

David R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20, 1958.

Alessandro Daniele, Tommaso Campari, Sagar Malhotra, and Luciano Serafini. Deep symbolic learning: Discovering symbols and rules from perceptions. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2023a.

Alessandro Daniele, Emile van Krieken, Luciano Serafini, and Frank van Harmelen. Refining neural network predictions using background knowledge. *Machine Learning Journal*, 112, 2023b.

Tirtharaj Dash, Sharad Chitlangia, Aditya Ahuja, and Ashwin Srinivasan. A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports*, 12, 2022.

Artur d'Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11, 1999.

Artur S. d'Avila Garcez, Marco Gori, Luís C. Lamb, Luciano Serafini, Michael Spranger, and Son N. Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *CoRR*, abs/1905.06088, 2019.

Maria F. Davila R., Sven Groen, Fabian Panse, and Wolfram Wingerath. Navigating tabular data synthesis research understanding user needs and tool capabilities. *ACM SIGMOD Record*, 53, 2025.

David Debot, Pietro Barbiero, Francesco Giannini, Gabriele Ciravegna, Michelangelo Diligenti, and Giuseppe Marra. Interpretable concept-based memory reasoning. In *Advances in Neural Information Processing Systems*, 2024.

Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113, 1999.

Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7, 2006.

Michelangelo Diligenti, Marco Gori, Marco Maggini, and Leonardo Rigutini. Bridging logic and kernel machines. *Machine Learning*, 2012.

Michelangelo Diligenti, Marco Gori, and Claudio Sacca. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244, 2017a.

Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. Integrating prior knowledge into deep learning. In *Proceedings of International Conference on Machine Learning and Applications*, 2017b.

Ivan Donadello, Luciano Serafini, and Artur d'Avila Garcez. Logic Tensor Networks for semantic image interpretation. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2017.

Paolo Dragone, Stefano Teso, and Andrea Passerini. Neuro-symbolic constraint programming for structured prediction. In *Proceedings of International Joint Conference on Learning & Reasoning, International Workshop on Neural-Symbolic Learning and Reasoning*, 2021.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, 2019.

Cynthia Dwork. Differential privacy. In *Proceedings of International Colloquium on Automata, Languages and Programming*, 2006.

Salijona Dyrmishi, Mihaela Cătălina Stoian, Maxime Cordy, Thomas Lukasiewicz, and Eleonora Giunchiglia. Deep generative models as an adversarial attack strategy for tabular machine learning. In *Proceedings of International Conference on Machine Learning and Cybernetics*, 2024.

Artur d'Avila Garcez and Luis C. Lamb. Neurosymbolic AI: The 3rd wave. *Artificial Intelligence Review*, 2023.

Justin Engelmann and Stefan Lessmann. Conditional Wasserstein GAN-based oversampling of tabular data for imbalanced learning. *Expert Systems with Applications*, 174, 2021.

Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *CoRR*, abs/1706.02633, 2017.

Julius Farkas. Theorie der einfachen ungleichungen. *Journal für die reine und angewandte Mathematik*, 124, 1902.

Kelwin Fernandes, Pedro Vinagre, Paulo Cortez, and Pedro Sernadela. Online News Popularity. UCI Machine Learning Repository, 2015.

Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. DL2: Training and querying neural networks with logic. In *Proceedings of International Conference on Machine Learning*, 2019.

Eleonora Giunchiglia and Thomas Lukasiewicz. Coherent hierarchical multi-label classification networks. In *Advances in Neural Information Processing Systems*, 2020.

Eleonora Giunchiglia and Thomas Lukasiewicz. Multi-label classification neural networks with hard logical constraints. *Journal of Artificial Intelligence Research*, 72, 2021.

Eleonora Giunchiglia, Mihaela Cătălina Stoian, and Thomas Lukasiewicz. Deep learning with logical constraints. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2022.

Eleonora Giunchiglia, Fergus Imrie, Mihaela van der Schaar, and Thomas Lukasiewicz. Machine learning with requirements: a manifesto. *CoRR*, abs/2304.03674, 2023a.

Eleonora Giunchiglia, Mihaela Cătălina Stoian, Salman Khan, Fabio Cuzzolin, and Thomas Lukasiewicz. ROAD-R: The autonomous driving dataset for learning with requirements. *Machine Learning*, 112, 2023b.

Eleonora Giunchiglia, Alex Tatomir, Mihaela Cătălina Stoian, and Thomas Lukasiewicz. CCN+: A neuro-symbolic framework for deep learning with requirements. *International Journal of Approximate Reasoning*, 171, 2024.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *Advances in Neural Information Processing Systems*, 2004.

Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37, 2020.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, 2017.

Shlomi Hacohen, Oded Medina, and Shraga Shoval. Autonomous driving: A survey of technological gaps using Google Scholar and Web of Science trend analysis. *IEEE Transactions on Intelligent Transportation Systems*, 23, 2022.

Hui Han, Wenyuan Wang, and Binghuan Mao. Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In *Proceedings of International Conference on Advances in Intelligent Computing*, 2005.

Peiyi Han, Wen Xu, Wanyu Lin, Jiahao Cao, Chuanyi Liu, Shaoming Duan, and Haifeng Zhu. C3-TGAN- controllable tabular data synthesis with explicit correlations and property constraints. *TechRxiv*, 10.36227/techrxiv.24249643, 2023.

Abdelhakim Hannousse and Salima Yahiouche. Towards benchmark datasets for machine learning based website phishing detection: An experimental study. *Engineering Applications of Artificial Intelligence*, 104, 2021.

Lasse Hansen, Nabeel Seedat, Mihaela van der Schaar, and Andrija Petrovic. Reimagining synthetic tabular data generation through data-centric AI: A comprehensive benchmark. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2023.

Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of Conference on Computer Vision and Pattern Recognition*, 2016.

Geoffrey Hinton. Lecture notes in neural networks for machine learning, 2014.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18, 2006.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

Tin Kam Ho. Random decision forests. In *Proceedings of International Conference on Document Analysis and Recognition*, 1995.

Nicholas Hoernle, Rafael-Michael Karampatsis, Vaishak Belle, and Kobi Gal. MultiplexNet: Towards fully satisfied logical constraints in neural networks. In *Proceedings of Association for the Advancement of Artificial Intelligence*, 2022.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. In *Advances in Neural Information Processing Systems*, 2021.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *Proceedings of Annual Meeting of the Association for Computational Linguistics*, 2016a.

Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, and Eric Xing. Deep neural networks with massive learned knowledge. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2016b.

Yuxiu Hua, Zhifeng Zhao, Zhiming Liu, Xianfu Chen, Rongpeng Li, and Honggang Zhang. Traffic prediction based on random connectivity in deep learning with long short-term memory. In *Proceedings of Vehicular Technology Conference (VTC-Fall)*, 2018.

Yu Huang and Yue Chen. Autonomous driving with deep learning: A survey of state-of-art technologies. *CoRR*, abs/2006.06091, 2020.

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, 2024.

James Jordon, Jinsung Yoon, and Mihaela van der Schaar. PATE-GAN: generating synthetic data with differential privacy guarantees. In *Proceedings of International Conference on Learning Representations*, 2019.

James Jordon, Lukasz Szpruch, Florimond Houssiau, Mirko Bottarelli, Giovanni Cherubin, Carsten Maple, Samuel N. Cohen, and Adrian Weller. Synthetic data – what, why and how? *CoRR*, abs/2205.03257, 2022.

Salman Khan, Izzeddin Teeti, Reza Javanmard Alitappeh, Mihaela Cătălina Stoian, Eleonora Giunchiglia, Gurkirt Singh, Andrew Bradley, and Fabio Cuzzolin. ROAD-Waymo: Action awareness at scale for autonomous driving. *CoRR*, abs/2411.01683, 2024.

Jayoung Kim, Jinsung Jeon, Jaehoon Lee, Jihyeon Hyeong, and Noseong Park. OCT-GAN: Neural ODE-based conditional tabular GANs. In *Proceedings of Web Conference*, 2021.

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

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations*, 2015.

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. In *Proceedings of International Conference on Learning Representations*, 2014.

Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. TabDDPM: Modelling tabular data with diffusion models. In *Proceedings of International Conference on Machine Learning*, 2023.

Jan Krajícek. Discretely ordered modules as a first-order extension of the cutting planes proof system. *Journal of Symbolic Logic*, 63, 1998.

Anton Danholt Lautrup, Tobias Hyrup, Arthur Zimek, and Peter Schneider-Kamp. Systematic review of generative modelling tools and utility metrics for fully synthetic tabular data. *ACM Computing Surveys*, 57, 2024.

Chaejeong Lee, Jayoung Kim, and Noseong Park. CoDi: co-evolving contrastive diffusion models for mixed-type tabular synthesis. In *Proceedings of International Conference on Machine Learning*, 2023.

Jaehoon Lee, Jihyeon Hyeong, Jinsung Jeon, Noseong Park, and Jihoon Cho. Invertible tabular GANs: Killing two birds with one stone for tabular data synthesis. In *Advances in Neural Information Processing Systems*, 2021.

Tao Li and Vivek Srikumar. Augmenting neural networks with first-order logic. In *Proceedings of Annual Meeting of the Association for Computational Linguistics*, 2019.

Zenan Li, Zehua Liu, Yuan Yao, Jingwei Xu, Taolue Chen, Xiaoxing Ma, and Jian Lü. Learning with logical constraints but without shortcut satisfaction. In *Proceedings of International Conference on Learning Representations*, 2023.

Luca Di Liello, Pierfrancesco Ardino, Jacopo Gobbi, Paolo Morettin, Stefano Teso, and Andrea Passerini. Efficient generation of structured objects with constrained adversarial networks. In *Advances in Neural Information Processing Systems*, 2020.

Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. PacGAN: The power of two samples in generative adversarial networks. In *Advances in Neural Information Processing Systems*, 2018.

Tennison Liu, Zhaozhi Qian, Jeroen Berrevoets, and Mihaela van der Schaar. GOGGLE: Generative modelling for tabular data by learning relational structure. In *Proceedings of International Conference on Learning Representations*, 2022.

Junwei Ma, Apoorv Dankar, George Stein, Guangwei Yu, and Anthony Caterini. TabPFGen – tabular data generation with tabPFN. In *Advances in Neural Information Processing Systems, Second Table Representation Learning Workshop*, 2023.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, 2018.

Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. LYRICS: A general interface layer to integrate logic inference and deep learning. In *Proceedings of Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD*, 2019.

Karen Matthys, Meredith Lee, NehaGoel, Sharada Kalanidhi, and Valerie Vani M. WiDS Datathon 2021, 2021.

Ryan McKenna, Brett Mullins, Daniel Sheldon, and Gerome Miklau. AIM: an adaptive and iterative mechanism for differentially private synthetic data. *Proceedings of VLDB Endowment*, 2022.

George Metcalfe. Fundamentals of fuzzy logics, 2005.

Pasquale Minervini and Sebastian Riedel. Adversarially regularising neural NLI models to integrate logical background knowledge. In *Proceedings of Conference on Computational Natural Language Learning*, 2018.

Pasquale Minervini, Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Adversarial sets for regularising neural link predictors. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, 2017.

Eleonora Misino, Giuseppe Marra, and Emanuele Sansone. VAEL: Bridging variational Autoencoders and probabilistic logic programming. In *Advances in Neural Information Processing Systems*, 2022.

Yatin Nandwani, Abhishek Pathak, Mausam, and Parag Singla. A primal dual formulation for deep learning with constraints. In *Advances in Neural Information Processing Systems*, 2019.

Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit MLE: Backpropagating through discrete exponential family distributions. In *Advances in Neural Information Processing Systems*, 2021.

Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. In *Proceedings of International Conference on Learning Representations*, 2017.

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

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.

Connor Pryor, Charles Dickens, Eriq Augustine, Alon Albalak, William Yang Wang, and Lise Getoor. NeuPSL: Neural probabilistic soft logic. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2023.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2007.

Luc De Raedt, Andrea Passerini, and Stefano Teso. Learning constraints from examples. In *Proceedings of Association for the Advancement of Artificial Intelligence*, 2018.

Luc De Raedt, Sebastijan Dumancic, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2020.

Jerry Reiter. Using cart to generate partially synthetic, public use microdata. *Journal of Official Statistics*, 21, 2005.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Mrinmaya Sachan, Kumar Avinava Dubey, Tom M. Mitchell, Dan Roth, and Eric P. Xing. Learning pipelines with limited data and domain knowledge: A study in parsing physics problems. In *Advances in Neural Information Processing Systems*, 2018.

Timur Sattarov, Marco Schreyer, and Damian Borth. FinDiff: Diffusion models for financial tabular data generation. In *Proceedings of ACM International Conference on AI in Finance*, 2023.

Neil Savage. Synthetic data could be better than real data. *Nature*, 2023.

Robert E. Schapire. Explaining AdaBoost. In *Empirical inference*. Springer, 2013.

Luciano Serafini and Artur d'Avila Garcez. Logic Tensor Networks: Deep learning and logical reasoning from data and knowledge. In *Proceedings of Workshop Series on Neural-Symbolic Learning and Reasoning*, 2016.

Thibault Simonetto, Salijona Dyrmishi, Salah Ghamizi, Maxime Cordy, and Yves Le Traon. A unified framework for adversarial attack and defense in constrained feature space. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2022.

Gurkirt Singh, Stephen Akrigg, Manuele Di Maio, Valentina Fontana, Reza Javanmard Alitappeh, Salman Khan, Suman Saha, Kossar Jeddisaravi, Farzad Yousefi, Jacob Culley, Tom Nicholson, Jordan Omokeowa, Stanislao Grazioso, Andrew Bradley, Giuseppe Di Gironimo, and Fabio Cuzzolin. ROAD: The road event awareness dataset for autonomous driving. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45, 2023.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of International Conference on Machine Learning*, 2015.

Akash Srivastava, Lazar Valkov, Chris Russell, Michael U. Gutmann, and Charles Sutton. VEEGAN: Reducing mode collapse in GANs using implicit variational learning. In *Advances in Neural Information Processing Systems*, 2017.

Akash Srivastava, Kristjan Greenewald, and Farzaneh Mirzazadeh. Bregmn: scaled-bregman generative modeling networks. *CoRR*, 2019.

Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *Proceedings of Conference on Artificial Intelligence*, 2017.

Mihaela Cătălina Stoian. Deep learning with requirements in the real world. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2024.

Mihaela Cătălina Stoian and Eleonora Giunchiglia. Beyond the convexity assumption: Realistic tabular data generation under quantifier-free real linear constraints. In *Proceedings of International Conference on Learning Representations*, 2025.

Mihaela Cătălina Stoian, Eleonora Giunchiglia, and Thomas Lukasiewicz. Exploiting T-norms for deep learning in autonomous driving. In *Proceedings of International Workshop on Neural-Symbolic Learning and Reasoning*, 2023.

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. In *Proceedings of International Conference on Learning Representations*, 2024a.

Mihaela Cătălina Stoian, Alex Tatomir, Thomas Lukasiewicz, and Eleonora Giunchiglia. PiShield: a PyTorch package for learning with requirements. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2024b.

Mihaela Cătălina Stoian, Salijona Dyrmishi, Maxime Cordy, Thomas Lukasiewicz, and Eleonora Giunchiglia. Generalised linearly-constrained layers for deep generative modelling. Manuscript in preparation, 2025.

Mihaela Cătălina Stoian, Eleonora Giunchiglia, and Thomas Lukasiewicz. A survey on deep learning approaches for tabular data generation: Utility, alignment, fidelity, privacy, diversity, and beyond. *Transactions on Machine Learning Research*, 2026.

Namjoon Suh, Xiaofeng Lin, Din-Yin Hsieh, Merhdad Honarkhah, and Guang Cheng. AutoDiff: combining Auto-encoder and Diffusion model for tabular data synthesizing. *CoRR*, abs/2310.15479, 2023.

Reihaneh Torkzadehmahani, Peter Kairouz, and Benedict Paten. DP-CGAN: differentially private synthetic data and label generation. *CoRR*, abs/2001.09700, 2020.

Geoffrey Towell and Jude Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70, 1994.

Boris van Breugel, Trent Kyono, Jeroen Berrevoets, and Mihaela van der Schaar. DECAF: generating fair synthetic data using causally-aware generative networks. In *Advances in Neural Information Processing Systems*, 2021.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, 2008.

Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy implications. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning*, 2020.

Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence*, 302, 2022.

Emile van Krieken, Thiviyan Thanapalasingam, Jakub M. Tomczak, Frank Van Harmelen, and Annette Ten Teije. A-neSI: A scalable approximate method for probabilistic neurosymbolic inference. In *Advances in Neural Information Processing Systems*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Mark Vero, Mislav Balunovic, and Martin Vechev. CuTS: Customizable tabular synthetic data generation. In *Proceedings of International Conference on Machine Learning*, 2024.

Zhiqiang Wan, Yazhou Zhang, and Haibo He. Variational autoencoder based synthetic data generation for imbalanced learning. In *Proceedings of IEEE Symposium Series on Computational Intelligence*, 2017.

D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), 1997.

Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14, 2008.

Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion GANs. In *Proceedings of International Conference on Learning Representations*, 2022.

Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739*, 2018.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of International Conference on Machine Learning*, 2018.

Lei Xu and Kalyan Veeramachaneni. Synthesizing tabular data using generative adversarial networks. *CoRR*, abs/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.

Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing neural networks into answer set programming. In *Proceedings of International Joint Conference on Artificial Intelligence*, 2020.

Jinsung Yoon, Lydia N. Drumright, and Mihaela van der Schaar. Anonymization through data synthesis using generative adversarial networks (ADS-GAN). *IEEE Journal of Biomedical and Health Informatics*, 24, 2020.

Yue Yu, Jie Chen, Tian Gao, and Mo Yu. DAG-GNN: DAG structure learning with graph neural networks. In *Proceedings of International Conference on Machine Learning*, 2019.

Hanbing Zhang, Yinan Jing, Fei Zhang, Zhixin Li, X. Sean Wang, Zhenqiang Chen, and Cheng Lv. TabTransGAN: A hybrid approach integrating GAN and transformer architectures for tabular data synthesis. *Information Processing & Management*, 62, 2025.

Hengrui Zhang, Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Xiao Qin, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Mixed-type tabular data synthesis with score-based diffusion in latent space. In *Proceedings of International Conference on Learning Representations*, 2024.

Yishuo Zhang, Nayyar A. Zaidi, Jiahui Zhou, and Gang Li. GANBLR: A tabular data generation model. In *Proceedings of International Conference on Data Mining*, 2021.

Zilong Zhao, Aditya Kunar, Robert Birke, and Lydia Y. Chen. CTAB-GAN: Effective table data synthesizing. In *Proceedings of Asian Conference on Machine Learning*, 2021.

Zilong Zhao, Aditya Kunar, Robert Birke, and Lydia Y. Chen. CTAB-GAN+: Enhancing tabular data synthesis. *CoRR*, abs/2204.00401, 2022.

Zilong Zhao, Robert Birke, and Lydia Y. Chen. TabuLa: Harnessing language models for tabular data synthesis. In *Proceedings of PAKDD Advances in Knowledge Discovery and Data Mining*, 2025.