IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Estimand-Agnostic Causal Query Estimation with Deep Causal Graphs

ÁLVARO PARAFITA and JORDI VITRIÀ
Departament de Matemàtica i Informàtica, Universitat de Barcelona, Spain

Corresponding author: Álvaro Parafita (e-mail: parafita.alvaro@ub.edu).

This work has been partially funded by projects RTI2018-095232-B-C21 (MINECO/FEDER, UE) and 2017SGR1742 (Generalitat de Catalunya).

ABSTRACT Causal Queries are usually estimated by means of an estimand, a formula consisting of observational terms that can be computed using passive data. Each query results in a different formula, which makes estimand-based methods extremely ad-hoc. In this work, we propose an estimand-agnostic framework capable of computing any identifiable causal query on an arbitrary Causal Graph (even in the presence of latent confounders) with only one general model. We provide multiple implementations of this general framework that leverage the expressive power of Neural Networks and Normalizing Flows to model complex distributions, and we derive estimation procedures for all kinds of observational, interventional and counterfactual queries, valid for any kind of graph for which the query is identifiable. Finally, we test our techniques in a modelling setting and an estimation benchmark to show how, despite being a query-agnostic framework, it can compete with query-specific models. Our proposal includes an open-source library that allows easy application and extension of our techniques for researchers and practitioners alike.

INDEX TERMS Causality, Structural Causal Model, Causal Query Estimation, Counterfactuals

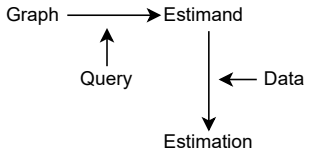
I. INTRODUCTION AND RELATED WORK

Answering causal queries, such as "What is the recovery rate when administering the treatment?", traditionally required a randomized experiment (**interventional data**), where participants are randomly assigned a treatment, thereby measuring the treatment effect while isolating other causes of recovery. This is not always feasible (due to ethical or economic concerns, for example), so an alternative approach is required, which consists of using passively-obtained datasets (**observational data**) consisting of samples that were *naturally* assigned a treatment depending on other factors combined with Causal Query Estimators. The theory is well established ([5], [7], [24], [26]) but its practical solutions are too specific and ad-hoc to the problem and query at hand, which makes them hard to apply to real-world scenarios. Our work focuses on defining an alternative general framework capable of answering any (identifiable) causal queries using one single model, always trained in the same way.

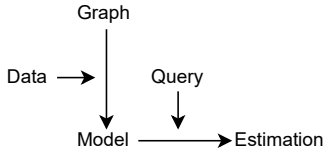
Let us consider an *observational* dataset D, consisting of N i.i.d. samples. Let D be a list of patients with variables: X, each patient's health history; T, whether a certain treatment was administered; and Y, their eventual recovery. From these samples we can extract descriptive statistics (**observational queries**), such as the recovery rate for the patients that were given the treatment ($P(Y \mid T = 1)$). This is *not* the same as the treatment effect on the recovery rate, however, as any confounding factor in X that affects both T and Y (e.g., the severity of the symptoms, which influences the choice of treatment) would bias the result. Assuming that dataset D resulted from an underlying Data Generating Process (DGP) $\mathcal{M}$, which generated its samples, if we also had access to $\mathcal{M}$, it would be possible to answer **interventional queries** such as the aforementioned treatment effect by replicating the process to impose a forced choice of treatment. Naturally, we do not have access to the DGP in most real-world scenarios, but Causal Theory is able to circumvent this problem.

Causal Query Estimation is the field concerned with defining *estimators*, i.e. methods to answer Causal Queries only through *observational data*. This is tackled from two different perspectives: Potential Outcomes [27] or Causal Graphs [23]. We focus on the latter, defining which pairs of variables are causally connected, resulting in a Causal Graph. This graph informs which queries can be answered, and the procedures needed to do so.

From this perspective, when attempting to answer a Causal Query (be it interventional, "What is the expected survival



(a) Estimand-based approach: for every identifiable query, derive an estimand and use it to estimate the query by using data.



(b) Estimand-agnostic approach: create a model from a graph and data, and then use it to estimate any identifiable query.

FIGURE 1: Comparison between estimand-based and estimand-agnostic approaches for Causal Query Estimation.

rate when giving the treatment?", or counterfactual, "What would my salary be if I were a man?"), practitioners are presented with a wide range of methods, each concerned with a particular kind of query and a specific graph structure. This is due to the fact that most of these methods operate with an **estimand**, a formula to transform a causal query into observational terms that can be estimated with data. Hence, given a new query, a specific estimand is extracted and models are defined around it, which is non-trivial for complex expressions. Since estimands depend on the Causal Graph underlying the data and the Causal Query to be estimated, **estimand-based approaches are extremely ad-hoc**: for every query and every graph structure, a new estimand is derived and a new model must be trained. Not only that, but were we to formulate a different query on the same causal structure, we might need to redefine and train a new model.

In response to this problem, we can consider **estimand-agnostic approaches**. These are normally based on the concept of Structural Causal Models (SCMs) [23], which represent a graph with each variable as a function of its parents and a noise signal. By learning a proxy-SCM that follows the same causal structure as the underlying DGP and training it towards modelling the observational distribution, we can then use this SCM to answer causal queries, provided that these are *identifiable* (that is, reliably estimated only through the observational distribution). As such, estimand-agnostic frameworks train once per graph/dataset, and any identifiable queries can later be answered by that single model. The two approaches are summarized in Fig. 1.

SCM-based approaches have been studied previously, beginning with Wright's [38] first SCM, which consisted of linear equations. However, their modelling capabilities have typically been limited by simplistic architectures or restrictive distributions (e.g., Bernoulli, Normal). Since a proxy-SCM needs to model the observational distribution of the underlying DGP in order to correctly estimate causal queries, this was not a feasible approach for real-world data until very recently, when the latest advances in density estimation provided by Deep Learning strategies were adapted to the field of Causal Query Estimation. Some of such strategies employ Generative Adversarial Networks (GANs), with examples including CausalGAN [10] or Variational Autoencoders (VAEs), such as CEVAE [15] or VACA [28]. Adopting a different perspective focused on modelling each node's distribution through a network function, we can mention two parallel works that share some aspects with our techniques. Pawlowski *et al*. [22] proposed Deep SCMs (DSCMs), powered with Normalizing Flows for continuous multidimensional variables, applied to image generation. Xia *et al*. [39] followed a similar approach and proposed Neural Causal Models (NCMs), but their work is concerned with theoretical aspects of proxy-SCM estimation and is currently limited to discrete datasets in the examples they provide.

Although these works suggest several avenues for tackling the problem of estimand-agnostic estimation, we find that they lack several desirable aspects for such a system. Let us define the following **desiderata for a practitioner-ready, Deep Learning powered, estimand-agnostic framework**:

1) **Explicit Likelihood**: being able to estimate likelihood queries (e.g., $P(Y \mid Z)$), with or without interventions or conditioning terms. In addition to estimating likelihoods of variables, this also helps with conditional sampling, as we will see in section III-B3.
2) **Latent Confounders**: accounting for the existence of latent confounders, without restricting[1] the kinds of identifiable queries that can be estimated.
3) **Counterfactuals**: allowing counterfactual estimation, not only purely interventional queries. This means enabling abduction of factual variables to propagate their abducted noise to the counterfactual graph.
4) **Expressiveness**: providing expressive implementations capable of modelling complex real-world data. Finding the appropriate probability distribution for each and every node can be time-consuming (not scalable) or even unfeasible (real-world data need not fit within any of these families), so a method adopting this approach would not fulfill the requirement.
5) **Scalability**: instead of defining a different network for each node, defining a single network for all variables at the same time. This limits the number of trainable parameters in the model for graphs with a large number of variables, therefore mitigating overfitting.
6) **Generality**: the method can follow the structure of arbitrary causal graphs, and its training and estimation procedures should be immediately applicable. Most methods derive an expression for each graph and query

[1]Compiling two confounded variables into a single variable lets us model them as part of a graph, but prevents us from intervening on only one of them.

they want to estimate, instead of defining general procedures that can adapt to each situation.

With this in mind, we propose **Deep Causal Graphs** (DCGs), a general, modular, estimand-agnostic framework, which fulfills all of the above points, and is therefore ready for practitioners to use. In order to present our contributions and explain the differences from previous and parallel works, we provide Table 1, which lists the items in the desiderata covered by each method. Together with our proposal (DCG) we include Distributional Causal Nodes (DCN) [21], our first contribution to the problem, in which we introduced the most basic implementation of our approach, later discussed in section III-A1. We encourage readers to refer back to this list once the full technique has been presented, as the reasoning behind each item will be clearer.

It is worth noting that only DSCMs provide a method for computing the explicit **likelihood** of a sample (given an invertible-explicit implementation for its nodes). **Latent Confounders** are only covered in CEVAE (the latent space) and NCMs, whereas VACA proposes collapsing variables affected by the same latent confounder as a multidimensional, heterogenous node, which limits the kinds of queries the model can answer. **Counterfactuals** are only discussed in VACA and DSCM; the NCM paper talks about the third rung on Pearl's Ladder of Causation [24], the counterfactual level, but focuses on purely interventional queries $P(Y \mid do(X))$ in the examples given. Regarding **expressiveness**, we say that CEVAE and NCM are not expressive enough in the implementations that they provide, since they require the assumption of a certain probability distribution in modelling each node rather than a more flexible alternative, such as the Normalizing Flows used in our own approach or DSCMs. With respect to **scalability**, every method except for VACA requires defining a separate network for each node, which leads to overfitting on larger graphs. Finally, in relation to **generality**, we say that CausalGAN and CEVAE are not general since the former requires the definition of a discriminator and two *labeller* networks for the GAN node, which it is not clear how to extend when modelling more than one node with GANs. Also, the latter defines a specific architecture for the particular kind of graph the authors work with, with no indication of how that structure would change with other kinds of graphs. As for our initial proposal, DCNs, we did not cover either the latent confounder case or counterfactual estimation, focusing instead on an implementation very similar to the one later found in NCM, which limited expressiveness. Neither did we use our new Graphical Conditioner (see section III-C), which affected scalability.

Furthermore, none of these methods provide **an algorithmic solution for estimating general queries**, leaving the derivation to the reader, which is not trivial in some cases. We devote section III-B to this problem, covering observational/interventional/counterfactual sampling, likelihood and expectation queries, all with or without a conditioning term. We also provide an **open-source library** with all of our implementations ready for practitioners to use and researchers to extend. This library and all the experiments in this work can be found at https://github.com/aparafita/dcg_supp.

In summary, the **contributions** of this work are as follows:

1) The definition of a general, modular, estimand-agnostic framework, along with its training and estimation procedures, encompassing many kinds of Causal Query Estimation problems.
2) The specification of a Normalizing Flow implementation[2] for this framework, which allows us to model complex continuous distributions, while flexible enough to adopt any advances in the topic.
3) The implementation of a complete open-source library containing all techniques described in the paper.

II. BACKGROUND

In this section, we will define Structural Causal Models, the main structure on which we base our framework, the concepts of intervention and counterfactuals, and the problem of identifiability in Causal Queries, which is essential to the validity of our estimations.

A. STRUCTURAL CAUSAL MODELS

We define a **Structural Causal Model** (SCM) as the tuple $\mathcal{M} = (\mathcal{V}, \mathcal{E}, \mathcal{U}, \mathcal{P}, \mathcal{F})$ with:

1) $\mathcal{V} = \{V_k\}_{k=1..K}$ measured variables, from which we have observed samples in our dataset.
2) $\mathcal{E} := \{E_k\}_{k=1..K}$ exogenous noise signals, one for each V_k, which provide stochasticity to the otherwise deterministic functions $f_k \in \mathcal{F}$.
3) $\mathcal{U} \subseteq \{U_{\{k,l\}}\}_{k,l=1..K,\ k \neq l}$ latent confounder variables, for which we do not have any samples, but correlate pairs[3] of measured variables V_k, V_l.
4) $\mathcal{P}(\mathcal{E}, \mathcal{U})$ prior distribution for both sets of latent variables. $\mathcal{E}$ and $\mathcal{U}$ are mutually and internally independent: $\mathcal{P}(\mathcal{E}, \mathcal{U}) = \prod_{E \in \mathcal{E}} \mathcal{P}(E) \cdot \prod_{U \in \mathcal{U}} \mathcal{P}(U)$.
5) $\mathcal{F} := \{f_k \mid V_k := f_k(Pa_k, U_{\{k,.\}}, E_k)\}_{k=1..K}$ functions describing the relationship between each variable $V_k \in \mathcal{V}$ and its corresponding E_k, its confounders $U_{\{k,.\}}$ and its parent set $Pa_k \subset \mathcal{V} \setminus \{V_k\}$.

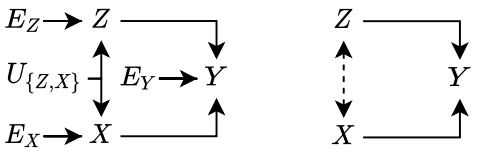
Let us denote $Pa'_k := Pa_k \cup U_{\{k,.\}}$, the set of all parent values (including confounders $U_{\{k,.\}}$) of node V_k. The relationships between the inputs in each f_k and the resulting variable V_k define a directed **graph structure** $\mathcal{G}_\mathcal{M} = (N, E)$ with nodes $N := \mathcal{V} \cup \mathcal{E} \cup \mathcal{U}$ and edges E connecting every input-output in $\mathcal{F}$, $E := \{X \to V_k \mid \forall X \in Pa'_k \cup \{E_k\}\}_k$. In this work, we focus on SCMs $\mathcal{M}$ with graphs $\mathcal{G}_\mathcal{M}$, which

[2] Although this technique coincides with Pawlowski's [22], we proposed it simultaneously [20] on a non-archived paper that precedes this work.

[3] This definition of SCMs limits its structure to latent confounders that are root nodes with exactly two children. Tian *et al.* [34] show that for arbitrary latent confounders (e.g., non-root latent confounders in between measured variables, or root confounders with more than two descendants) we can *project* them onto a new set of confounders that follows our restriction. This projection preserves the set of independences between the measured variables; therefore, it does not preclude the set of queries we can compute from the model, nor their validity.

TABLE 1: Desiderata for an estimand-agnostic Causal Query Estimation framework.

Method	Explicit Likelihood	Latent Confounders	Counterfactuals	Expressiveness	Scalability	Generality
CausalGAN [10]	✗	✗	✗	✓	✗	✗
CEVAE [15]	✗	✓	✗	✗	✗	✗
VACA [28]	✗	✗	✓	✓	✓	✓
DSCM [22]	✓	✗	✓	✓	✗	✓
NCM [39]	✗	✓	✗	✗	✗	✓
DCN [21]	✓	✗	✗	✗	✗	✓
DCG	✓	✓	✓	✓	✓	✓



(a) (b)

FIGURE 2: Example of an SCM. We normally omit variables in $\mathcal{E}$ (they are considered implicit) and latent confounders $\mathcal{U}$ are represented by dashed bi-directional edges between both affected variables. The explicit, unfolded representation (a) is usually summarized by means of the implicit graph (b).

are **Directed Acyclic Graphs** (DAGs). Henceforth, we assume that our DAGs list their measured variables $\mathcal{V}$ following a topological order of the graph ($V_i \in An(V_j) \rightarrow i \leq j$, with $An(X)$ the set of ancestors of X including X).

By way of example, let us define an SCM $\mathcal{M}$ with variables Rain (Z, "Has it rained in the last 24 hours?"), Sprinkler (X, "Did the user activate the sprinkler yesterday?") and Wet (Y, "Is the grass wet?"). There is a confounder between Rain and Sprinkler that has not been measured (and hence is latent), called Weather ($U_{\{Z,X\}}$), which affects both rain and the probability of activating the sprinkler. The resulting graph $\mathcal{G}$ is the one in Fig. 2a. We normally omit exogenous noise signals $\mathcal{E}$ from the graph, since every observable variable implicitly has its own, and latent confounders are represented by a bi-directed dashed arrow between both affected nodes. Fig. 2b shows the corresponding implicit graph.

An SCM describes a **sampling procedure** through its latent priors $\mathcal{P}$ and its functional relationships $\mathcal{F}$. We take a sample $(\varepsilon, u) \sim \mathcal{P}(\mathcal{E},\mathcal{U})$, and then progressively apply each function $f_i \in \mathcal{F}$ to generate a new value for V_i. Since $\mathcal{V}$ follows a topological order of the graph, each function f_i already has its inputs $(pa_i, u_{\{i,.\}}, \varepsilon_i)$, and can therefore deterministically generate a new value v_i for V_i. In other words, $\mathcal{F}$ is a functional from random variables $(\mathcal{E},\mathcal{U})$ to $\mathcal{V}$; the result of this is called the **observational distribution** $\mathcal{P}(\mathcal{V})$.

Due to the SCM's structure, the observational distribution $\mathcal{P}(\mathcal{V})$ fulfills the **Parental Markov Condition** [23] *w.r.t.* graph $\mathcal{G}_\mathcal{M}$, which means that every variable V_k is independent of all its nondescendants conditional on its parents Pa'_k. Consequently, given the variables in $\mathcal{V}$ in topological order,

$$P(\mathcal{V},\mathcal{U}) = \prod_{k=1..K} P(V_k \mid V_{<k},\mathcal{U})\ P(\mathcal{U}) = \prod_{k=1..K} P(V_k \mid Pa'_k) \prod_{U\in\mathcal{U}} P(U), \tag{1}$$

with $V_{<k} := \{V_1, \ldots, V_{k-1}\}$.

B. INTERVENTIONS

The concept of *interventions* entails editing the model $\mathcal{M}$ by changing some of its functions $\mathcal{F}$. The most basic kind of intervention is the *atomic intervention*, which replaces the functions of a set of variables $\mathbf{X} \subset \mathcal{V}$ by some constant value $\mathbf{x}$, denoted by $do(\mathbf{X} = \mathbf{x})$. As a result, for each $X \in \mathbf{X}$, the function f_X becomes $f_X := x$. This creates a new SCM, the **intervened model** $\mathcal{M}_\mathbf{x}$, with an altered set of functions $\mathcal{F}_\mathbf{x}$; note that the corresponding intervened graph $\mathcal{G}_{\mathcal{M}_\mathbf{x}}$ cuts any edges pointing towards nodes in $\mathbf{X}$.

Other kinds of interventions define new functions f_X, which do take inputs from other variables. For example, a movie recommender could be modelled as an intervention on the *exposure* of a movie to a user by defining a new function with information about each movie as its inputs. In this case, the intervened graph replaces the edges leading to *exposure* by new edges connecting to the inputs of the new function. In the following sections, we will operate with atomic interventions only; any of our procedures should be reevaluated for other kinds of interventions, which we will leave for future work.

C. COUNTERFACTUAL PARALLEL WORLDS

A **counterfactual** is the hypothetical result that an intervention may have on an individual for whom we have already observed a different *factual* outcome. For example, we measured a certain blood sugar level on a patient who was not treated and we want to know what the blood sugar would have been had they taken the treatment. This *parallel* world where certain variables are *intervened upon* and consequently result in a different outcome is what we call the *counterfactual world*.

Returning to the previous example in Fig. 2, let us describe a certain sample $v = (z, x, y)$, where we observe that the grass is not wet (y), even though it did rain (z) yesterday, but the sprinkler (x) had not been turned on. Knowing this factual observation gives us insight about the latent variables (maybe it was a particularly hot day after the rain and the

water had evaporated since); we extract that information by computing $P(\mathcal{E},\mathcal{U} \mid v)$, a process called **abduction**. We then study what effect a new **intervention**, turning the Sprinkler on, would have had on the eventual outcome variable, Wet, knowing the posterior state of the latent variables thanks to the *factual* observations. This results in a **prediction** of that *counterfactual* outcome, which lets us answer the query "Would the grass be wet had we turned the sprinkler on, knowing that it is dry now, and that it did rain yesterday but we did not turn the sprinkler on?". The three-step process described above, abduction-intervention-prediction, is the counterfactual process defined by Pearl [23], which lets us consider this kind of hypothetical causal query.

Counterfactuals are essential for **explainability** applications [36]: knowing the effects that certain interventions would have had in contrast with the factual outcome we observe allows us to study the effect of these variables not for the general case, but for particular individuals. "How would my salary change had I been a man?" is an example of a counterfactual query, where the interest is not on actually applying the intervention, but in finding the reasons behind a certain outcome, or even the **fairness** of a decision [11].

Note that in the previous example, given the original SCM $\mathcal{M}$, the counterfactual query consists of estimating an intervention on a slightly modified SCM $\mathcal{M}_{v,do(X=x')} := (\mathcal{V},\mathcal{E},\mathcal{U},\mathcal{P}(\mathcal{E},\mathcal{U} \mid v),\mathcal{F}_{x'})$ with the latent priors conditioned on the observed variables v and the functions $\mathcal{F}$ affected by $do(X = x')$. However, this notation is ambiguous, as the variables in $\mathcal{V}$ are duplicated between the factual and counterfactual model, and it is essential to distinguish between the two in order to derive the proper expressions.

An alternative notation is the so-called **Parallel Worlds Graph** [32]; we extend the original SCM $\mathcal{M}$ with a second one, with every variable in $\mathcal{V}$ replicated (normally denoted by a subscript with the intervention value, e.g., $Y_{x'}$) but with every variable in $\mathcal{E}$ and $\mathcal{U}$ shared between the two. As an example, consider Fig. 3; in this graph, E_Y points towards Y and $Y_{x'}$. Normally, any intervened variable (X) does not appear in the counterfactual world (they are constant variables, given atomic interventions); also, any duplicated variables with the exact same parents should be fused together as one ($Z = Z_{x'}$, since both share $U_{\{Z,X\}}$ as the only parent and f_Z as their functional assignment, therefore having the exact same distribution). For clarity, even though this is not shown in the example graph, we should proceed with it in mind.

When we talk about counterfactuals, we normally condition on some factual outcome $v = (z,x,y)$ and query the variables in the counterfactual world subject to intervention $do(X = x')$. Therefore, to bridge the gap between both worlds, we must compute the posterior for every latent variable in $\mathcal{E}$ and $\mathcal{U}$; specifically, E_Y's prior changes, which affects the counterfactual variable $Y_{x'}$. Note that $Z_{x'}$ is not affected by the intervention, since it is an ancestor of X, which is why we would normally fuse it with Z.

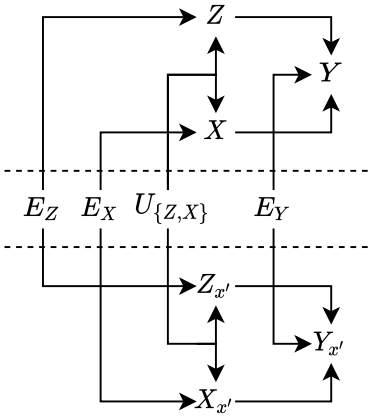


FIGURE 3: Parallel World Graph example. Based on the graph from Fig. 2, we construct two subgraphs, the factual (up) and the counterfactual (down), linked by all latent variables. To better convey the "parallel world" concept, here we do not remove $X_{x'}$ or fuse Z and $Z_{x'}$.

D. IDENTIFIABILITY

Given a finite i.i.d. dataset $\mathcal{D} = (v^{(i)})_{i=1..N}$ for measured variables $\mathcal{V}$, we can safely assume that an **underlying SCM** $\mathcal{M}$ generated our dataset. We may not know the functional relationships $\mathcal{F}$ nor the latent priors $\mathcal{P}$ that define this SCM, but finding the graph $\mathcal{G}_\mathcal{M}$ is feasible, either by experimental testing or by using causal discovery algorithms [33]. This paper does not focus on finding this graph structure, instead assuming it is already known. In the following sections, we assume that our datasets $\mathcal{D}$ come from an underlying SCM $\mathcal{M}$ from which we only know the causal graph $\mathcal{G}_\mathcal{M}$, and that its observational distribution $\mathcal{P}(\mathcal{V}) > 0$ in all its domain.

Consider a query $Q(.)$, which depends on an SCM $\mathcal{M}$ (e.g., $Q(\mathcal{M}) := \mathbb{E}[Y \mid do(X = x)]$). Based on an i.i.d. dataset D generated with an underlying $\mathcal{M}$, we want to find an estimator of $Q(\mathcal{M})$. Our approach consists in **defining a proxy SCM $\mathcal{M}'$ with equivalent graph $\mathcal{G}_{\mathcal{M}'} = \mathcal{G}_\mathcal{M}$ and observational distribution $\mathcal{P}_{\mathcal{M}'}(\mathcal{V}) = \mathcal{P}_\mathcal{M}(\mathcal{V})$**; then, we will answer the query with $\mathcal{M}'$ as if it were the underlying $\mathcal{M}$. Can we use $Q(\mathcal{M}')$ as an estimator of $Q(\mathcal{M})$ if the functions in $\mathcal{F}_{\mathcal{M}'}$ or the priors $\mathcal{P}_{\mathcal{M}'}(\mathcal{E},\mathcal{U})$ are not the same as the ones in $\mathcal{M}$? Yes, but only if the query is *identifiable*.

Let us define the class of models $\mathbb{M}(\mathcal{M})$ consisting of all SCMs $\mathcal{M}'$ such that $\mathcal{G}_{\mathcal{M}'} = \mathcal{G}_\mathcal{M}$ and $\mathcal{P}_{\mathcal{M}'}(\mathcal{V}) = \mathcal{P}_\mathcal{M}(\mathcal{V})$. We say that a query Q is **identifiable** in $\mathbb{M}(\mathcal{M})$ if $\forall \mathcal{M}' \in \mathbb{M}(\mathcal{M}),\ Q(\mathcal{M}') = Q(\mathcal{M})$. This can be **proven by finding an estimand** for Q [23], a formula consisting of only observational terms (involving only the measured variables $\mathcal{V}$ without interventions). If it exists, then both $\mathcal{M}$ and $\mathcal{M}'$ would output the same result for that estimand, since they share the same causal structure (hence the estimand applies for both models) and observational distribution $\mathcal{P}(\mathcal{V})$ (hence each term in the formula returns the same results).

Queries can be proven to be identifiable (or not) **by automatic algorithms**: interventional queries (with or with-

out a conditional) are covered by [31] and [30], and counterfactual queries by [32]. These algorithms either provide an estimand, proving identifiability, or prove it cannot exist, proving non-identifiability. Note that we only need to prove the existence of an estimand; if it exists, the query is identifiable and we can discard the estimand and estimate the query with our proxy SCM directly. See [35] or [25] for implementations of the first two algorithms in R and Python, respectively.

In conclusion, given a proxy SCM $\mathcal{M}'$ with the same causal structure as $\mathcal{G}_{\mathcal{M}}$ and the same distribution $\mathcal{P}_{\mathcal{M}}(\mathcal{V})$[4], if a certain query (interventional or counterfactual) is *identifiable*, we can estimate it with $\mathcal{M}'$ as if we were using the original underlying $\mathcal{M}$. Note that our model need not have the same functions $\mathcal{F}$ nor latent priors $\mathcal{P}(\mathcal{E}), \mathcal{P}(\mathcal{U})$, only that $\mathcal{P}_{\mathcal{M}'}(\mathcal{V}) = \mathcal{P}_{\mathcal{M}}(\mathcal{V})$. More importantly, we do not need the estimand that proved the query's identifiability, as the SCM itself can estimate the query.

III. DEEP CAUSAL GRAPHS

In this section, we describe our framework: Deep Causal Graphs (DCGs). Our aim here is two-fold: 1) to define the architecture of our DCG models $\mathcal{M}$, by means of which we can flexibly model complex real-world data distributions $\mathcal{P}(\mathcal{V})$ while also complying with the required graph structure $\mathcal{G}$; and 2) to define the procedures with which we can estimate these queries. We begin by defining the smallest building block, Deep Causal Units, and continue by discussing the training of DCGs and their corresponding estimation procedures. We finish the section with the Graphical Conditioner, which allows us to train every node with a single network at once, thereby preventing overfitting the model when the number of nodes increases.

A. DEEP CAUSAL UNIT

A DCG is an SCM where every variable/node V_k in $\mathcal{V}$ is modelled by a submodule, called the **Deep Causal Unit** (DCU). Each DCU can be understood as a subnetwork with trainable parameters that models the distribution of its own node conditioned on its parents, $\mathcal{P}(V_k \mid Pa'_k)$, while providing functionality for three distinct operations:

1) *Sample*, generating samples from $\mathcal{P}(V_k \mid Pa'_k)$ by taking a value $\epsilon_k \sim \mathcal{P}(E_k)$ and passing it through the function f_k modelled by the node.
2) *Loglk*, computing the log-likelihood $\log p(v_k \mid pa'_k)$ corresponding to the random variable V_k that results from using f_k as the sampling operation. This operation must be **differentiable** *w.r.t.* the distribution's parameters Θ, as it will employed to compute the training loss for the overall DCG.
3) *Abduct*, sampling from the posterior $\mathcal{P}(E_k \mid V_k, Pa'_k)$. This is required for counterfactual estimation.

[4]Note that we can never expect to achieve a perfect match between our SCM's $\mathcal{P}_{\mathcal{M}'}(\mathcal{V})$ and the real underlying distribution $\mathcal{P}_{\mathcal{M}}(\mathcal{V})$. This miscalibration has an effect on the eventual estimations we perform with the model; an analysis on this topic is left for future work.

Note that this definition of the DCU requires that every node in the graph defines its own subnetwork (the *Conditioner*), which would not scale when the number of variables is too high. Alternatively, we can employ the Graphical Conditioner, discussed in section III-C, which encompasses every node's network into a single network, thereby bypassing the problem. For clarity of explanation, we will proceed as if we defined a specific network for each node; when we explain the Graphical Conditioner, we will see how this is simply an abstraction for a single all-encompassing network.

In the following subsections III.A-D, we will cover four possible implementations of this specification, each with its use cases. As long as they can execute these three operations, they are DCUs and can be integrated within the overall DCG framework.

1) Distributional Causal Nodes

The most basic implementation for DCUs is the **Distributional Causal Node** (DCN), first proposed in [21]. Here, we assume that each random variable $(V_k \mid Pa'_k)$ behaves like a known parametric distribution family (e.g., the Exponential distribution) with parameters Θ. These parameters come as a function of its parents Pa'_k ($\Theta := \Theta(Pa'_k)$), which we model with a feed-forward network and pertinent activation functions for each parameter depending on its domain (e.g., a softplus function for $\sigma > 0$, or a softmax for $(p_k)_{k=1..K}$ so that $\sum_{k=1}^{K} p_k = 1$); this network is the DCN's *Conditioner*.

Since we know the distribution family, we can use the corresponding **log-likelihood** (op. 2) formula; the only requirement is that it is differentiable *w.r.t.* Θ. For **sampling** (op. 1), differentiability is desirable[5] but not necessary; we just need to define a certain prior for E_X independent to Θ and a deterministic function that transforms samples $\varepsilon_X \sim E_X$ into samples $x \sim P(X \mid Pa'_X)$. This can be done by using the reparametrization trick [9] or with inverse transform sampling ($\varepsilon_X \sim \mathcal{U}(0,1), x := F_{\Theta}^{-1}(\varepsilon_X)$, with F_{Θ} the CDF of $X(\Theta) = (X \mid Pa'_X)$). Finally, **abduction** (op. 3) needs to sample values from $P(E_X \mid X, Pa'_X)$. If the sampling operation is invertible, this distribution is constant and we just invert the formula; otherwise, we need a different strategy, discussed in the following two paragraphs. Table 2 contains the specifications for several continuous and discrete distributions.

As an example of a more involved abduction process, consider the Bernoulli/Categorical distribution. For sampling, we need K i.i.d. Gumbel values to use the Gumbel-argmax trick, which allows samples to be generated differentiably *w.r.t.* parameters p; this, however, results in a non-injective sample function. Nonetheless, abduction is still possible: given the observed category k' ($x = k'$) and pa'_X, which gives us $\Theta = (p_k)_k$, we sample $g_{k'} \sim \mathcal{G}(0,1)$ and based on this value, we sample the remaining $K-1$ values from $\mathcal{G}(\log p_k, 1)$

[5]We could define alternative training methods, such as Adversarial Training, if all sampling operations in each DCU were differentiable *w.r.t.* Θ; however, given a differentiable loglk operation, Maximum Likelihood Estimation (our main training method) will always be applicable.

TABLE 2: DCN specification for several continuous and discrete distributions: Normal $\mathcal{N}$, Exponential Exp, Asymmetric Laplace ALD, Beta $\mathcal{B}$, Truncated Distribution $X(\Theta)$ to interval (a, b), Categorical Cat and $Poisson$. These distributions are already implemented in our library; this table is merely to illustrate how they are defined as DCNs.

$Distr(\Theta)$	E_X PRIOR	SAMPLE: $x \sim P(X \mid \Theta(pa'_X))$	LOGLK: $\log p(x \mid \Theta(pa'_X))$	ABDUCT: $\varepsilon \sim P(E_X \mid x, \Theta(pa'_X))$
$\mathcal{N}(\mu, \sigma)$	$\mathcal{N}(0,1)$	$x \leftarrow \sigma \cdot \varepsilon + \mu$	$-\frac{1}{2}(\log 2\pi\sigma^2 + \frac{(x-\mu)^2}{\sigma^2})$	$\varepsilon \leftarrow \frac{x-\mu}{\sigma}$
$Exp(\lambda)$	$\mathcal{U}(0,1)$	$x \leftarrow -\frac{\log \varepsilon}{\lambda}$	$\log \lambda - \lambda x$	$\varepsilon \leftarrow exp\{-\lambda x\}$
$ALD(\mu, \lambda, \kappa)$	$\mathcal{U}(0,1)$	$\varepsilon \leftarrow \varepsilon(\kappa + \kappa^{-1}) - \kappa$ $s \leftarrow sign(\varepsilon)$ $x \leftarrow \mu - \frac{1}{\lambda s \kappa^s} \log(1 - \varepsilon s \kappa^s)$	$s \leftarrow sign(x - \mu)$; $\log \frac{\lambda}{\kappa + \kappa^{-1}} - (x - \mu)\lambda s \kappa^s$	$s \leftarrow sign(x - \mu)$ $\varepsilon \leftarrow (1 - e^{-(x-\mu)\lambda s \kappa^s}) s \kappa^{-s}$ $\varepsilon \leftarrow \frac{\varepsilon + \kappa}{\kappa + \kappa^{-1}}$
$\mathcal{B}(\alpha, \beta)$	$\mathcal{U}(0,1)$	$x \leftarrow PPF(\varepsilon, \alpha, \beta)$	$\log PDF(x, \alpha, \beta)$	$\varepsilon \leftarrow CDF(x, \alpha, \beta)$
$X(\Theta \mid X \in (a, b))$ $\varepsilon_a := CDF(a, \Theta)$ $\varepsilon_b := CDF(b, \Theta)$	$\mathcal{U}(0,1)$	$\varepsilon \leftarrow \varepsilon(\varepsilon_b - \varepsilon_a) + \varepsilon_a$ $x \leftarrow PPF(\varepsilon, \Theta)$	$\log \frac{PDF(x, \Theta)}{\varepsilon_b - \varepsilon_a}$	$\varepsilon \leftarrow CDF(x, \Theta)$ $\varepsilon \leftarrow \frac{\varepsilon - \varepsilon_a}{\varepsilon_b - \varepsilon_a}$
$Cat(p_1, \cdots, p_K)$	$\mathcal{G}(0,1)^K$	$x \leftarrow argmax_k(\log p_k + \epsilon_k)$	$\sum_k x_k \cdot \log p_k$	$k' \leftarrow argmax_k\, x_k$ $\varepsilon_{k'} \sim \mathcal{G}(0,1)$; $\forall k \neq k',\ \varepsilon_k \sim \mathcal{G}(\log p_k, 1)$ $\forall k \neq k',\ \varepsilon_k \leftarrow -\log(e^{-\varepsilon_k} + e^{-\varepsilon_{k'}})$ $\forall k,\ \varepsilon_k \leftarrow \varepsilon_k - \log p_k$
$Poisson(\lambda)$	$\mathcal{U}(0,1)$	$x \leftarrow min\{n \mid \sum_{k=0}^{n+1} \frac{\lambda^k}{k!} > e^\lambda \varepsilon\}$	$x \log \lambda - \lambda - \sum_{k=1}^{x} \log k$	$\varepsilon \sim \mathcal{U}(e^{-\lambda} \sum_{k=0}^{x} \frac{\lambda^k}{k!}, e^{-\lambda} \sum_{k=0}^{x+1} \frac{\lambda^k}{k!})$

truncated by the previous $g_{k'}$. Finally, we transform these g values back to a $\mathcal{G}(0,1)$ so as to decouple them from the parameters $\log p$. See [16] and [17] for more details.

In the case of the Beta distribution, we cannot find a reparametrization formula that allows for abduction. However, **inverse transform sampling** will always cover these two operations and allow the distribution to be applied to a DCN, as long as we can compute its log-likelihood differentiably *w.r.t.* Θ and we have algorithms for its Percentile Point Function (PPF) and Cumulative Distribution Function (CDF), one inverse of the other. With these, we can transform from our X to $\mathcal{U}(0,1)$ and back. This general strategy is applicable to a number of other distributions, and even allows for distributions X truncated to intervals (a, b) (possibly infinite) as long as its CDF is differentiable *w.r.t.* Θ: we compute the CDF of its extremes $\varepsilon_a, \varepsilon_b$ and use them on all three steps of the DCU, as shown in the table.

In summary, DCNs are general, expressive DCU implementations that encompass a wide array of distributions. None of the following DCUs work for discrete distributions, so DCNs are the *de facto* DCU in these cases. However, for the continuous case, the requirement to specify a certain distribution family for each variable does not scale in graphs with many nodes; it can also be too restrictive, as a known family might not fit real world data. To avoid these problems of scalability and expressiveness, we propose an alternative DCU implementation in the following subsection. However, when dealing with simpler distributions or a small number of training samples, DCNs are still a good option.

As a final note, if we wanted to use a linear SCM embedded in the DCG framework, this would be possible with DCNs, by forcing every node's Conditioner network to be a simple Linear layer with appropriate activations. This means that every procedure described in section III-B is also applicable to the linear case. However, we must be sure that such a restrictive architecture is capable of modelling $\mathcal{P}(\mathcal{V})$; otherwise, its estimations would not be reliable.

2) Normalizing Causal Flows

A different strategy for continuous distributions is the use of Conditional Normalizing Flows, which are density estimation methods based on defining an invertible function between two random variables X and E, given conditioning Z, $E = f(X \mid Z)$. For our purposes, $X := V_k$, $E := E_k$ and $Z := Pa'_k$. Then, we can define **Normalizing Causal Flows** (NCF), a flow-based DCU implementation: 1) **sample** consists of obtaining a value $\epsilon_k \sim \mathcal{P}(E_k)$ (with $\mathcal{P}(E_k)$ predetermined by the flow) and then $v_k := f_k^{-1}(\varepsilon_k \mid pa'_k)$; 2) **loglk** comes from the flow's log-likelihood procedure; 3) **abduct** outputs values by applying f directly: $\epsilon_k = f_k(v_k \mid pa'_k)$.

The function f_k is normally defined as the conjunction of a Transformer (the actual function f_k, which depends on some parameters Θ) and a *Conditioner* (a network that takes the conditioning values as input and outputs Θ, used by the Transformer to *transform* X into E). See [18] for an extensive survey on the subject. Note that this description of the DCU does not impose any restrictions on the Transformer operation, and for unidimensional variables (as is the case for most nodes in a causal graph) the Conditioner need not have a particular architecture. As such, we can use any Transformer architecture from the literature and it will work like any

other DCU in the overall DCG framework; this allows us to leverage any advances in the field for our models, which is essential in properly modelling the desired $\mathcal{P}(\mathcal{V})$.

For multi-dimensional variables, the Conditioner might require special restrictions (e.g., an autoregressive structure, coupling layers, etc.). If that is the case, we can isolate the Conditioner for this variable as a separate network, leaving the overall Graphical Conditioner for the rest of $\mathcal{V}$; as a result, we can use highly specialized networks for complex nodes, while leaving the general Conditioner, which is less prone to overfitting, for simpler nodes.

3) Mixture DCU

An interesting finding is that mixtures of *any* DCU implementation are themselves valid DCUs. Let us consider a node X, with parents Pa'_X, and assume we have K DCU implementations $(X^{(k)})_{k=1..K}$ for X (not necessarily homogeneous), each with its own $E^{(k)}$ and likelihood function $f^{(k)}$. Let us define an additional noise signal $(A \mid Pa'_X)$ modelled with a Categorical distribution with probabilities $p := (p^{(k)})_{k=1..K}$ dependent on Pa'_X. We can now define their K-mixture:

- **Sample**. Generate $a \in \{1, \dots, K\} \sim \mathcal{P}(A \mid Pa'_X)$. This can be done with Gumbel sampling, as in the Categorical-DCN implementation. Then, given the corresponding k resulting from that Categorical, we sample from the kth DCU component $(X^{(k)} \mid Pa'_X)$ as usual.
- **Loglk**. We only need the likelihood of a mixture:

$$\log f(x \mid Pa'_X) = \log \sum_{k=1..K} p^{(k)} f^{(k)}(x \mid Pa'_X) = \log \sum_{k=1..K} \exp\left(\log p^{(k)} + \log f^{(k)}(x \mid Pa'_X)\right).$$

  The log-sum-exp trick is used for numerical stability.
- **Abduct**. With a Mixture Node, our exogenous variables are A and $(E^{(k)})_{k=1..K}$, so we need to sample from $P(A, (E^{(k)})_k \mid X, Pa'_X)$, but we do not know which component $X^{(k)}$ generated X. However,

$$P(A, (E^{(k)})_k \mid X, Pa'_X) = P(A \mid X, Pa'_X)\, P(E^{(A)} \mid X, Pa'_X, A) \prod_{k \neq A} P(E^{(k)}).$$

  Given a value for A, we can independently sample from every other $E^{(k)}$, $k \neq A$ and only abduct with the A-th component $P(E^{(A)} \mid X, Pa'_X, A)$. The remaining $P(A \mid X, Pa'_X)$ is solved by conditional sampling: generate N i.i.d. samples from $(A \mid Pa'_X)$ and then use weights $s(\log P(X \mid Pa'_X, a))$, s being the softmax operation. Refer to section III-B3 for more details.

Mixture Nodes can be used to empower more restrictive DCUs (DCNs, in particular, benefit from this). By way of example, we can define Gaussian Mixtures with this technique using the simple Gaussian-DCN implementation. Additionally, it is possible to create mixtures from models trained with different splits in a Cross Validation setup; this helps in datasets with a limited number of training samples, as the validation set in one split can also be employed when training the rest. We will elaborate on this point in section IV-B.

4) Compound DCUs

A natural extension to Mixture DCUs is the Compound DCU. A Compound distribution is a parametrical distribution X dependent on parameters Θ that are themselves random variables with prior $\mathcal{P}(\Theta)$. As a result, X is an **uncountable mixture**, $\mathcal{P}(X) = \int \mathcal{P}(X \mid \Theta)\, \mathcal{P}(\Theta)\, d\Theta$, with every possible value for Θ describing a different component with a certain likelihood of being selected. This DCU generalizes the work in [1], which proposed a similar implementation for uncountable mixtures of Asymmetric Laplace Distributions.

We will assume the components of this mixture to be homogeneous. Each subcomponent is defined by the same set of parameters Θ, which in turn are computed with a network that takes its parents' values as input, $\Theta := \Theta(Pa'_X)$. However, to model the uncountable mixture with a single network, what we do instead is extend the corresponding exogenous signal E_X with a second source of stochasticity: $E_X := (E'_X, E''_X)$, where E'_X follows the usual role in the DCU implementation and E''_X is not used for the sampling operation, but rather employed in computing the parameters $\Theta := \Theta(Pa'_X, E''_X)$. Given any arbitrary prior for E''_X (e.g., $E''_X \sim \mathcal{N}(0, 1)$), using it in the Conditioner network as an additional input adds a source of stochasticity to the parameters' computation, which results in an uncountable mixture of DCUs.

The simplest way to implement this technique is to create an additional latent variable E''_X for every node X that we want to model as a Compound DCU; this latent variable only affects its corresponding X (it is not a confounder). The result is a DCU, since:

- **Sample**. We take a value $\varepsilon''_X \sim \mathcal{P}(E''_X)$, compute the parameters $\Theta_X := \Theta_X(Pa'_X, E''_X)$ and then apply the DCU's sample operation for X as usual.
- **Loglk**. We compute log-likelihoods with X's loglk operation, but marginalized over E''_X:

$$\log f(x \mid Pa'_X) = \log \mathbb{E}_{E''_X \mid Pa'_X}\left[f(x \mid Pa'_X, E''_X)\right] = \log \mathbb{E}_{E''_X}\left[\exp \log f(x \mid \Theta_X(Pa'_X, E''_X))\right].$$

  Note that $E''_X \perp\!\!\!\perp Pa'_X$ (since X acts as a collider). We also use the log-sum-exp trick for numerical stability.
- **Abduct**. As with the Mixture DCU, the downside to this method is that we cannot invert the ε''_X that generated Θ_X and the corresponding x. However:

$$P(E'_X, E''_X \mid X, Pa'_X) = P(E''_X \mid X, Pa'_X)\, P(E'_X \mid X, Pa'_X, E''_X).$$

  The first term is covered by conditional sampling again, generating N i.i.d. samples from $\mathcal{P}(E''_X)$ (note that $E''_X \perp\!\!\!\perp Pa'_X$) using weights $s(\log P(X \mid Pa'_X, \varepsilon''_X))$

as before. The second term comes directly from the internal DCU abduction, as $\Theta_X = \Theta_X(Pa'_X, \varepsilon''_X)$.

This technique allows us to reach the most general form of the parametric approach (DCNs). With just one component, we can implement unmeasurable mixtures of components by moving the stochasticity of the mixture to this additional noise signal, introduced as a new input to the parameter's network. In the example in section V, we will see the potential of Compound DCNs, especially for small datasets. However, note that this technique can be applied to any other kind of DCU, as it makes no assumptions about its internal structure.

B. DCG PROCEDURES

In this subsection, we will detail all DCG procedures needed for model training and query estimation. In the following equations, when referring to a subset of variables $\mathbf{V} \subset \mathcal{V}$ or $\mathbf{E} \subseteq \mathcal{E}$, let $\mathbf{V}^c := \mathcal{V} \setminus \mathbf{V}$, $\mathbf{E}^c := \mathcal{E} \setminus \mathbf{E}$. Let us also denote $\mathcal{E}_{\mathbf{V}} := \{E_V \in \mathcal{E} \mid V \in \mathbf{V}\}$ (then $\mathcal{E}^c_{\mathbf{V}} := \mathcal{E} \setminus \mathcal{E}_{\mathbf{V}}$). We will operate with the Parallel Worlds Graph model [32] described in section II-C, where we replicate all variables in $\mathcal{V}$ to the new counterfactual world $\mathcal{V}_{\mathbf{x}}$ subject to intervention $do(\mathbf{X} = \mathbf{x})$, with only $\mathcal{E}$ and $\mathcal{U}$ being shared. This allows us to distinguish between expressions like $P(Y_{\mathbf{x}} \mid Z)$ and $P(Y_{\mathbf{x}} \mid Z_{\mathbf{x}})$ (pre- and post-intervention conditionals).

1) Observational and Purely-Interventional Queries

First and foremost, we define the **training objective**. DCGs can be trained through Maximum Likelihood Estimation: if there are no latent confounders ($\mathcal{U} = \emptyset$), then $\log P(\mathcal{V}) = \sum_{V \in \mathcal{V}} \log P(V \mid Pa'_V)$; otherwise, we can marginalize over $\mathcal{U}$: $\log P(\mathcal{V}) = \log \mathbb{E}_{\mathcal{U}}\left[P(\mathcal{V} \mid U)\right] = \log \mathbb{E}_{\mathcal{U}}\left[\exp \sum_{V \in \mathcal{V}} \log P(V \mid Pa'_V)\right]$. This expectation can be approximated by sampling N i.i.d. values from our SCM's prior $\mathcal{P}(\mathcal{U})$. The use of logarithms helps with numerical stability, and we also use the log-sum-exp trick for the latter case. Each of the terms $\log P(V \mid Pa'_V)$ can be estimated with each DCU's *loglk* operation, which is required to be differentiable *w.r.t.* the network's parameters. This allows us to optimize the model $\mathcal{M}$, with a view to maximizing the average log-likelihood of an i.i.d. dataset D taken from the underlying distribution $\mathcal{P}(\mathcal{V})$ that we wish to model.

Secondly, we develop the **sampling routine** to generate samples $\mathbf{v} \sim \mathcal{P}(\mathcal{V})$. To this end, we sample values $\boldsymbol{\varepsilon} \sim \mathcal{P}(\mathcal{E})$, $\mathbf{u} \sim \mathcal{P}(\mathcal{U})$ from their respective priors, and, for each node $V \in \mathcal{V}$, following a topological order of the graph, we use its *sample* operation, passing its parents' values (which may include subsets from $\mathbf{u}$) and its exogenous noise signal $\boldsymbol{\varepsilon}_V$. This generates values $\mathbf{v}$, which follow the DCG's $\mathcal{P}(\mathcal{V})$. Additionally, to sample from $\mathcal{P}(\mathcal{V}_{\mathbf{x}})$ ($\mathcal{V}$ in $\mathcal{M}_{\mathbf{x}}$, subject to $do(\mathbf{X} = \mathbf{x})$), we employ the previous procedure on the intervened model $\mathcal{M}_{\mathbf{x}}$, replacing each f_X by $X := x$ for all $X \in \mathbf{X}$.

Next, we study how to **estimate log-likelihoods** of subsets $\mathbf{V} \subset \mathcal{V}$. Note that $\log P(\mathbf{V}) = \log \mathbb{E}_{\mathcal{E}^c_{\mathbf{V}}, \mathcal{U}}\left[P(\mathbf{V} \mid \mathcal{E}^c_{\mathbf{V}}, \mathcal{U})\right] = \log \mathbb{E}_{\mathcal{E}^c_{\mathbf{V}}, \mathcal{U}}\left[\exp \sum_{V \in \mathbf{V}} \log P(V \mid Pa'_V)\right]$. Each term comes from the DCU's *loglk* operation and its parent values result from applying the previous sampling procedure to fill any variables in $\mathbf{V}^c$. We can also compute conditional queries, simply by realizing that $P(\mathbf{V} \mid \mathbf{Z}) = \frac{P(\mathbf{V}, \mathbf{Z})}{P(\mathbf{Z})}$, both of these terms are computable with the previous procedure. Then, in the presence of interventions $do(\mathbf{X} = \mathbf{x})$, we can simply consider the intervened model $M_{\mathbf{x}}$ to answer the aforementioned kinds of queries, either $P(\mathbf{V}_{\mathbf{x}})$ or $P(\mathbf{V}_{\mathbf{x}} \mid \mathbf{Z}_{\mathbf{x}})$.

Finally, let us consider **expectation queries** $\mathbb{E}_{\mathbf{V}}\left[f(\mathbf{V})\right]$ for arbitrary functions f. These can be estimated using Monte Carlo by taking N i.i.d. samples from $\mathbf{V}$ with the methods detailed above and then averaging the resulting samples $(f(\mathbf{v}^{(i)}))_{i=1..N}$ to estimate the expectation. For conditional queries, in the form $\mathbb{E}_{\mathbf{V}|\mathbf{Z}}\left[f(\mathbf{V})\right]$, we can use importance sampling to solve this, as in the previous case. Note that $\mathbb{E}_{\mathbf{V}|\mathbf{Z}}\left[f(\mathbf{V})\right] = \mathbb{E}_{\mathbf{V}}\left[f(\mathbf{V}) \cdot \frac{P(\mathbf{V}|\mathbf{Z})}{P(\mathbf{V})}\right] = \mathbb{E}_{\mathbf{V}}\left[f(\mathbf{V}) \cdot \frac{P(\mathbf{Z}|\mathbf{V})}{P(\mathbf{Z})}\right]$, so we can sample N i.i.d. $(\mathbf{v}^{(i)})_{i=1..N}$ values as before and perform a weighted average[6] of terms $(f(\mathbf{v}^{(i)}))_{i=1..N}$ with corresponding weights $w^{(i)} := s(\log P(\mathbf{Z} \mid \mathbf{v}))^{(i)}$, with s being the softmax operation. This last step will henceforth be referred to as the **softmax trick**. Lastly, any of these queries subject to an intervention $do(\mathbf{X} = \mathbf{x})$ (if conditional, the conditioning Z is post-interventional, $Z_{\mathbf{x}}$) are treated as previously, considering the intervened model $\mathcal{M}_{\mathbf{x}}$.

2) Counterfactual Queries

In this section, we will cover counterfactual expectations. Although we employ the 3-step process (abduction, intervention, prediction) described by Pearl in [23], we derive the formula to describe where abduction should take place and how DCU operations help perform the desired estimation.

Let us consider the query $\mathbb{E}_{\mathbf{V}_{\mathbf{x}}|\mathbf{Z}}\left[f(\mathbf{V}_{\mathbf{x}})\right]$. Note that set $\mathbf{Z} \subset \mathcal{V}$ might not contain some variables in $\mathcal{V}$ ($\mathbf{Z}^c \neq \emptyset$), and the abduction operation for each node requires a value for every parent of the node and the node itself. Hence, we need to marginalize over $\mathcal{E}^c_{\mathbf{Z}}$ and $\mathcal{U}$ conditioned on $\mathbf{Z}$ (with these two variables and $\mathbf{Z}$ we can obtain values for the remaining $\mathbf{Z}^c$ deterministically); we use the softmax trick to sample unconditionally. On the other hand, to compute the expectation over $\mathbf{V}_{\mathbf{x}}$, we only need values for the remaining $\mathcal{E}$, so we will sample from $\mathcal{E}_{\mathbf{Z} \setminus \mathbf{X}}$ conditioned on all the previous information (abduction); this generates values for $\mathcal{V}_{\mathbf{x}} \setminus \mathbf{X}_{\mathbf{x}}$ (every remaining value in $\mathcal{M}_{\mathbf{x}}$) to finally answer our query:

[6] Note that we approximate this expectation using Monte Carlo by taking N i.i.d. samples $(v^{(i)})_{i=1..N}$ from $\mathbf{V}$ so that $\mathbb{E}_{\mathbf{V}}\left[f(\mathbf{V}) \cdot \frac{P(\mathbf{Z}|\mathbf{V})}{P(\mathbf{Z})}\right] \approx \frac{1}{N}\sum_{i=1..N} f(\mathbf{v}^{(i)}) \cdot \frac{P(\mathbf{Z}|\mathbf{v}^{(i)})}{P(\mathbf{Z})}$. Now, since $P(\mathbf{Z}) = \mathbb{E}_{\mathbf{V}}\left[P(\mathbf{Z} \mid \mathbf{V})\right] \approx \frac{1}{N}\sum_{i=1..N} P(\mathbf{Z} \mid \mathbf{v}^{(i)})$, we can use those same samples $(\mathbf{v}^{(i)})_{i=1..N}$ for the numerator and denominator, which results, by adding log exp before each term, in N times the softmax s of the set $(\log P(\mathbf{Z} \mid \mathbf{v}^{(i)}))_{i=1..N}$ and the N term is cancelled by the first expectation approximation, resulting in the weighted average described above. We can apply this technique in every instance of importance sampling described throughout this work.

$$\mathbb{E}_{\mathbf{V}_\mathbf{x}|\mathbf{Z}}\left[f(\mathbf{V}_\mathbf{x})\right] = \mathbb{E}_{\mathcal{E}^c_\mathbf{Z},\mathcal{U}|\mathbf{Z}}\left[\mathbb{E}_{\mathbf{V}_\mathbf{x}|\mathbf{Z},\mathcal{E}^c_\mathbf{Z},\mathcal{U}}\left[f(\mathbf{V}_\mathbf{x})\right]\right] = \mathbb{E}_{\mathcal{E}^c_\mathbf{Z},\mathcal{U}}\left[\mathbb{E}_{\mathbf{V}_\mathbf{x}|\mathbf{Z},\mathcal{E}^c_\mathbf{Z},\mathcal{U}}\left[f(\mathbf{V}_\mathbf{x})\right]\frac{P(\mathbf{Z}\mid\mathcal{E}^c_\mathbf{Z},\mathcal{U})}{P(\mathbf{Z})}\right] \approx \sum_{i=1}^{N}\mathbb{E}_{\mathcal{E}_{\mathbf{Z}\setminus\mathbf{X}}|\mathbf{Z},\varepsilon^{c\,(i)}_\mathbf{Z},\mathbf{u}^{(i)}}\left[f(V_\mathbf{x}(\varepsilon^{c\,(i)}_\mathbf{Z},\mathbf{u}^{(i)},\mathcal{E}^{(i)}_{\mathbf{Z}\setminus\mathbf{X}}))\right] s(\log P(\mathbf{Z}\mid\varepsilon^c_\mathbf{Z},\mathbf{u}))^{(i)}. \tag{2}$$

We use importance sampling and the softmax trick to simplify the first marginalization over $\mathcal{E}^c_\mathbf{Z},\mathcal{U}\mid\mathbf{Z}$. The second expectation can be approximated by abducting M i.i.d. samples from every node in $\mathbf{Z}\setminus\mathbf{X}$; then we can use these samples to generate values for $\mathcal{V}_\mathbf{x}\setminus\mathbf{X}_\mathbf{x}$. Note that this procedure is simplified when $\mathbf{Z}=\mathcal{V}$ and $\mathcal{U}=\emptyset$, meaning there are no "missing" variables. In those cases, we only perform abduction and a simple average without weights.

Further queries could be answered by means of the three DCU operations, using similar derivation. Evidently, each estimation procedure results in different estimators with more or less variance, but the fact that we can train a single model for an arbitrary graph and employ it for any of these (identifiable) queries using a general estimator is, in our view, more powerful for the end-user than the variety of ad-hoc models currently present in the literature. Moreover, our library already provides utilities for these procedures, so that practitioners can apply them to their problems.

3) Conditional Sampling

The remaining operations concern the procedure by which we can sample from observational, interventional and counterfactual distributions when there are some conditioning values. Let us consider an observational distribution $P(\mathbf{V}\mid\mathbf{z})$, from which we want to generate N i.i.d. samples $(\mathbf{v}^{(i)})_{i=1..N}$. Since $P(\mathbf{V}\mid\mathbf{z}) = P(\mathbf{V})\frac{P(\mathbf{V}|\mathbf{z})}{P(\mathbf{V})}$, we can:

1) Generate M samples $(\mathbf{v}^{(i,j)})_{j=1..M} \sim P(\mathbf{V})$ from unconditioned $\mathbf{V}$.
2) Choose one of these M samples by weighted sampling, with *unnormalized* weights $\widetilde{w}^{(i,j)} := \frac{P(\mathbf{v}^{(i,j)}|\mathbf{z})}{P(\mathbf{v}^{(i,j)})}$.
3) Repeat N times, to generate each $v^{(i)}$.

The corresponding normalized weights $w^{(i,j)}$ result from the softmax operation: $w^{(i,j)} := s(\log\widetilde{w}^{(i,\cdot)})^{(i,j)} = \frac{\exp\log\widetilde{w}^{(i,j)}}{\sum_{j=1..M}\exp\log\widetilde{w}^{(i,j)}} = \frac{\widetilde{w}^{(i,j)}}{\sum_{j=1..M}\widetilde{w}^{(i,j)}}$. Note that $\log\widetilde{w}^{(i,j)} = \log P(\mathbf{v}^{(i,j)}\mid\mathbf{z}) - \log P(\mathbf{v}^{(i,j)})$, and both terms can be obtained through the above procedures.

One downside of this technique is that it requires M unconditional samples to generate each conditional sample. In order to mitigate this problem, we can generate M samples once, and then take N subsamples with replacement. Both alternatives are valid procedures, provided M is big enough. Another potential point of failure is if the conditioning term $\mathbf{Z}=\mathbf{z}$ only happens in highly unlikely regions of $\mathbf{V}$, which would make it harder to find a $\mathbf{v}$ that agrees with $P(\mathbf{V}\mid\mathbf{z})$. In these cases, a bigger value is required for M.

This solves the **observational case** $P(\mathbf{V}\mid\mathbf{z})$. **Interventional conditioned distributions** $P(\mathbf{V}_\mathbf{x}\mid\mathbf{z}_\mathbf{x})$ (subject to intervention $do(\mathbf{X}=\mathbf{x})$) are handled in the same way, but in the intervened SCM. Finally, for **counterfactual distributions** $P(\mathbf{V}_\mathbf{x}\mid\mathbf{z})$, note that we can focus on sampling from every latent variable (conditioned on $\mathbf{z}$), and then follow the deterministic functions in $\mathcal{F}$. Let us split $\mathcal{E}=\mathcal{E}_\mathbf{Z}\cup\mathcal{E}^c_\mathbf{Z}$. Then:

$$P(\mathcal{U},\mathcal{E}_\mathbf{Z},\mathcal{E}^c_\mathbf{Z}\mid\mathbf{z}) = P(\mathcal{U},\mathcal{E}^c_\mathbf{Z}\mid\mathbf{z})P(\mathcal{E}_\mathbf{Z}\mid\mathbf{z},\mathcal{U},\mathcal{E}^c_\mathbf{Z}) = P(\mathcal{U},\mathcal{E}^c_\mathbf{Z})\frac{P(\mathcal{U},\mathcal{E}^c_\mathbf{Z}\mid\mathbf{z})}{P(\mathcal{U},\mathcal{E}^c_\mathbf{Z})}\prod_{k:\,V_k\in\mathbf{Z}}P(E_k\mid v_k,Pa'_k(\mathcal{U},E_{<k})). \tag{3}$$

As before, the first term in the second line is split in two to bypass the conditional with weighted sampling. The second term can be decomposed following the topological order of the graph, focusing on each E_k individually, conditional on its v_k value (given by $\mathbf{z}$) and its parents' Pa'_k, which can be computed deterministically by $\mathcal{U}$ and every prior $E_{<k} := \{E_1,\dots,E_{k-1}\}$. Each of these terms can be computed through the DCU's *abduct* operation.

In summary, for each $i=1..N$, we sample M unconditional terms $((\mathbf{u}^{(i,j)},\varepsilon^{c\,(i)}_\mathbf{Z}))_{j=1..M}$, abduct each E_k sequentially and compute weights $w^{(i,j)} := s(\log\widetilde{w}^{(i,\cdot)})^{(i,j)}$ as before. We then perform weighted sampling with weights w to generate sample i, and repeat N times to obtain N samples $(\mathbf{u}^{(i)},\varepsilon^{(i)})_{i=1..N}$. With this, all that remains is to use the deterministic functions in $\mathcal{F}_\mathbf{x}$ (subject to intervention $do(\mathbf{X}=\mathbf{x})$) and obtain the final samples $(v^{(i)}_\mathbf{x})_{i=1..N}$.

C. GRAPHICAL CONDITIONER

We finish this section with the technique that allows **a single network to be trained for the whole DCG graph**, encompassing all DCU nodes. This is based on [37], which proposes a Conditioner for Normalizing Flows, which respects any independencies described by a DAG. Our Conditioner has an additional requirement, as it models every node's parameters Θ, which are heterogeneous between nodes.

Let us consider a DCG $\mathcal{M}$ with variables $\mathcal{V}$; let us denote $\mathcal{V}' := \mathcal{V}\cup\mathcal{U}$, $D := |\mathcal{V}|$, $D' := |\mathcal{V}'| = |\mathcal{V}| + |\mathcal{U}|$, with $\mathcal{V} = (V_1,\dots,V_D)$ in a topological order and $\mathcal{U} = \{U_{D+1},U_{D'}\}$ in an arbitrary order. Each node $V\in\mathcal{V}$ depends on parameters Θ_V; let us define $\Theta := (\Theta_{V_1},\dots,\Theta_{V_D})$, the concatenation of all Θ_V and $K := |\Theta|$. Since $P(V_1,\dots,V_D) = \mathbb{E}_\mathcal{U}\left[\prod_{V\in\mathcal{V}}P(V\mid Pa'_V)\right]$ and each term depends on $\Theta_V = \Theta_V(Pa'_V)$, we define a *masking matrix* A with shape $D'\times K$, where each term $a_{d,k}$ is an indicator for whether the d-th variable (either from $\mathcal{V}$ or $\mathcal{U}$) is an input for Θ_k. Fig. 4 shows the masking matrix for the graph in Fig. 2. Since every variable is binary, we model them with Bernoulli-DCNs, each with a single p, hence, $\Theta = (p_Z,p_X,p_Y)$.

We can define an arbitrary neural network f with D' inputs and K outputs that, given values for $\mathcal{V}$ and $\mathcal{U}$, returns values for parameters Θ. If we used it directly, $\Theta = f(\mathbf{v},\mathbf{u})$, we would not be respecting the independencies defined by our graph $\mathcal{G}_\mathcal{M}$. However, when computing parameter Θ_k, we can multiply the concatenated vector $(\mathbf{v},\mathbf{u})$ by $A_{\cdot,k}$, so that any variable not a parent of V_k will be masked.

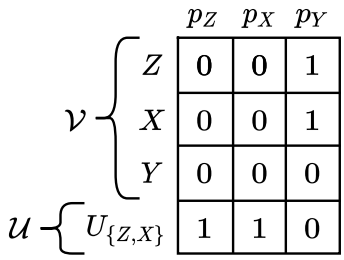


FIGURE 4: Masking matrix for the graph in Fig. 2.

Note that all procedures described in the previous section can be performed using this masking operation: likelihoods (and the training objective) can be computed directly, by estimating all parameters Θ and applying each DCU's *loglk* operation; sampling requires using each node's *sample* operation sequentially in topological order, passing the required parents' values that are computed as we proceed.

In practice, we can define **any network f with arbitrary architecture** to compute our parameters Θ. Not only that, but the use of a **single network**, instead of individual networks for each node, **reduces model complexity, overfitting risk, memory requirements and training times** significantly.

IV. EXPERIMENTS

We now showcase our DCG's capabilities for modelling real-world distributions and test it against a well-established potential outcomes benchmark to evaluate its estimations on interventional and counterfactual queries. Code for all experiments can be found along with the DCG library.

A. MODELLING EXAMPLE

Let us consider the Power dataset from the UCI Machine Learning Repository [2]. We follow the data preprocessing from [3], resulting in six continuous variables. We train a DCG as a complete graph ($V_i \rightarrow V_j, \forall i < j$), which is sufficient to model the *observational* distribution $\mathcal{P}(\mathcal{V})$, and estimate the average log-likelihood of the test split to compare its modelling capabilities against other density estimation methods.

We follow the same experimental setup, flow-transformer architecture and training routine as [3], except that our flows are individually defined for each variable (thus preventing any permutation). Every node in the graph is an NCF, defined as a uni-dimensional Transformer with a shared Graphical Conditioner (see section III-C). The Transformer consists of an initial Batch Normalization Layer followed by a conditional Affine Layer (dependent on the node's parents). Then, ten Rational-Quadratic Spline Flows [3], each divided into eight blocks in the interval $(-3, 3)$, coupled with another Batch Normalization and a conditional Affine Layer. The flow's prior (its exogenous signal prior) is a univariate Normal Distribution. Regarding the (single) Conditioner network architecture, we employ a Residual Network consisting of the following five blocks: a Dropout layer, a Linear Layer and a Rectified Linear Unit, with residual connections between each block. For training, we employ the AdamW algorithm [14], with 10^{-3} learning rate annealed with a cosine schedule dropping down to 0 every 50 epochs. Training continues until 100 epochs have passed without improvements on the validation metric.

We compare our average test log-likelihood ± 1.96 standard deviations (0.54 ± 0.01) against some other methods: the Masked Autoregressive Flow [19] (0.45 ± 0.01) with ResMADEs and invertible linear layers instead of permutations; the Graphical Normalizing Flow [37] (0.62 ± 0.04), the method from which our Graphical Conditioner is derived; and the Rational Quadratic Neural Spline Flow [3] (0.66 ± 0.01), which proposed the flow-transformer in our DCGs. Our model is essentially the same as the two latter ones, except for the fact that we cannot use permutation layers, given that we are restricted by the graph structure we have imposed. Since we do not know the underlying graph, we can only employ a complete graph, with an arbitrary ordering that might not be the most appropriate for the data. We attribute the discrepancy between our and their results to these facts.

Nonetheless, the resulting metric is **competitive with the state of the art**. Moreover, the high modularity of our DCU implementations and the DCG shared Conditioner allows for alternative models that could result in even better performance. Fig. 5 shows the marginal densities of each variable estimated with our DCGs, compared with the histograms from the real data. Note that, **despite each variable being fundamentally different, the same architecture can model all of them indistinctly**.

B. ESTIMATION BENCHMARK

We study here a Causal Query Estimation task, for $Q := P(Y_t \mid Z_t)$ on a graph $(X \rightarrow T, Y;\ T \rightarrow Y)$. We employ two semi-synthetic datasets: IHDP [6], with a continuous outcome, and Jobs [12], with a discrete outcome. We follow the experimental setup and results from [29], and focus on four metrics: e_{ATE}, absolute error in the Average Treatment Effect (ATE); e_{PEHE}, RSME in the Individual Treatment Effect (ITE); e_{ATT}, absolute error in the ATE for the Treated; and R_{pol}, policy risk (also related to ITE). We perform our estimations on 1,000 and ten replications of the experiment respectively, which allows for confidence intervals to be obtained for each metric (± 1.96 standard deviations). We compare against simple and bi-headed Linear Regression (LR_1, LR_2), Causal Effect Variational Autoencoder (CEVAE, the only method in the Desiderata that offered results for this benchmark) [15], Balancing Neural Networks (BNN) [8], Treatment-Agnostic Representation Network (TAR) [29], Counterfactual Regression (CFR) [29], Adaptively similarity-preserved representation learning for

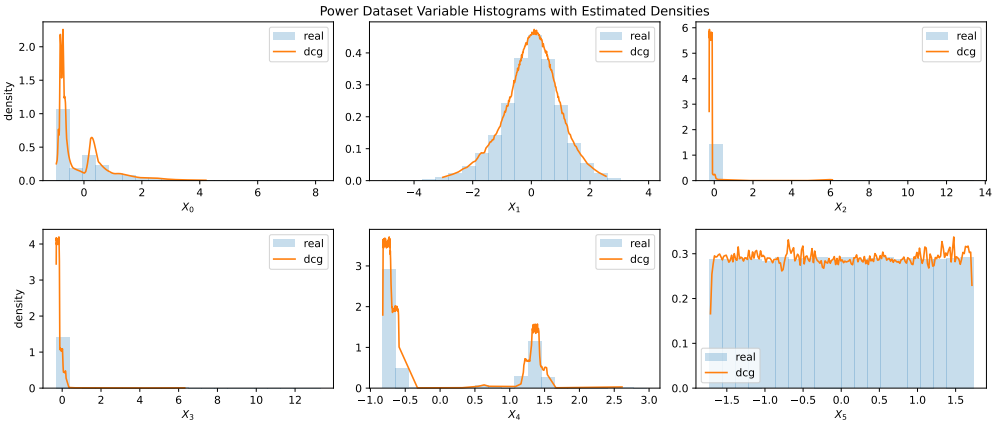


FIGURE 5: Power dataset [2] variable histograms (blue) with DCG-estimated marginal densities (orange).

Causal Effect estimation (ACE)[7] [40], Subspace Learning Based Counterfactual Inference (SCI) [41] and Causal Optimal Transport (CausalOT)[8] [13]. Except for CEVAE and our own technique, all methods are estimand-based and therefore more specialized to estimation of these particular queries.

1) Experiment Setup

For this particular benchmark, we compare against methods that learn $\mathbb{E}[Y_t \mid X_t]$, with T binary and X confounder covariates (25 dimensions for IHDP, 17 for Jobs). Normally, we would model the whole graph node by node, but here we are only interested in the distribution $(Y \mid X, T)$ (learning node Y), as the estimation procedure for each metric does not require modelling either T or X (the edge is $X \rightarrow T$, therefore the intervening T cannot affect X). What we can do instead is define what we call an *Input DCU*, a placeholder node in our DCG that accepts i.i.d. values from the dataset. We can employ the training values for these variables as "samples" when performing the usual estimations, so that we avoid modelling those nodes, as we do not need to perform either *loglk* or *abduct* with them.

We employ this simplification for two reasons. Firstly, the IHDP experiment consists of 1,000 replications, and a model must be trained for each of them. Not only that, but we employ a 5-Fold Cross-Validation (CV) strategy (detailed below), requiring 5,000 models. Were we to train the whole 27-variable graph, training times for the whole benchmark would be prohibitively long. Additionally, modelling errors in the variables of X would propagate to the outcomes in Y. This would be unfair on our model, which is designed to be graph- and query-agnostic when comparing with highly specialized models designed for one singular graph and query. We therefore decided on this simplified strategy for the benchmark. This means that we cannot evaluate the effect of error propagation on our estimations, and such an experiment remains for future work.

Regarding the CV strategy, since the number of samples in the IHDP dataset is quite limited (672 training samples), and our flows are highly flexible models, we train five different models instead - one for each CV split of the original training and validation subsets - and join them all together in a single model with a Mixture Node, discussed in section III-A3. We fix their weights as a constant (independent of Pa'_Y), the softmax of their average validation log-likelihood. This mixture allows us to use the available data more effectively, as the data normally employed for validation can also be used for training in every other submodel. In the following section, we refer to this mixture as Mixture-DCG, whereas Single-DCG represents the single best model in validation from five splits.

With respect to the DCG architecture, each dataset requires a different type of DCU model, as the IHDP outcome is continuous and the Jobs outcome is binary. For the former, we employ an NCF with Rational Quadratic Flows as before, whereas for the latter, we only require a Bernoulli DCN. Both methods use the same Conditioner architecture, consisting of a bi-headed Network, following the example of [29]. Finally, we use the same training procedure as before, with slight hyperparameter adjustments depending on the dataset.

Note that no method in the benchmark uses the factual outcome of Y (Y_f) in their estimation, only the covariates X and the treatment T. However, our DCGs can employ this information (with counterfactual estimation) if available. We compute the resulting metrics without (DCG) and with (DCG*) Y_f to evaluate how our method responds when we have post-facto information about the treatment outcome (e.g., "What would the recovery rate have been if we had administered the treatment?"). Table 3 presents all of the results; we highlight every method that improves on the rest (except DCG*), and DCG* if it is the best among them.

2) Discussion

DCGs achieve the best performance on e_{ATE} for the IHDP

[7] ACE provide results for e_{PEHE} on IHDP, but not for $\sqrt{e_{PEHE}}$, which makes it impossible to compute when averaging the errors for all samples.

[8] CausalOT provides results for both metrics on IHDP, but they do not specify if these metrics come from the train or test split.

TABLE 3: Metrics on the IHDP-Jobs potential outcomes benchmark. Lower is better.

	IHDP				JOBS			
	e_{ATE}		$\sqrt{e_{PEHE}}$		e_{ATT}		R_{pol}	
	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST	TRAIN	TEST
LR_1	$.73 \pm .04$	$.94 \pm .06$	$5.8 \pm .3$	$5.8 \pm .3$	$\mathbf{.01 \pm .00}$	$.08 \pm .04$	$.22 \pm .00$	$.23 \pm .02$
LR_2	$.14 \pm .01$	$.31 \pm .02$	$2.4 \pm .1$	$2.5 \pm .1$	$.01 \pm .01$	$.08 \pm .03$	$.21 \pm .00$	$.24 \pm .01$
CEVAE [15]	$.34 \pm .01$	$.46 \pm .02$	$2.7 \pm .1$	$2.6 \pm .1$	$.02 \pm .01$	$\mathbf{.03 \pm .01}$	$\mathbf{.15 \pm .00}$	$.26 \pm .00$
BNN [8]	$.37 \pm .03$	$.42 \pm .03$	$2.2 \pm .1$	$2.1 \pm .1$	$.04 \pm .01$	$.09 \pm .04$	$.20 \pm .01$	$.24 \pm .02$
TAR [29]	$.26 \pm .01$	$.28 \pm .01$	$.88 \pm .02$	$.95 \pm .02$	$.05 \pm .02$	$.11 \pm .04$	$.17 \pm .01$	$\mathbf{.21 \pm .01}$
CFR [29]	$.25 \pm .01$	$.27 \pm .01$	$\mathbf{.71 \pm .02}$	$\mathbf{.76 \pm .02}$	$.04 \pm .01$	$.09 \pm .03$	$.17 \pm .01$	$\mathbf{.21 \pm .01}$
ACE [40]	-	-	-*	-*	-	-	$.22 \pm .01$	$.22 \pm .01$
SCI [41]	-	-	-	-	-	-	$.21 \pm .01$	$.23 \pm .01$
CAUSALOT [13]	-*	-*	-*	-*	-	-	$.20 \pm .01$	$.21 \pm .03$
SINGLE-DCG	$.19 \pm .02$	$.22 \pm .02$	$1.0 \pm .08$	$1.0 \pm .08$	$.08 \pm .02$	$.08 \pm .02$	$.24 \pm .01$	$.24 \pm .03$
MIXTURE-DCG	$\mathbf{.12 \pm .01}$	$\mathbf{.17 \pm .01}$	$.93 \pm .07$	$.96 \pm .08$	$.07 \pm .01$	$.07 \pm .02$	$.23 \pm .01$	$.25 \pm .05$
SINGLE-DCG*	$.16 \pm .01$	$.19 \pm .01$	$.94 \pm .07$	$.96 \pm .09$	$.09 \pm .02$	$.07 \pm .02$	$.19 \pm .01$	$.19 \pm .04$
MIXTURE-DCG*	$\mathbf{.12 \pm .01}$	$\mathbf{.15 \pm .01}$	$.85 \pm .06$	$.87 \pm .08$	$.08 \pm .02$	$.07 \pm .02$	$\mathbf{.10 \pm .02}$	$\mathbf{.10 \pm .03}$

dataset, and second place on e_{ATT} test for the Jobs dataset, improved on only by CEVAE, which is the best model for this query. Note, however, that the train split is better modelled by far simpler methods (LR_1), which suggests that overfitting might be damaging the performance of the other models. On the other hand, CFR [29] is a clear improvement on ITE metrics (e_{PEHE} and R_{pol}) except for R_{pol} on the training set, whereas DCGs achieve third place for the IHDP dataset, close to TAR. Note that the DCGs we implement for this experiment are essentially equivalent to TAR [29] in all regards (the exact same Conditioner architecture) except that **instead of estimating the expected treatment outcome, they model the actual distribution** through Normalizing Flows, from which we later estimate the expectation. We believe that the added complexity in modelling the distribution (an additional functionality missing from the other methods) might account for this slight drop in performance. CFR, on the other hand, uses balancing regularization between both treatment distributions to improve its results, something that we omitted for this experiment in benefit of simple, general models for arbitrary query estimation. Regarding Single-DCG and Mixture-DCG, the mixture accomplishes better results overall, demonstrating the applicability of this technique on small datasets.

In conclusion, DCGs achieve **competitive results against state-of-the-art models** in potential outcome estimation, **even though they do not use ad-hoc estimands** of the query at hand. Rather, they employ general training procedures valid for arbitrary graphs that result in models capable of answering any identifiable query, again with general procedures valid for any other graph. Also note that our model architecture can be interchanged for any other, providing high modularity that could even obtain better results than those achieved here. We expect future work to provide extensions of our technique with more powerful model architectures.

C. MODELLING ERROR VS. ESTIMATION ERROR

Section II-D explains why a correct estimation of an identifiable Causal Query is viable using our techniques if the model follows the same Causal Structure $\mathcal{G}_{\mathcal{M}}$ and the same observational distribution $\mathcal{P}(\mathcal{V})$. Although Neural Networks are universal approximators, in practice we cannot realistically achieve the exact same distribution, only approximate it. Further work on this subject should study the effect of this discrepancy, and the variance of our estimations resulting from error propagation across every node in the graph.

Regarding the former problem, here we perform a minor experiment as a sanity check: will the estimation metrics in the previous benchmark improve throughout training, while getting closer to the underlying distribution $\mathcal{P}(\mathcal{V})$? We study this by means of the scatterplot in Fig. 6, which relates the negative log-likelihood of the test dataset (training loss) with every metric in the former experiment (including the estimations with Y_f, denoted with a star (*)). Although this does not prove that diminishing modelling errors entail better estimations, it does corroborate the hypothesis, at least for this experiment.

V. BIKE SHARING DATASET EXAMPLE

For this last section, we showcase our techniques with the Bike Sharing Dataset [4], which contains the number of bikes used on a Bike Rental service in Washington, D.C., USA on a daily basis between 2011 and 2012 (731 samples), along with weather data for each day. Our aim here is to study the effect of temperature (T) on bike rental (Y), assuming that the underlying DGP follows the causal graph shown in Fig. 7. Here, *season* and *weather* are Categorical variables (4 and 3 levels respectively); *working day* is a Bernoulli variable; *temperature*, *humidity*, *windspeed* and *feeling temperature* are continuous variables normalized to the $(0, 1)$ interval, and *bikes* is the target variable, a count of rented bikes on a given day (which we assume to be continuous for ease of modelling). We define a latent confounder between *humidity* and *wind speed* to reflect weather factors that affect both but are not captured by the Categorical variable *weather*. Note that this is just an example; we do not wish to make any claims regarding the real underlying graph. Henceforth, we assume this graph to be valid only for illustrative purposes.

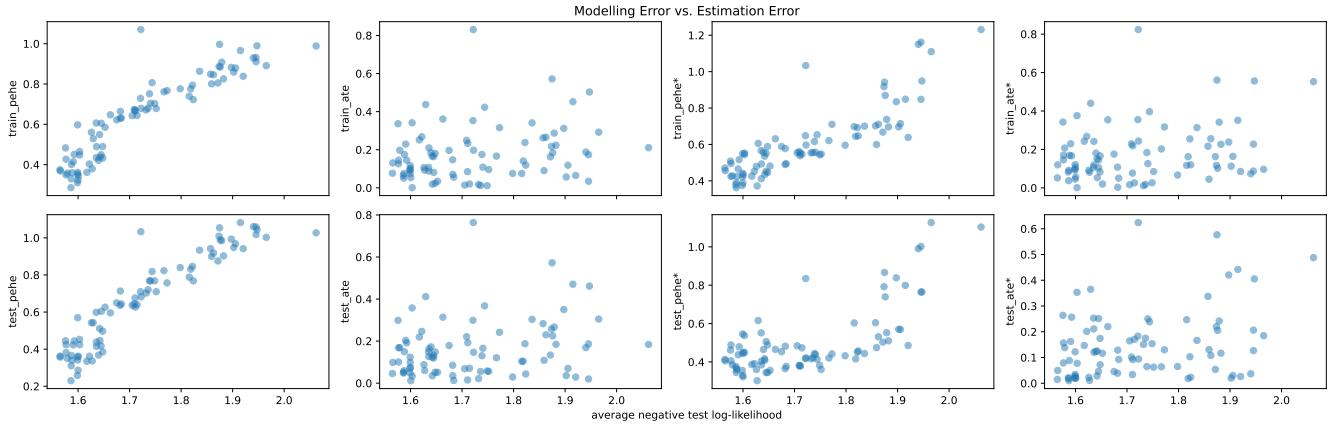


FIGURE 6: Average negative test log-likelihood vs. estimation errors throughout training (lower is better), for the IHDP benchmark, first replication, first CV split. As expected, lower modelling error leads to lower estimation error, and especially in PEHE-related metrics.

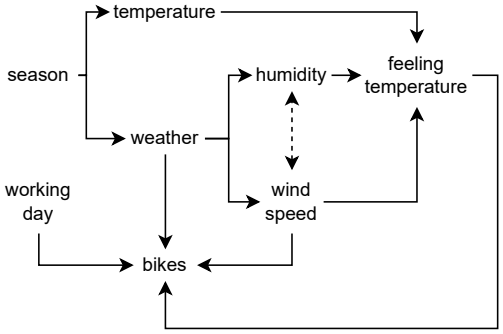


FIGURE 7: Bike Sharing Dataset, proposed Causal Graph.

1) DCG Definition and Training

First we need to decide which DCU implementation will be employed for each node. To illustrate the potential of the different DCUs mentioned in this work, we train two different models, one consisting of Compound-DCNs and one with NCFs (but DCNs for discrete variables). For the former case, since *temperature*, *humidity*, *wind speed* and *feeling temperature* are all bounded to the $(0, 1)$ interval, we employ the Beta Distribution for their variables. As for *bikes*, the counting variable, we use the Normal Distribution (ND). Note that despite modelling the node with an ND, it is only an ND when conditioned by all its parent values; when we marginalize those conditioning terms, the result is a mixture of Normals, which is far more expressive than first assumed. Regarding the NCF implementation, we use two different flows to account for the change in domain, but this adjustment requires no special structure in their flows (in fact, both use the same Transformer architecture).

To showcase the ease of use of the library, we include a code snippet in Listing 1 (summarized and without imports for brevity) showing how the graph is created and trained in the NCF case. For the full code (including the C-DCN graph), please refer to the supplementary material found together with the library. In the definition string we specify each variable name, an alias for its DCU implementation, its dimensionality and, if any, all of its parent nodes. We also create a latent confounder between humidity and windspeed, u, with its own DCU. After that, we assign each DCU implementation to its alias, pass a network creation function for the Graphical Conditioner, call a warm start process on the graph, and finally train the module with early stopping (100 epochs of patience).

Fig. 8 shows the dataset histogram for all continuous variables and the marginalized likelihood curves evaluated by each DCG ($P(x) = \mathbb{E}_{\mathcal{E}_X^c, \mathcal{U}}\left[P(x \mid \mathcal{E}_X^c, \mathcal{U})\right]$). Although both alternatives are quite similar, C-DCNs struggle with *cnt* (count of rented bikes) due to its multi-modality, whereas NCFs can be as flexible as required given a powerful flow architecture. The choice of DCU depends on the complexity of the data and the number of samples (the smaller the dataset, the more overfitting introduced by NCFs; in this case, DCNs or C-DCNs are preferable). In the following section, we will evaluate three identifiable queries with the NCF DCG.

2) Causal Query Estimation

We start with an **interventional query**: the average number of rented bikes when intervening *temperature* by a certain value t. Fig. 9a plots the effect (y-axis) of each intervention $do(T = t)$ (x-axis). We compute this effect in two ways: *back-door*, using the back-door formula[9] $\mathbb{E}_{Y_t}\left[Y\right] = \mathbb{E}_S\left[\mathbb{E}_{Y|T=t,S}\left[Y\right]\right]$; and *dcg*, computing the causal effect through the DCG itself. For the first case, we use Random Forests to model each term needed to compute the query. As can be seen from the figure, there are no significant differences between the two curves, which shows that, despite

[9]The back-door formula can be used under certain graph structures to define an estimand for the estimation of interventional effects.

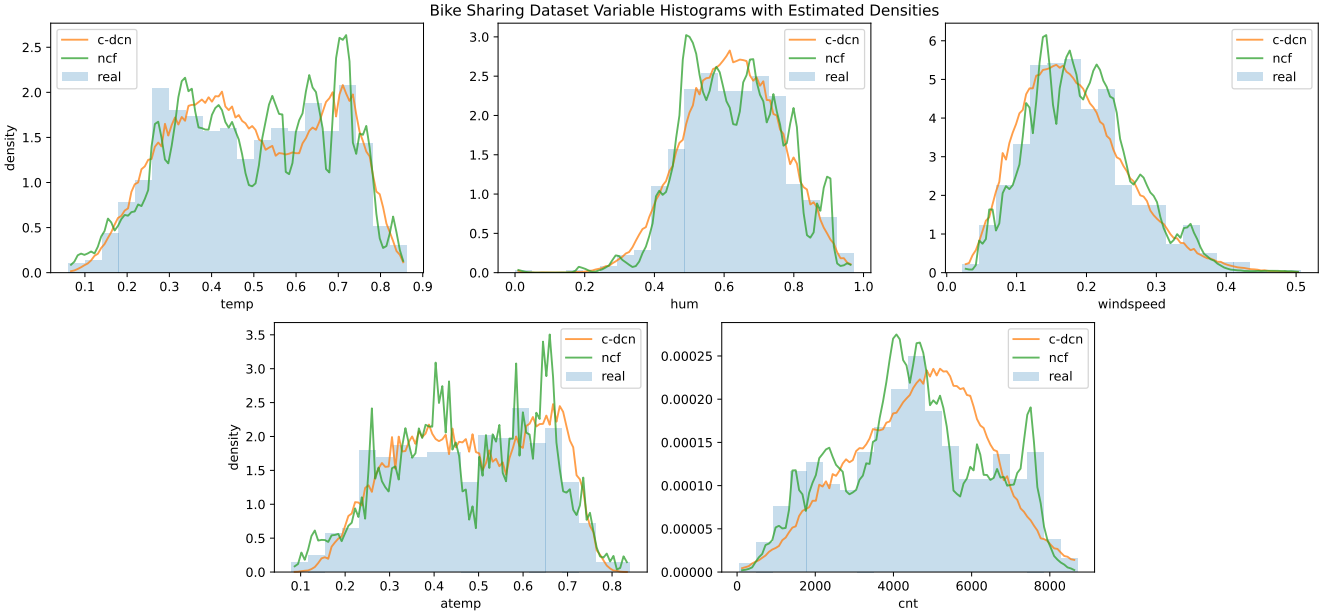
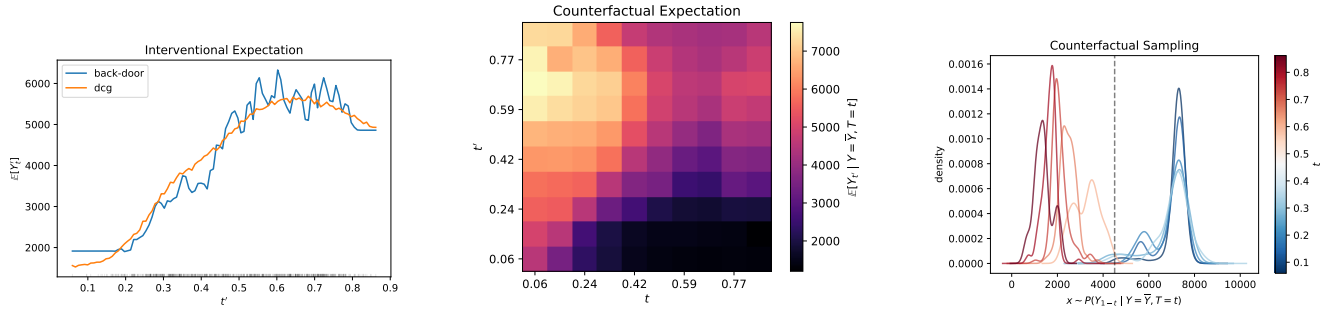


FIGURE 8: Bike Sharing Dataset, variable histograms (blue) with DCG densities: Compound-DCN and NCF.



(a) (b) (c)

FIGURE 9: Bike Sharing Dataset, query visualizations; (a) shows the interventional effect of *temperature* t on *rentals* Y_t, estimated with DCGs and the back-door formula for comparison; (b) shows the counterfactual effects of intervened *temperature* t' on *rentals* when we have observed an average number of *rentals* and different values of *temperature* t; (c) shows KDE curves resulting from counterfactual sampling, where we observe *temperature* t and an average number of *rentals*, but we intervene on *temperature* $1 - t$.

following a completely different strategy to the back-door approach, DCGs still manage to answer the query reliably.

Next, we study a **counterfactual query**: the expected number of rented *bikes* intervened by *temperature* with value $do(T = t')$, when we have already observed a different *temperature* $T = t$ and the rented *bikes* being the average number ($Y = \overline{Y}$). Fig. 9b shows a heatmap of this counterfactual quantity (colour is the counterfactual value) when we have observed a certain t value (x-axis) and intervened on *temperature* with a different t' (y-axis). From the previous query, we know that Y and interventions on T are directly correlated, which tells us that days with higher temperatures should expect above-average counts of bikes, whereas days with lower temperatures expect below-average counts. However, an average number was observed in both cases. This means that higher values of t start from an average Y, and can only go down as the intervention reduces temperature. Conversely, lower values of t start from an average count and go to the highest values as t' increases. In layman's terms, if today was a cold day but we still had an average number of rentals, this must mean that it was a busier day than usual, and increasing the temperature can only result in higher rentals.

Finally, Fig. 9c showcases an example of **counterfactual sampling**. Having again observed an average count of rentals ($Y = \overline{Y}$) and a certain *temperature* t, we intervene by inverting the *temperature* $do(T = 1 - t)$ (remember that T is normalized to $(0, 1)$) and obtain samples from the counterfactual variable Y_{1-t}. We use Kernel Density Estimation to plot the density curves from these samples, so we can see the effect of each value t and the corresponding $1 - t$

```
graph = CausalGraph.from_definition(
    CausalGraph.parse_definition('''
        season cat 4
        work bern 1
        wthr cat 3 season
        tmp cont01 1 season
        hum cont01 1 wthr u
        wspd cont01 1 wthr u
        atmp cont01 1 tmp hum wspd
        y cont 1 work wthr wspd atmp

        u lat 1
    ''',
    cat=Categorical, bern=Bernoulli,
    cont01=NCF01, cont=NCFreal,
    lat=LatentNormal
    ),
    net_f=net_f
).warm_start(Xtrain).to(device)

train(
    graph,
    data_loader(TensorDataset(Xtrain)),
    data_loader(TensorDataset(Xval)),
    loss_f=loss_f(ex_n=100),
    patience=100
)
```

Listing 1: Code example for creating and training a DCG.

all at once. t is coded by colour, with lower temperatures in blue and higher temperatures in red. As we can see, observing low temperatures t (blue) means intervening with high temperatures, which increases the number of rentals, resulting in above-average values (past the dashed vertical line). Higher temperatures (red), on the other hand, are intervened on with lower temperatures, which can only result in lower counts. The fact that we can obtain these counterfactual samples allows for extra flexibility in our causal studies. We are not only restricted to expectation studies, but can, for example, look for multi-modalities or asymmetries in their distributions.

VI. ASSUMPTIONS AND LIMITATIONS

This section is devoted to summarizing all assumptions and limitations of the technique described throughout this work.

Given an i.i.d. dataset D resulting from an underlying SCM $\mathcal{M}$ with graph $\mathcal{G}_\mathcal{M}$, we assume that:

- The graph structure $\mathcal{G}_\mathcal{M}$ of the underlying SCM of interest $\mathcal{M}$ is known, and it is a Directed Acyclic Graph.
- Any latent confounders $\mathcal{U}$ in $\mathcal{M}$ are root nodes with exactly two children. Otherwise, we use a projection of the graph (see footnote 3).
- The observational distribution $\mathcal{P}(\mathcal{V})$ resulting from $\mathcal{M}$ is positive for all its values.
- Any queries to be estimated are identifiable *w.r.t.* graph $\mathcal{G}_\mathcal{M}$; we can determine identifiability with the algorithms mentioned in section II-D.
- Any non-atomic interventions (other than $do(\mathbf{X} = \mathbf{x})$) call for a reevaluation of the required estimation procedures.

Regarding limitations of the current state of the technique, we present the following challenges:

- We can never achieve a perfect match between our model's distribution $\mathcal{P}_{\mathcal{M}'}(\mathcal{V})$ and the underlying $\mathcal{P}_\mathcal{M}(\mathcal{V})$; it is, after all, an approximation of that distribution. This mismatch creates a miscalibration effect on the eventual estimations. A study of the relationship between modelling error and estimation error remains for future work.
- In graphs with high depth (the length of the longest path from roots to leafs), errors on node sampling add up level by level, and estimations might be affected by this compounding error. Although this is related to the previous point, as these errors come as a result of miscalibration with the original observational distribution, graph depth also amplifies the problem. A quantification of these effects is left as an open question.
- Our main assumption in this work is that the underlying graph $\mathcal{G}_\mathcal{M}$ is a Directed Acyclic Graph. However, most real-world problems deal with cyclic graphs. In their current state, DCGs do not work with these kinds of graphs, and an extension of the technique to this setting is a promising research avenue to explore.

VII. CONCLUSION

Most causal query estimation methods operate from an estimand-based perspective, making them extremely ad-hoc. Here, we propose Deep Causal Graphs (DCG), a flexible, general, modular framework capable of answering causal queries without an estimand, as long as they are identifiable.

These techniques operate on complex real-world distributions, thanks to our implementations with Distributional Causal Nodes and Normalizing Flows. We also provide Mixture implementations (finite or uncountable) to further extend their modelling capabilities or even operate with small datasets, thanks to Cross Validation combined with mixtures.

Furthermore, we detail DCG procedures to estimate the aforementioned causal queries: these cover observational, interventional and counterfactual queries for estimating likelihoods, expectations and even sample from these distributions, with or without conditionals. These procedures are graph-agnostic, i.e. they can be applied to arbitrary graphs as long as the target query is identifiable.

We demonstrate the modelling capabilities and estimation performance of our framework in two experiments, and provide a complete study to exemplify a potential application of our technique. We also provide a software library ready for practitioners and researchers alike, thanks to its ease of use and high modularity.

We expect this work to help spread the use of estimand-agnostic methods, especially among non-expert practitioners thanks to our library, so that we can eventually build a landscape of Causal Query Estimation frameworks similar to the one found in the Machine Learning field.

Further work could provide alternative implementations for the Deep Causal Unit specification, study the sensitivity of query estimations to miscalibrations on modelling the observational distribution, or study how to incorporate cyclic Causal Graphs to extend the applicability of our techniques.

References

[1] Axel Brando, Jose A Rodriguez, Jordi Vitria, and Alberto Rubio Muñoz. "Modelling heterogeneous distributions with an Uncountable Mixture of Asymmetric Laplacians". In: *Advances in neural information processing systems* 32 (2019).

[2] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[3] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. "Neural spline flows". In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 7511–7522.

[4] Hadi Fanaee-T and Joao Gama. "Event labeling combining ensemble detectors and background knowledge". In: *Progress in Artificial Intelligence* (2013), pp. 1–15.

[5] Miguel A Hernán and James M Robins. *Causal Inference: What If*. Boca Raton: Chapman & Hall/CRC, 2020.

[6] Jennifer L Hill. "Bayesian nonparametric modeling for causal inference". In: *Journal of Computational and Graphical Statistics* 20.1 (2011), pp. 217–240.

[7] Guido W Imbens and Donald B Rubin. *Causal inference in statistics, social, and biomedical sciences*. Cambridge University Press, 2015.

[8] Fredrik Johansson, Uri Shalit, and David Sontag. "Learning representations for counterfactual inference". In: *International conference on machine learning*. PMLR. 2016, pp. 3020–3029.

[9] Diederik P. Kingma and Max Welling. "Auto-encoding variational Bayes". In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2014.

[10] Murat Kocaoglu, Christopher Snyder, Alexandros G. Dimakis, and Sriram Vishwanath. "CausalGAN: Learning Causal Implicit Generative Models with Adversarial Training". In: *International Conference on Learning Representations (ICLR)*. 2018.

[11] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. "Counterfactual fairness". In: *Advances in neural information processing systems* 30 (2017).

[12] Robert J LaLonde. "Evaluating the econometric evaluations of training programs with experimental data". In: *The American economic review* (1986), pp. 604–620.

[13] Qian Li, Zhichao Wang, Shaowu Liu, Gang Li, and Guandong Xu. "Causal Optimal Transport for Treatment Effect Estimation". In: *IEEE transactions on neural networks and learning systems* (2021).

[14] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[15] Christos Louizos, Uri Shalit, Joris M Mooij, David Sontag, Richard Zemel, and Max Welling. "Causal effect inference with deep latent-variable models". In: *Advances in neural information processing systems* 30 (2017).

[16] Chris J Maddison, Daniel Tarlow, and Tom Minka. "A* sampling". In: *Advances in Neural Information Processing Systems* 27 (2014).

[17] Chris J Maddison and Danny Tarlow. *Gumbel Machinery*. https://cmaddis.github.io/gumbel-machinery. Jan. 2017.

[18] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. "Normalizing flows for probabilistic modeling and inference". In: *Journal of Machine Learning Research* 22.57 (2021), pp. 1–64.

[19] George Papamakarios, Theo Pavlakou, and Iain Murray. "Masked autoregressive flow for density estimation". In: *Advances in neural information processing systems* 30 (2017).

[20] Álvaro Parafita and Jordi Vitrià. "Causal Inference with Deep Causal Graphs". In: *arXiv preprint arXiv:2006.08380* (2020).

[21] Álvaro Parafita and Jordi Vitrià. "Explaining visual models by causal attribution". In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE. 2019, pp. 4167–4175.

[22] Nick Pawlowski, Daniel Coelho de Castro, and Ben Glocker. "Deep Structural Causal Models for Tractable Counterfactual Inference". In: *Advances in Neural Information Processing Systems* 33 (2020).

[23] Judea Pearl. *Causality*. Cambridge university press, 2009.

[24] Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect*. Basic books, 2018.

[25] Martı Pedemonte, Jordi Vitrià, and Álvaro Parafita. "Algorithmic Causal Effect Identification with causal-effect". In: *arXiv preprint arXiv:2107.04632* (2021).

[26] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

[27] Donald B Rubin. "Causal inference using potential outcomes: Design, modeling, decisions". In: *Journal of the American Statistical Association* 100.469 (2005), pp. 322–331.

[28] Pablo Sanchez-Martin, Miriam Rateike, and Isabel Valera. "VACA: Design of Variational Graph Autoencoders for Interventional and Counterfactual Queries". In: *arXiv preprint arXiv:2110.14690* (2021).

[29] Uri Shalit, Fredrik D Johansson, and David Sontag. "Estimating individual treatment effect: generalization bounds and algorithms". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3076–3085.

[30] Ilya Shpitser and Judea Pearl. "Identification of conditional interventional distributions". In: *22nd Conference on Uncertainty in Artificial Intelligence, UAI 2006*. 2006, pp. 437–444.

[31] Ilya Shpitser and Judea Pearl. "Identification of joint interventional distributions in recursive semi-Markovian causal models". In: *Proceedings of the National Conference on Artificial Intelligence*. Vol. 21. 2. 2006, p. 1219.

[32] Ilya Shpitser and Judea Pearl. "What counterfactuals can be tested". In: *23rd Conference on Uncertainty in Artificial Intelligence, UAI 2007*. 2007, pp. 352–359.

[33] Peter Spirtes and Kun Zhang. "Causal discovery and inference: concepts and recent methodological advances". In: *Applied informatics*. Vol. 3. 1. SpringerOpen. 2016, pp. 1–28.

[34] Jin Tian and Judea Pearl. "On the testable implications of causal models with hidden variables". In: *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*. 2002, pp. 519–527.

[35] Santtu Tikka and Juha Karvanen. "Identifying Causal Effects with the R Package causaleffect". In: *Journal of Statistical Software* 76.i12 (2017).

[36] Sahil Verma, John Dickerson, and Keegan Hines. "Counterfactual explanations for machine learning: A review". In: *arXiv preprint arXiv:2010.10596* (2020).

[37] Antoine Wehenkel and Gilles Louppe. "Graphical normalizing flows". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 37–45.

[38] Sewall Wright. "Correlation and causation". In: *Journal of Agricultural Research* (1921).

[39] Kevin Xia, Kai-Zhan Lee, Yoshua Bengio, and Elias Bareinboim. "The causal-neural connection: Expressiveness, learnability, and inference". In: *Advances in Neural Information Processing Systems* 34 (2021).

[40] Liuyi Yao, Sheng Li, Yaliang Li, Mengdi Huai, Jing Gao, and Aidong Zhang. "Ace: Adaptively similarity-preserved representation learning for individual treatment effect estimation". In: *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2019, pp. 1432–1437.

[41] Liuyi Yao, Yaliang Li, Sheng Li, Mengdi Huai, Jing Gao, and Aidong Zhang. "SCI: Subspace Learning Based Counterfactual Inference for Individual Treatment Effect Estimation". In: *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2021, pp. 3583–3587.

ÁLVARO PARAFITA received the B.S. degree in Computer Science and in Mathematics from Universitat de Barcelona (UB), Barcelona, Spain, in 2016, and the M.S. degree in Innovation and Research in Informatics, specialized in Data Science, from Universitat Politècnica de Catalunya, Barcelona, Spain, in 2018. He is currently pursuing the Ph.D. degree in Mathematics and Informatics at UB.

From 2013 to 2016, he worked as a Data Analyst in the Marketing field. From 2016 to 2018, he was a Freelance Data Scientist, working on developing Machine Learning solutions for several companies. His current research interest focuses on applying Causal Theory to Explainability and Fairness applications.

JORDI VITRIÀ is a Full Professor at Universitat de Barcelona (UB), which he joined in 2007, and where he teaches an introductory course on Algorithms and advanced courses on Data Science and Deep Learning. From April 2011 to January 2016 he served as Head of the UB's Applied Mathematics and Analysis Department. He is now a member of the Mathematics & Computer Science Department at UB.

His research, when personal computers had 128KB of memory, was originally oriented towards digital image analysis and how to extract quantitative information from them, but soon evolved towards computer vision problems. Now, he is leading a research group working in causal inference, computer vision and machine learning. He has authored more than 100 peer-reviewed papers and holds 8 international patents. He has directed 14 PhD theses in the area of machine learning and computer vision. He has been the leader of a large number of research projects at international and national level.