NESTER: An Adaptive Neurosymbolic Method for Causal Effect Estimation

Abbavaram Gowtham Reddy, Vineeth N Balasubramanian

Indian Institute of Technology Hyderabad, India
cs19resch11002@iith.ac.in, vineethnb@iith.ac.in

Abstract

Causal effect estimation from observational data is a central problem in causal inference. Methods based on potential outcomes framework solve this problem by exploiting inductive biases and heuristics from causal inference. Each of these methods addresses a specific aspect of causal effect estimation, such as controlling propensity score, enforcing randomization, etc., by designing neural network (NN) architectures and regularizers. In this paper, we propose an adaptive method called Neurosymbolic Causal Effect Estimator (NESTER), a generalized method for causal effect estimation. NESTER integrates the ideas used in existing methods based on multi-head NNs for causal effect estimation into one framework. We design a Domain Specific Language (DSL) tailored for causal effect estimation based on causal inductive biases used in literature. We conduct a theoretical analysis to investigate NESTER's efficacy in estimating causal effects. Our comprehensive empirical results show that NESTER performs better than state-of-the-art methods on benchmark datasets.

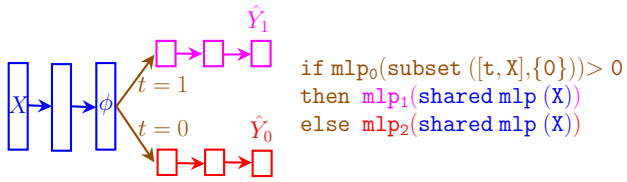
1 Introduction

Causal effect estimation measures the effect of a treatment variable on an outcome variable (e.g., the effect of a medicine on recovery). Randomized Controlled Trials (RCTs), where individuals are randomly split into *treated* and *control* groups, are considered the gold standard approach for causal effect estimation (Chalmers et al. 1981). However, RCTs are often: (i) unethical (e.g., to find the causal effect of smoking on lung disease, a randomly chosen person cannot be forced to smoke), and/or (ii) impossible/infeasible (e.g., in finding the causal effect of blood pressure on the risk of an adverse cardiac event, it is impossible to intervene on the same patient with and without high blood pressure with all other parameters the same) (Sanson-Fisher et al. 2007; Carey and Stiles 2016). These limitations leave us with observational data to compute causal effects.

Observational data, similar to RCTs, suffer from *the fundamental problem of causal inference* (Pearl 2009), *viz.* for any individual, we cannot observe all potential outcomes at the same time (e.g., we cannot uniquely record the same person's medical condition at a given time to two different treatments individually, say, on consuming a medicinal drug and an alternate placebo). Observational data also suffers from *selection bias* (e.g., certain age groups are more likely to take certain kinds of medication compared to other age groups) (Collier and Mahoney 1996). For these reasons, estimating unbiased causal effects from observational data is challenging. However, due to the many use cases in the real-world, estimating causal effects from observational data has remained an important problem in causal inference (Rosenbaum and Rubin 1983, 1985; Brady, Collier, and Sekhon 2008; Morgan and Winship 2014), with recent efforts leveraging learning-based methods to this end (Curth and van der Schaar 2021a).

Simpson's paradox (Pearl, Glymour, and Jewell 2016) underpins the necessity of choosing the correct set of features to *control/adjust* for estimating causal effects from observational data. The Pearlian framework (Pearl 2009) uses graphical criteria such as *back-door* criterion and *front-door* criterion depending on the available adjustment variables and identifiability conditions. However, the Pearlian framework requires knowledge of the underlying causal graph, which is not feasible for many real-world scenarios. On the other hand, under the *ignorability* assumption, methods based on the *Rubin-Neyman potential outcomes framework* (Rubin 1974), assume that a known set of observed features to control is available. However, as discussed above, observational data suffers from issues such as selection bias, leading to biased estimates of causal effects. Various methods have been proposed to address one or more of these issues in recent literature (Shalit, Johansson, and Sontag 2017; Shi, Blei, and Veitch 2019; Curth and van der Schaar 2021a).

This paper provides a pathway to integrate existing solutions based on the potential outcomes framework, especially the methods based on multi-head neural network (NN) architectures with regularizers, into a single framework. We propose an adaptive method called NEuroSymbolic causal effecT EstimatoR (NESTER) that automatically synthesizes a neurosymbolic program for estimating causal effects. Synthesized programs by NESTER can instantiate existing methods based on multi-head NN architectures with regularizers as special cases. For example, as shown in Fig 1, NESTER can synthesize a program $\mathcal{P}_T$ that is functionally similar to that of TARNet architecture (Shalit, Johansson, and Sontag 2017). The two NN heads of TARNet corresponding to the two treatment values $t = 1, t = 0$ can be seen as implementing an



(a) $TARNet$ (b) $\mathcal{P}_T$

Figure 1: (a) TARNet architecture. $\mathbf{X}$ is feature vector, t is treatment, $\hat{Y}_1, \hat{Y}_0$ are estimated potential outcomes and ϕ is learned representation at the end of shared layers. (b) Program $\mathcal{P}_T$ synthesized by NESTER using our Domain-Specific Language (DSL) (Tab 1) that is functionally similar to TARNet. Colors are used to show the equivalence between the components of TARNet and $\mathcal{P}_T$.

$\texttt{if} - \texttt{then} - \texttt{else}$ program primitive (see § 4 for detailed explanation). In $\mathcal{P}_T$, if '$\texttt{mlp}_0(\texttt{subset}([\texttt{t}, \mathbf{X}], \{0\})) > 0$', '$\texttt{then}$' clause gets executed otherwise '$\texttt{else}$' clause gets executed. Subscripts $1, 2$ in $\texttt{mlp}_1, \texttt{mlp}_2$ indicate two different instantiations of the primitive $\texttt{mlp}$ (acronym for multilayer perceptron) corresponding to the two heads of TARNet. $\texttt{shared mlp}$ primitives, without subscripts, indicate a common primitive whose output is shared by $\texttt{mlp}_1, \texttt{mlp}_2$ similar to how two heads of TARNet share the representation ϕ.

As part of our proposed method NESTER, we design a Domain-Specific Language (DSL) (a specific *context-free grammar*) of program primitives such as $\texttt{if} - \texttt{then} - \texttt{else}, \texttt{mlp}$ containing learnable components, which is then used by neurosymbolic program synthesis (NPS) module to synthesize programs. This process is equivalent to assembling the primitives in our DSL to obtain a model architecture/workflow. In other words, NESTER learns to adaptively synthesize differentiable programs for a given set of input-output examples, wherein the sequence of learnable program modules provides an overall network architecture used to estimate causal effects.

NPS methods synthesize programs using a DSL of program primitives that satisfy given observational data of input-output pairs so that the synthesized programs generalize well to unseen inputs (see Appendix § E, F for examples) (Biermann 1978; Gulwani 2011; Parisotto et al. 2017). Recently, various NN-based techniques have been proposed to perform NPS (Parisotto et al. 2017; Valkov et al. 2018; Gaunt et al. 2017; Bošnjak et al. 2017; Shi et al. 2022; Tang and Ellis 2022; Shah et al. 2020; Cui and Zhu 2021). We use an NPS paradigm where each program primitive is a differentiable module (Shah et al. 2020; Cui and Zhu 2021). Such *differentiable programs* simultaneously optimize program primitive parameters while learning the overall program structure. In this paper, to synthesize programs, we use (i) *neural admissible relaxation* (NEAR) (Shah et al. 2020), which uses NNs as relaxations of partial programs while searching the program space and (ii) *domain-specific program architecture differentiable synthesis* (dPads) (Cui and Zhu 2021) that improves on time complexity by learning the probability distribution of program architectures in a continuous relaxation of the search space of DSL grammar rules. Our key contributions are summarized below.

- We develop an adaptive neurosymbolic method that can learn to synthesize programs for estimating causal effects. Such a method is not restricted by its architecture and is easy to implement and extend. To the best of our knowledge, this is the first neurosymbolic approach to estimate causal effects.
- We propose a domain-specific language (DSL) for causal effect estimation, whose program primitives are inspired by causal effect estimation efforts in literature.
- We theoretically study the universal approximation ability of a synthesized neurosymbolic program and show how this provides a pathway for our method for causal effect estimation. We show that the proposed method can be viewed as a generalization of causal effect estimation methods based on multi-head NN architectures.
- We perform comprehensive empirical studies on multiple benchmark datasets where NESTER outperforms existing state-of-the-art models.

2 Related Work

Matching and Covariate Adjustment Methods: Early methods for causal effect estimation are primarily based on matching techniques (Brady, Collier, and Sekhon 2008; Morgan and Winship 2014) where similar samples in treatment and control groups are compared using methods such as nearest neighbor matching (Stuart 2010) and propensity score matching (Rosenbaum and Rubin 1983). For example, in nearest neighbor matching, for each sample in the treatment group, the nearest point from the control group w.r.t. Euclidean distance is identified, and the difference in observed outcomes between the treatment and corresponding control group samples is the estimate of causal effect. However, such matching techniques do not scale to high-dimensional data (Abadie and Imbens 2006). Another family of methods estimates causal effects using covariate adjustment (Pearl 2009). Assuming the availability of a sufficient adjustment set, these models fit conditional probabilities given the treatment variable and a sufficient adjustment set. Such models are, however, known to suffer from high variance in the estimated causal effects (Shalit, Johansson, and Sontag 2017). Covariate balancing through weighting is another technique to control the confounding bias in estimating causal effects (Rosenbaum and Rubin 1983; CRUMP et al. 2009; Diamond and Sekhon 2013; Li and Fu 2017). As noted in (Assaad et al. 2021), such methods face challenges with large weights and high-dimensional inputs. Besides, leveraging the success of learning-based methods has yielded significantly better performance in recent years.

Representation Learning-based Methods: Recent methods to estimate causal effects are largely been based on multi-head NN architectures (NN architectures which branch out into different heads for different treatments) equipped with regularizers (Shalit, Johansson, and Sontag 2017; Shi, Blei, and Veitch 2019; Curth and van der Schaar 2021b; Schwab et al. 2020; Chu, Rathbun, and Li 2020). CFR (Shalit, Johansson, and Sontag 2017) is a two-headed NN architecture with a

regularizer that forces latent representations of treatment and control groups close to each other to adjust confounding features. Dragonnet (Shi, Blei, and Veitch 2019) is a two-headed NN architecture with a regularizer that predicts treatment value from latent representations; this allows pre-treatment covariates to be used in predicting potential outcomes. Considering multiple treatment values and continuous dosage for each treatment, (Schwab et al. 2020) devised an NN architecture with multiple heads for multiple treatments, and multiple sub-heads from each of the treatment-specific heads to model (discretized) dosage values. Assuming that potential outcomes are strongly related, (Curth and van der Schaar 2021a,b) proposed techniques that improve existing models using the structural similarities between potential outcomes. These methods, however, have a fixed architecture design, and each addresses a specific problem in estimating causal effects. Our approach is also NN-based but uses a neurosymbolic approach to automatically synthesize an architecture, allowing it to generate different programs for different observational data. Causal discovery-based effect estimation and generative modeling-based effect estimation are discussed in Appendix § H along with the differences between NPS and neural architecture search (NAS).

Neurosymbolic Program Synthesis (NPS): Program synthesis, *viz.* automatically learning a program that satisfies given input-output pairs (Biermann 1978), is helpful in diverse tasks such as low-level bit manipulation code (Solar-Lezama et al. 2005), data structure manipulations (Solar Lezama 2008), and regular expression-based string generation (Gulwani 2011). For each task, a specific DSL is used to synthesize programs. Even with a small DSL, the number of programs that can be synthesized is very large. Several techniques such as greedy enumeration, Monte Carlo sampling, Monte Carlo tree search (Kocsis and Szepesvári 2006), evolutionary algorithms (Valkov et al. 2018), and recently, node pruning with neural admissible relaxation (NEAR) (Shah et al. 2020) have been proposed to search for optimal programs from a vast search space efficiently. Improving NEAR, dPads (Cui and Zhu 2021) propose differentiable program architecture synthesis that avoids the problem of combinatorial search required to find optimal programs. We use both NEAR and dPads to implement our method.

3 Background and Problem Formulation

Let $\mathcal{D} = \{(t_i, \mathbf{x}_i, y_i)\}_{i=1}^N$ be a set of N observational data points. $t_i \in \mathbb{R}$ denotes the treatment variable, $\mathbf{x}_i \in \mathbb{R}^n$ denotes the $n-$dimensional feature vector, and $y_i \in \mathbb{R}$ denotes the observed potential outcome. Each $(t_i, \mathbf{x}_i, y_i)$ is randomly sampled from $p(T, \mathbf{X}, Y)$, where $T, \mathbf{X}$, and Y are the corresponding random variables. In a binary treatment setting ($t \in \{0, 1\}$), for the i^{th} observation, let Y_i^0 denote the true potential outcome under treatment $t_i = 0$ and Y_i^1 denote the true potential outcome under treatment $t_i = 1$. Because of *the fundamental problem of causal inference*, we observe only one of Y_i^0, Y_i^1 for a given $[t_i, \mathbf{x}_i]$. Hence, y_i can be expressed in terms of Y_i^0, Y_i^1 as $y_i = t_i Y_i^1 + (1 - t_i) Y_i^0$. One of the goals in causal effect estimation from observational data is to learn an estimator $f(t, \mathbf{x})$ such that the difference in estimated potential outcomes under the treatments $t = 1$ and $t = 0$, $f(1, \mathbf{x}_i) - f(0, \mathbf{x}_i)$, is close to the difference in true potential outcomes: $Y_i^1 - Y_i^0;\ \forall i$. This difference for a specific instance i is called the *Individual Causal Effect (ICE)*.

Definition 3.1. *The Individual Causal Effect (ICE) of T on Y for an instance $\mathbf{x} \sim \mathbf{X}$ is defined as*

$$ICE_T^Y(\mathbf{x}) \coloneqq \mathbb{E}[Y^1 - Y^0 | \mathbf{x}] \tag{1}$$

Definition 3.2. *The expected Precision in Estimation of Heterogeneous Effect (ϵ_{PEHE}) using $f(t, \mathbf{x})$ is defined as*

$$\epsilon_{PEHE}(f) \coloneqq \mathop{\mathbb{E}}_{\mathbf{x} \sim \mathbf{X}} \left[\left((f(1, \mathbf{x}) - f(0, \mathbf{x})) - ICE_T^Y(\mathbf{x}) \right)^2 \right] \tag{2}$$

Extending *ICE* to an entire population, our goal is to estimate the *Average Causal Effect (ACE)* (Pearl 2009) of the treatment variable T on the outcome variable Y.

Definition 3.3. *The Average Causal Effect (ACE) of T on Y is defined as*

$$ACE_T^Y \coloneqq \mathbb{E}[Y^1] - \mathbb{E}[Y^0] \tag{3}$$

Definition 3.4. *The error in estimation of Average Causal Effect (ϵ_{ACE}) using $f(t, \mathbf{x})$ is defined as*

$$\epsilon_{ACE}(f) \coloneqq |\mathop{\mathbb{E}}_{\mathbf{x} \sim \mathbf{X}}[f(1, \mathbf{x}) - f(0, \mathbf{x})] - ACE_T^Y| \tag{4}$$

$\mathbb{E}[Y^t]$ refers to the expected value of Y when every instance in the population is given the treatment t (if t is not binary, causal effects are calculated w.r.t. a baseline treatment value t^* i.e., $1, 0$ in Defn 3.3 are replaced with t, t^* respectively). To estimate the quantities in Eqns 1-4 from observational data, it is required to guarantee *identifiability*. Following (Shalit, Johansson, and Sontag 2017; Lechner 2001; Imbens 2000; Schwab et al. 2020; Zhang, Liu, and Li 2021), we make the following assumptions sufficient to guarantee the *identifiability* of causal effects.

Assumptions 3.1. ***(i) Ignorability*** *(No-latent-confounding): For a given set of pre-treatment covariates, treatment is randomly assigned, i.e., conditioned on a set of pre-treatment covariates $\mathbf{X}$, T is independent of Y^0, Y^1 i.e., $((Y^0, Y^1) \perp\!\!\!\perp T | \mathbf{X})$.* ***(ii) Positivity****: Treatment assignment for each instance is not deterministic, and it must be possible to assign all treatments to each instance, i.e., $0 < p(t|\mathbf{x}) < 1 \ \forall t, \mathbf{x}$.* ***(iii) Stable Unit Treatment Value Assumption (SUTVA))****: The observed outcome of any instance under a treatment must be independent of the treatment assignment to other individuals.*

Assuming the feature vector $\mathbf{X}$ satisfies the properties in Assumptions 3.1, we can write $\mathbb{E}[Y^t] = \mathbb{E}_{\mathbf{x} \sim \mathbf{X}} [\mathbb{E}[Y | T = t, \mathbf{X} = \mathbf{x}]]$, called the *adjustment formula* (Pearl 2009). Using this, a simple technique to estimate $\mathbb{E}[Y | T = t, \mathbf{X} = \mathbf{x}]$ is to fit a regression model for Y given T and $\mathbf{X}$. Estimation of $\mathbb{E}[Y | T = t, \mathbf{X} = \mathbf{x}]$ is the primary task of many causal effect estimation methods. In the same vein, we also aim to synthesize programs that compute the quantity $\mathbb{E}[Y | T = t, \mathbf{X} = \mathbf{x}]$. Once $\mathbb{E}[Y | T = t, \mathbf{X} = \mathbf{x}]$ is estimated, the ATE_T^Y can be estimated as $ATE_T^Y = \mathbb{E}_{\mathbf{x} \sim \mathbf{X}}[\mathbb{E}[Y | T = 1, \mathbf{X} = \mathbf{x}]] - \mathbb{E}_{\mathbf{x} \sim \mathbf{X}}[\mathbb{E}[Y | T = 0, \mathbf{X} = \mathbf{x}]]$.

Neurosymbolic Program Synthesis (NPS): Let $(\mathcal{P}, \theta)$ be a neurosymbolic *program* where $\mathcal{P}$ denotes the program *structure* and θ denotes the program *parameters*. $(\mathcal{P}, \theta)$ is differentiable in θ. $\mathcal{P}$ is synthesized using a Context-Free Grammar

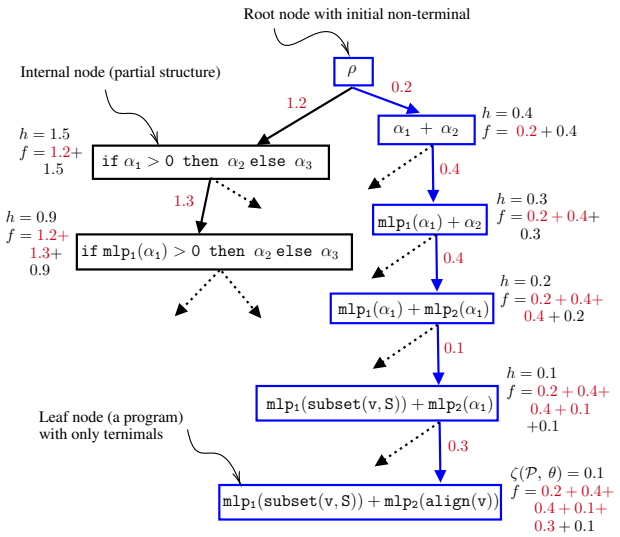


Figure 2: Example program tree generated using DSL in Tab 1. Structural costs are shown in red color (e.g., $s(r) = 0.2$ for the rule $r : \rho \rightarrow \alpha_1 + \alpha_2$). h is the heuristic value, and f is the sum of structural cost and heuristic value. The path from the root node to a leaf node returned by A^* algorithm is shown in blue color.

(CFG) (Hopcroft, Motwani, and Ullman 2001) (which is a DSL in this work). A CFG consists of a set of *rules* of the form $\rho \rightarrow \alpha_1, \ldots, \alpha_n$ where ρ is a *non-terminal* and $\alpha_1, \ldots, \alpha_n$ are either *non-terminals* or *terminals*. A non-terminal denotes a missing sub-expression in a program structure and a terminal is a symbol that appears in a final program structure. Program synthesis starts with an initial non-terminal, then iteratively applies the rules to produce a series of *partial structures*, *viz.* structures made from one or more non-terminals and zero or more terminals. These partial structures form internal nodes of a *program tree*, and the rules form the (directed) edges connecting these nodes (e.g., a rule r is considered as an edge from node u to node v when v is obtained from u by applying r). The process continues until no non-terminals are left, i.e., we have synthesized a program. The resultant program tree's leaf nodes (a.k.a. goal nodes) contain structures consisting of only terminals (see Fig 2). Let $s(r)$ be the cost incurred in using the rule r for generating a program structure (leaf node) or partial structure (internal node) from a given partial structure. The structural cost of any node u is the sum of the structural costs of rules used to get u from the root node. The structural cost $s(\mathcal{P})$ of a program $\mathcal{P}$ is defined as $s(\mathcal{P}) = \sum_{r \in R(\mathcal{P})} s(r)$, where $R(\mathcal{P})$ is the multiset of rules used to create the structure $\mathcal{P}$. In this paper, we set $s(r)$ to a constant real number for all rules (e.g., $s(r) = 1 \ \forall r \in R(\mathcal{P})$). The program learning problem is usually formulated as a node search problem, i.e., starting with an empty tree, the tree is expanded by creating new partial structures and structures.

We use A^* algorithm to generate a program tree. While generating a program tree, a node u with minimum $f(u)$ value is expanded next, where $f(u) = s(\mathcal{P}(u)) + h(u)$ is the sum of the structural cost $s(\mathcal{P}(u))$ of the partial structure $\mathcal{P}(u)$ in u and the heuristic value $h(u)$ at the node u. $h(u)$ underestimates the cost to reach goal node from u (see Fig 2 and § 4.3). While searching for an optimal program, the program parameters (and program structures) are updated simultaneously along with the synthesis of the programs. Since our goal is to synthesize a program $(\mathcal{P}, \theta)$ that estimates the quantity $\mathbb{E}[Y|T = t, \mathbf{X} = \mathbf{x}]$, which can be modeled as a regression problem, the squared error is a good choice for assessing the performance of the program $(\mathcal{P}, \theta)$ in estimating potential outcomes. Hence, for a synthesized program $(\mathcal{P}, \theta)$, we define $\zeta(\mathcal{P}, \theta) = \mathbb{E}_{(t,\mathbf{x},y)\sim\mathcal{D}}[((\mathcal{P}, \theta)(t, \mathbf{x}) - y)^2]$ as the loss incurred by $(\mathcal{P}, \theta)$ in estimating potential outcomes. The overall goal of NPS is then to find a structurally simple program with low prediction error, i.e., to solve the following optimization problem.

$$(\mathcal{P}^*, \theta^*) = \arg\min_{(\mathcal{P},\theta)} (s(\mathcal{P}) + \zeta(\mathcal{P}, \theta)) \tag{5}$$

$\rho \rightarrow$ `if` $\alpha_1 > 0$ `then` α_2 `else` α_3 | `mlp`(α) | `shared mlp`(α) | `subset`$(\mathbf{v}, \mathbf{S})$ | `propensity`$(\mathbf{v})$ | `align`$(\mathbf{v})$ | $\odot(\alpha_1, \alpha_2)$

$\alpha/\alpha_1/\alpha_2/\alpha_3 \rightarrow$ `if` $\alpha_1 > 0$ `then` α_2 `else` α_3 | `mlp`(α) | `shared mlp`(α) | `subset`$(\mathbf{v}, \mathbf{S})$ | `propensity`$(\mathbf{v})$ | `align`$(\mathbf{v})$ | $\odot(\alpha_1, \alpha_2)$ | $\mathbf{v}$ | $\mathbf{x}$

Table 1: A DSL for the causal effect estimation in Backus-Naur form (Winskel 1993) and its semantics. ρ is the initial non-terminal.

4 NESTER: Methodology

The key idea of our methodology is to design a Domain-Specific Language (DSL) for causal effect estimation and subsequently leverage well-known search algorithms such as A^* to synthesize programs for given observational data. We begin by discussing the proposed DSL and its connections to existing literature, followed by our overall algorithm that uses this DSL to synthesize programs.

4.1 DSL for Causal Effect Estimation

We pose the causal effect estimation problem as the problem of mapping a set of observational input data points to the corresponding observed outcomes. Formally, given $\mathcal{D} = \{(t_i, \mathbf{x}_i, y_i)\}_{i=1}^N$, the set $\{(t_i, \mathbf{x}_i)\}_{i=1}^N$ contains inputs and the set $\{y_i\}_{i=1}^N$ contains outputs. For simplicity, let $\mathbf{v}_i = [t_i; \mathbf{x}_i]$ (concatenation of t_i and $\mathbf{x}_i$) denote the i^{th} input. By learning a mapping between given input-output examples, a synthesized program learns to estimate the potential outcomes for unseen inputs. To this end, we propose the following program primitives $1 - 7$ (basic building blocks of a synthesized program), which are differentiable and encode specific causal inductive biases in an NN model. These primitives comprise our proposed DSL, shown in Tab 1.

1. The primitive "`if` $\alpha_1 > 0$ `then` α_2 `else` α_3" works similar to the equivalent programming construct. To avoid

discontinuities and enable backpropagation, following (Shah et al. 2020), we implement a smooth approximation i.e., $\texttt{if}\ \alpha_1 > 0\ \texttt{then}\ \alpha_2\ \texttt{else}\ \alpha_3$ can be written as $\texttt{sig}(\beta \cdot \alpha_1) \cdot \alpha_2 + (1 - \texttt{sig}(\beta \cdot \alpha_1)) \cdot \alpha_3$, where $\texttt{sig}(\cdot)$ is the *sigmoid* function and β is a temperature parameter. As $\beta \to 0$, the approximation approaches the usual $\texttt{if} - \texttt{then} - \texttt{else}$. Since we implement a smooth approximation of $\texttt{if} - \texttt{then} - \texttt{else}$, α_1 doesn't need to be a boolean value. $\alpha_1, \alpha_2, \alpha_3$ are real numbers.

2. The primitive "$\texttt{mlp}(\alpha)$" is a multi-layer perceptron that takes a vector α as input and returns a real number.
3. The primitive "$\texttt{shared mlp}(\alpha)$" is a multi-layer perceptron. All instances of $\texttt{shared mlp}(\alpha)$ in a synthesized program share the same set of parameters. It takes a vector α as input and returns a vector $\phi(\alpha)$ as output.
4. The primitive "$\texttt{subset}(\mathbf{v}, \texttt{S})$" selects a set of features of $\mathbf{v} \in \mathbb{R}^{n+1}$ indexed by the set $\texttt{S}$ of indices. Other features of $\mathbf{v}$ that are not indexed by the set S are set to 0. Hence, both the input and output are vectors from $\mathbb{R}^{n+1}$.
5. The primitive "$\texttt{propensity}(\mathbf{v})$" works similar to "$\texttt{shared mlp}$", taking $\mathbf{v} \in \mathbb{R}^{n+1}$ as input and returning $\phi(\mathbf{x}) \in \mathbb{R}^n$ as output. In addition, during training, for a given batch of inputs, it creates a loss value $\mathcal{L}_{prop}$ that computes the binary cross-entropy loss in predicting t from $\phi(\mathbf{x})$. $\mathcal{L}_{prop}$ is added to the overall loss in Eqn 5 if $\texttt{propensity}(\mathbf{v})$ is part of a synthesized program.
6. The primitive "$\texttt{align}(\mathbf{v})$" works similar to "$\texttt{shared mlp}$", taking $\mathbf{v} \in \mathbb{R}^{n+1}$ as input and returning $\phi(\mathbf{x}) \in \mathbb{R}^n$ as output. In addition, during training, for a given batch of inputs, it creates a loss value $\mathcal{L}_{align}$ that computes the Maximum Mean Discrepancy (MMD) between $p(\phi(\mathbf{x})|t = 1)$ and $p(\phi(\mathbf{x})|t = 0)$. $\mathcal{L}_{align}$ is added to the overall loss in Eqn 5 if $\texttt{align}(\mathbf{v})$ is part of a synthesized program.
7. "$\odot(\alpha_1, \alpha_2)$" where $\odot \in \{+, \times\}$ gives additional flexibility to the program synthesizer to combine other primitives. $\odot(\alpha_1, \alpha_2)$ takes two real numbers as inputs and returns a real number as output after performing the operation $\odot$.

Note that the primitives $1 - 7$ are combined when the corresponding input and output dimensions match. For example, α_1 in "$\texttt{if}\ \alpha_1 > 0\ \texttt{then}\ \alpha_2\ \texttt{else}\ \alpha_3$" can be $\texttt{mlp}(\alpha)$ but not $\texttt{shared mlp}(\alpha)$ as the output of $\texttt{mlp}(\alpha)$ is a real number matching the required dimension of α_1.

4.2 DSL Encodes Causal Inductive Biases

As discussed in § 2, existing learning-based causal effect estimation methods introduce inductive biases into machine learning models through regularizers or through changes in NN architectures. We now explain how the primitives of our DSL encode causal inductive biases introduced in popular causal effect estimation methods such as TARNet, CFR, Dragonnet, SNet, etc.

Connection to multi-head NNs: Recall that, from § 3, our goal is to estimate the quantity $\mathbb{E}[Y|T = t, \mathbf{X} = \mathbf{x}]$. If a single model is used to estimate both $\mathbb{E}[Y|T = 1, \mathbf{X} = \mathbf{x}]$ and $\mathbb{E}[Y|T = 0, \mathbf{X} = \mathbf{x}]$, it is often the case that $\mathbf{X}$ is high-dimensional and the treatment T is a relatively much smaller set of variables (often, just one variable) when compared to $\mathbf{X}$. Hence, T may not impact the model when making predictions, resulting in the estimated causal effect being biased towards zero (Künzel et al. 2019). Using two different models to estimate $\mathbb{E}[Y|T = 1, \mathbf{X} = \mathbf{x}]$ and $\mathbb{E}[Y|T = 0, \mathbf{X} = \mathbf{x}]$ suffers from high variance in estimating causal effect due to limited data in treatment-specific sub-groups as well as from selection bias.

To mitigate the aforementioned issues, (Shalit, Johansson, and Sontag 2017) propose an NN architecture in which two separate heads for treatment-specific outcomes are spawned from a shared representation layer. In NPS, to implement a shared representation layer, we can leverage the $\texttt{shared mlp}(\alpha)$ primitive as explained in Fig 1. To implement the two-heads spanning from latent representation, an NPS can leverage the $\texttt{if} - \texttt{then} - \texttt{else}$, $\texttt{mlp}$ and $\texttt{subset}$ primitives. That is, in "$\texttt{if}\ \alpha_1 > 0\ \texttt{then}\ \alpha_2\ \texttt{else}\ \alpha_3$", we can substitute "$\texttt{mlp}(\texttt{subset}(\mathbf{v}, \{0\}))$" in place of α_1 for conditional check and then accordingly execute α_2 or α_3 (a few points to recall at this juncture are: (i) $\texttt{mlp}$ returns a real number, (ii) t is at index 0 in $\mathbf{v}$, (iii) due to smooth approximation of $\texttt{if} - \texttt{then} - \texttt{else}$, α_1 doesn't need to evaluate to a boolean value). An NN with multiple heads would be implemented similarly using nested $\texttt{if} - \texttt{then} - \texttt{else}$ primitives.

Connection to propensity matching: Recall the adjustment formula: $\mathbb{E}[Y^t] = \mathbb{E}_{\mathbf{x} \sim \mathbf{X}}\left[\mathbb{E}[Y|T = t, \mathbf{X} = \mathbf{x}]\right]$ where the set of features $\mathbf{X}$ are controlled. Instead of controlling the entire set $\mathbf{X}$, which may be high dimensional, it is enough to control the propensity score $p(T|\mathbf{X})$ (Rosenbaum and Rubin 1983). To achieve propensity matching, we employ the primitive $\texttt{propensity}(\mathbf{v})$. Following (Shi, Blei, and Veitch 2019), for a given batch of data points, apart from producing a representation $\phi(\mathbf{x})$, $\texttt{propensity}(\mathbf{v})$ creates a loss value in predicting the treatment from the learned representation $\phi(\mathbf{x})$. On the other hand, if we know which subset of features from $\mathbf{X}$ to control, we can adjust only those subset of features. If we are unsure of the specific features to control, multiple instances of $\texttt{subset}(\mathbf{v}, \texttt{S})$ with different $\texttt{S}$ values can be used, allowing the NPS to select the appropriate subset to control for a given data.

Connection to randomized controlled trials: To improve the results from two-head NN models, CFR (Shalit, Johansson, and Sontag 2017) uses *IPM regularization* (using Maximum Mean Discrepancy (Gretton et al. 2012) or Wasserstein distance (Cuturi and Doucet 2014)) on a latent layer representation before spanning two treatment-specific heads. Minimizing the IPM between treatment and control subgroups: $p(\phi(\mathbf{X})|T = 1), p(\phi(\mathbf{X})|T = 0)$ ensures $T \perp\!\!\!\perp \phi(\mathbf{X})$ and thus mimicking randomized controlled trials (Shalit, Johansson, and Sontag 2017). The primitive $\texttt{align}(\mathbf{v})$ is intended to achieve a similar purpose in our DSL. That is, for a given batch of data points, apart from producing a representation $\phi(\mathbf{X})$, $\texttt{align}(\mathbf{v})$ creates a loss value equal to the MMD between $p(\phi(\mathbf{X})|t = 1)$ and $p(\phi(\mathbf{X})|t = 1)$.

We reiterate that we do not hard-code/pre-define the network architecture; the NPS learns to generate programs that compose primitives suitably to minimize overall loss during training. See Appendix § A, where we show how existing NN architectures CFR and Dragonnet are special cases of

NESTER. We now present the algorithm to synthesize neurosymbolic programs for the estimation of causal effects.

4.3 NESTER Algorithm

We refer to § 3 for the background on NPS, which we build on here. We use the A^* algorithm to implement NESTER. At any internal node u, A^* algorithm relies on a *heuristic value* $h(u)$ that underestimates the cost to reach the goal node from u. $h(u)$ decides which node to expand next. During program tree generation, non-terminals in an internal node u are substituted by a type-correct NN (e.g., if the primitive $\alpha_1 + \alpha_2$ returns a real number as output, the MLPs substituted for α_1, α_2 must also return real numbers as output). The training loss of the resultant program $(\mathcal{P}(u), \theta(u))$ on $\mathcal{D}$ then acts as the heuristic value $h(u)$ at node u (Shah et al. 2020). We run the A^* algorithm using this heuristic function to find programs that estimate causal effects. We outline our overall algorithm in Algorithm 1 of Appendix § B. Algorithm 1 returns the program that satisfies the objective in Eqn 5 (additional losses can be added to objective 5; see primitives $5, 6$ in § 4.1). Similar to traditional NN training, the parameters of the best program are chosen based on the cross-validation score. By keeping the overall program to only a limited depth allows us to build models that are efficiently learned and effective in practice with almost no additional time overhead compared to state-of-the-art methods.

5 NESTER: Analysis

We analyze NESTER from two perspectives: (i) the capability of a program synthesized using NPS methods to perform causal effect estimation and (ii) the capabilities of our proposed DSL in relation to well-known learning-based causal effect estimation methods. For the former, we hypothesize that NPS is a viable candidate for estimating causal effects if the relationship between treatment and effect is a continuous function. To this end, we first state the notion of an ϵ-admissible heuristic in Defn 5.1, show how a synthesized program's training loss can serve as such an ϵ-admissible heuristic in Lemma 5.1, and then state our result in Propn 5.1.

Definition 5.1. ***(ϵ-Admissible Heuristic (Harris 1974; Pearl 1984))*** *In an informed search algorithm, a heuristic function $h(u)$ that estimates the cost to reach the goal node g from a node u is said to be admissible if $h(u) \leq h^*(u), \forall u$ where $h^*(u)$ is the true cost to reach g from u. Given an $\epsilon > 0$, $h(u)$ is said to be $\epsilon-$admissible if $h(u) \leq h^*(u) + \epsilon, \forall u$.*

Lemma 5.1. ***(Neural Admissible Relaxations (Shah et al. 2020))*** *In an informed search algorithm $\mathcal{A}$, given an internal node u_i and a leaf node u_l, let the cost of the leaf edge (u_i, u_l) be $s(r)+\zeta(\mathcal{P}, \theta^*)$, where $\theta^* = \arg\min_\theta \zeta(\mathcal{P}, \theta)$ and $s(r)$ is the structural cost incurred by using rule r to create u_l from u_i. If an NN model $\mathcal{N}$ is used to substitute each non-terminal α of u_i such that $\mathcal{N}$ can approximate α up to an arbitrary precision δ for any given δ, the training loss of the program obtained is an $\epsilon-$admissible heuristic for u_i.*

Proposition 5.1. ***(Universal Approximation Result for NPS)*** *If the interventional effect of the treatment variable T on the target variable Y is a continuous function $g(T)$ i.e., $\mathbb{E}[Y|do(T)] = g(T)$, using any DSL $\mathcal{L}$ for synthesizing a single-hidden-layer neural network, NESTER synthesizes a program $(\mathcal{P}, \theta)$ that $\epsilon-$approximates g for a given $\epsilon > 0$.*

All proofs are in Appendix § A. Our proof follows from the universal approximation theorem for NN models (Hornik, Stinchcombe, and White 1989), a DSL for a single-hidden-layer NN and Lemma 5.1. The above result shows that if the relationship between treatment and effect is a continuous function, using NESTER is a viable candidate for estimating causal effects. We next discuss the capabilities of the proposed DSL w.r.t. existing methods.

Proposition 5.2. ***(Error Bounds of NESTER)*** *The program $(\mathcal{P}_C, \theta_C)$ generated by NESTER using the proposed DSL, whose architecture is the same as CFR, has the same error bounds in estimating causal effects as that of CFR.*

The above theoretical results show that the models for causal effect estimation generated by NESTER can be shown to have performance bounds similar to existing methods.

6 Experiments and Results

We perform a comprehensive suite of experiments to study the usefulness of NESTER in estimating causal effects with our proposed DSL. Our code and instructions to reproduce the results are included in the supplementary material and will be made publicly available.

Datasets: Evaluating causal effect estimation methods requires all potential outcomes to be available (Defn 3.2 and Defn 3.4), which is not possible due to *the fundamental problem of causal inference*. Thus, following (Shalit, Johansson, and Sontag 2017; Yoon, Jordon, and van der Schaar 2018; Shi, Blei, and Veitch 2019; Farajtabar et al. 2020), we experiment on two semi-synthetic datasets–Twins (Almond, Chay, and Lee 2005), IHDP (Hill 2011)–that are derived from real-world RCTs (see Appendix § C for details). For these two datasets, ground truth potential outcomes (a.k.a. counterfactual outcomes) are synthesized and available, and hence can be used to study the effectiveness of models in predicting potential outcomes. We also experiment on one real-world dataset–Jobs (LaLonde 1986)–where we observe only one potential outcome. We note that we are commensurate or better than existing work on the number of datasets studied. More details of datasets are provided in Appendix§ C.

Baselines: We compare NESTER with 16 baselines from various categories of methods as shown in Tab 2. We implement NESTER using NEAR (Shah et al. 2020) (NESTER-NEAR) that uses neural networks as relaxations of partial programs and dPads (Cui and Zhu 2021) (NESTER-dPads) that avoids combinatorial search over possible programs using a differentiable pruning strategy.

Evaluation Metrics: For the experiments on IHDP and Twins datasets where we have access to both potential outcomes, following (Shalit, Johansson, and Sontag 2017; Yoon, Jordon, and van der Schaar 2018; Shi, Blei, and Veitch 2019; Farajtabar et al. 2020), we use the evaluation metrics: ϵ_{ACE} and ϵ_{PEHE} (Defn 3.2 and Defn 3.4). ϵ_{ACE} measures the error in the estimation of the average causal effect in a population. ϵ_{PEHE} is operated on the error in the estimation of individual causal effects. For the experiments on the Jobs

Datasets (Metric) →	IHDP ($\epsilon_{ACE}(\downarrow)$)		Twins ($\epsilon_{ACE}(\downarrow)$)		Jobs ($\epsilon_{ACT}(\downarrow)$)	
Methods ↓	In-Sample	Out-of-Sample	In-Sample	Out-of-Sample	In-Sample	Out-of-Sample
OLS-1	.73±.04	.94±.05	.0038±.0025	.0069±.0056	**.01±.00**	.08±.04
OLS-2	.14±.01	.31±.02	.0039±.0025	.0070±.0059	**.01±.01**	.08±.03
k-NN	.14±.01	.90±.05	.0028±.0021	.0051±.0039	.21±.01	.13±.05
BLR	.72±.04	.93±.05	.0057±.0036	.0334±.0092	**.01±.01**	.08±.03
BART	.23±.01	.34±.02	.1206±.0236	.1265±.0234	.02±.00	.08±.03
Random Forest	.73±.05	.96±.06	.0049±.0034	.0080±.0051	.03±.01	.09±.04
Causal Forest	.18±.01	.40±.03	.0286±.0035	.0335±.0083	.03±.01	.07±.03
BNN	.37±.03	.42±.03	.0056±.0032	.0203±.0071	.04±.01	.09±.04
TARNet	.26±.01	.28±.01	.0108±.0017	.0151±.0018	.05±.02	.11±.04
MHNET	.14±.13	.37±.43	.0108±.0008	.0101±.0002	.04±.01	**.06±.02**
GANITE	.43±.05	.49±.05	.0058±.0017	.0089±.0075	**.01±.01**	**.06±.03**
CFR_{WASS}	.25±.01	.27±.01	.0112±.0016	.0284±.0032	.04±.01	.09±.03
Dragonnet	.16±.16	.29±.31	.0057±.0003	.0150±.0003	.04±.00	.07±.00
CMGP	.11±.10	.13±.12	.0124±.0051	.0143±.0116	.06±.06	.09±.07
TNet	.20±.18	.22±.11	.0200±.0070	.0200±.0070	.03±.01	.07±.08
SNet	.09±.10	.14±.12	.0040±.0030	.0040±.0030	.03±.03	.07±.07
NESTER - NEAR	**.05±.04**	**.05±.03**	**.0034±.0005**	.0039±.0006	**.01±.00**	.08±.08
NESTER - dPads	**.05±.01**	**.05±.02**	.0035±.0001	**.0028±.0001**	**.01±.00**	.08±.08

Table 2: Results on IHDP, Twins, and Jobs datasets. Lower is better. The best numbers are in bold. Simple machine learning models, ensemble models, and neural network-based models are separated using horizontal lines. OLS-1 refers to Ordinary Least Squares with treatment as a feature, OLS-2 refers to OLS with two regressors for two treatments. Further analysis on k-NN results and dataset details are in Appendix § C

dataset where we observe only one potential outcome per data point, following (Shalit, Johansson, and Sontag 2017; Yoon, Jordon, and van der Schaar 2018; Shi, Blei, and Veitch 2019; Farajtabar et al. 2020), we use the metric *error in the estimation of average causal effect on the treated* (ϵ_{ACT}). Definitions and more details of these metrics are provided in Appendix § C. Following (Shalit, Johansson, and Sontag 2017; Shi, Blei, and Veitch 2019; Yoon, Jordon, and van der Schaar 2018), We report both in-sample and out-of-sample performance w.r.t. $\epsilon_{ACE}, \epsilon_{ACT}, \sqrt{\epsilon_{PEHE}}$ in our results. Unlike traditional supervised learning, in-sample performance is non-trivial in this context, since we do not observe counterfactual outcomes (all potential outcomes) during training. Additional details on the experimental setup are presented in Appendix § C.

Results: The results shown in Tab 2 show the superior performance of NESTER over existing methods. To understand the results, we investigate the programs synthesized by NESTER. For example, the synthesized program for Jobs dataset is shown below where the synthesized program is similar to a two-head NN architecture with conditional check based on the propensity score.

$$\texttt{if } \texttt{mlp}_0(\texttt{propensity}(\mathbf{v})) > 0 \texttt{ then } \texttt{mlp}_1(\texttt{align}(\mathbf{v}))$$
$$\texttt{else } \texttt{mlp}_2(\texttt{align}(\mathbf{v}))$$

While the existing two head NN models perform conditional check based on treatment variable, the above program synthesized for Jobs dataset is also a valid program because it aligns with causal inductive bias that propensity score can be viewed as a direct parent of treatment variable (Rosenbaum and Rubin 1983; Shi, Blei, and Veitch 2019) and thus be valid for conditional check to decide on the specific NN head to execute. Program synthesis with appropriate primitives generates such valid programs that are not considered in literature. For Twins dataset, NESTER synthesized a simple program "$\texttt{mlp}(\mathbf{v})$" supporting the fact that simple programs such as linear regression performs better w.r.t. both in-sample and out-of-sample data (see Tab 2). For IHDP dataset, which is a collection of 1000 simulated datasets, we observe that the program below is synthesized more often.

$$\texttt{if } \texttt{mlp}_0(\texttt{subset}(\mathbf{v}, \{0, \ldots, |\mathbf{v}|\})) > 0$$
$$\texttt{then } \texttt{mlp}_1(\texttt{align}(\mathbf{v})) \texttt{ else } \texttt{mlp}_2(\texttt{align}(\mathbf{v}))$$

The program above is similar to a two head NN architecture where the conditional check is based on the entire feature set. To permit efficient learning (and to some degree, interpretability of the learned program, as discussed in Appendix §G), we limit the program depth to utmost 5 for the main experiments. In Appendix § D, we present results with other depths, results w.r.t. ϵ_{PEHE} metric, and the run time analysis of our method compared to baselines.

7 Limitations and Conclusions

We present an adaptive method for estimating causal effects using neurosymbolic program synthesis. We propose a DSL for causal effect estimation by comparing program primitives and model building blocks. The viability and suitability of this approach are theoretically demonstrated. Our approach is validated through extensive experimentation on benchmark datasets with multiple baselines, showcasing its usefulness. Enhancing the DSL with additional program primitives and addressing potential run-time issues induced thereof is a promising future direction. Relaxing the no latent confounding can be another interesting future work. This work has no known detrimental effects.

Acknowledgments

This work was partly supported by the Prime Minister's Research Fellowship (PMRF) program and an Adobe Research Gift. We thank Sai Srinivas Kancheti for involving in discussions and providing valuable suggestions. We are grateful to the anonymous reviewers for their valuable feedback, which improved the presentation of the paper.

References

A. Smith, J.; and E. Todd, P. 2005. Does matching overcome LaLonde's critique of nonexperimental estimators? *Journal of Econometrics*, 125(1-2): 305–353.

Abadie, A.; and Imbens, G. W. 2006. Large sample properties of matching estimators for average treatment effects. *Econometrica*, 74(1): 235–267.

Almond, D.; Chay, K. Y.; and Lee, D. S. 2005. The Costs of Low Birth Weight. *The Quarterly Journal of Economics*, 120(3): 1031–1083.

Assaad, S.; Zeng, S.; Tao, C.; Datta, S.; Mehta, N.; Henao, R.; Li, F.; and Carin Duke, L. 2021. Counterfactual Representation Learning with Balancing Weights. In *AISTATS*.

Bica, I.; Jordon, J.; and van der Schaar, M. 2020. Estimating the Effects of Continuous-valued Interventions using Generative Adversarial Networks. In *NeurIPS*.

Biermann, A. W. 1978. The Inference of Regular LISP Programs from Examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(8): 585–600.

Bošnjak, M.; Rocktäschel, T.; Naradowsky, J.; and Riedel, S. 2017. Programming with a Differentiable Forth Interpreter. In *ICML*.

Brady, H.; Collier, D.; and Sekhon, J. 2008. The Neyman-Rubin Model of Causal Inference and Estimation Via Matching Methods. *The Oxford Handbook of Political Methodology*.

Carey, T. A.; and Stiles, W. B. 2016. Some problems with randomized controlled trials and some viable alternatives. *Clinical Psychology & Psychotherapy*, 23(1): 87–95.

Chalmers, T. C.; Smith, H.; Blackburn, B.; Silverman, B.; Schroeder, B.; Reitman, D.; and Ambroz, A. 1981. A method for assessing the quality of a randomized control trial. *Controlled Clinical Trials*, 2(1): 31–49.

Chipman, H. A.; George, E. I.; and McCulloch, R. E. 2010. BART: Bayesian additive regression trees. *The Annals of Applied Statistics*, 4(1): 266 – 298.

Chu, Z.; Rathbun, S. L.; and Li, S. 2020. Matching in Selective and Balanced Representation Space for Treatment Effects Estimation. In *CIKM*.

Cinelli, C.; Forney, A.; and Pearl, J. 2022. A Crash Course in Good and Bad Controls. *Sociological Methods & Research*.

Collier, D.; and Mahoney, J. 1996. Insights and pitfalls: Selection bias in qualitative research. *World politics*, 49(1): 56–91.

CRUMP, R. K.; HOTZ, V. J.; IMBENS, G. W.; and MITNIK, O. A. 2009. Dealing with limited overlap in estimation of average treatment effects. *Biometrika*, 96(1): 187–199.

Cui, G.; and Zhu, H. 2021. Differentiable Synthesis of Program Architectures. In *NeurIPS*.

Curth, A.; and van der Schaar, M. 2021a. Nonparametric Estimation of Heterogeneous Treatment Effects: From Theory to Learning Algorithms. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR.

Curth, A.; and van der Schaar, M. 2021b. On Inductive Biases for Heterogeneous Treatment Effect Estimation. In *Advances in Neural Information Processing Systems*.

Cuturi, M.; and Doucet, A. 2014. Fast Computation of Wasserstein Barycenters. In *ICML*.

Diamond, A.; and Sekhon, J. S. 2013. Genetic matching for estimating causal effects: A general multi-variate matching method for achieving balance in observational studies. *Review of Economics and Statistics*, 95(3): 932–945.

Dorie, V. 2016. NPCI: Non-parametrics for Causal Inference.

Farajtabar, M.; Lee, A.; Feng, Y.; Gupta, V.; Dolan, P.; Chandran, H.; and Szummer, M. 2020. Balance Regularized Neural Network Models for Causal Effect Estimation. *CoRR*, abs/2011.11199.

Gaunt, A. L.; Brockschmidt, M.; Kushman, N.; and Tarlow, D. 2017. Differentiable Programs with Neural Libraries. In *ICML*.

Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A. C.; and Bengio, Y. 2014. Generative Adversarial Nets. In *NIPS*.

Gretton, A.; Borgwardt, K. M.; Rasch, M. J.; Schölkopf, B.; and Smola, A. 2012. A Kernel Two-Sample Test. *JMLR*, 13(25): 723–773.

Gulwani, S. 2011. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46(1): 317–330.

Gupta, S.; Childers, D.; and Lipton, Z. C. 2022. Local Causal Discovery for Estimating Causal Effects. In *NeurIPS 2022 Workshop on Causality for Real-world Impact*.

Harris, L. R. 1974. The heuristic search under conditions of error. *Artificial Intelligence*, 5(3): 217–234.

Hill, J. L. 2011. Bayesian Nonparametric Modeling for Causal Inference. *Journal of Computational and Graphical Statistics*, 20(1): 217–240.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2001. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1): 60–65.

Hornik, K.; Stinchcombe, M.; and White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5): 359–366.

Hoyer, P.; Janzing, D.; Mooij, J. M.; Peters, J.; and Schölkopf, B. 2008. Nonlinear causal discovery with additive noise models. In *Advances in Neural Information Processing Systems*.

Imbens, G. W. 2000. The role of the propensity score in estimating dose-response functions. *Biometrika*, 87(3): 706–710.

Kocsis, L.; and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293. Springer.

Künzel, S. R.; Sekhon, J. S.; Bickel, P. J.; and Yu, B. 2019. Metalearners for estimating heterogeneous treatment effects using machine learning. *Proceedings of the National Academy of Sciences*, 116(10): 4156–4165.

LaLonde, R. J. 1986. Evaluating the Econometric Evaluations of Training Programs with Experimental Data. *The American Economic Review*, 76(4): 604–620.

Lechner, M. 2001. Identification and estimation of causal effects of multiple treatments under the conditional independence assumption. In *Econometric evaluation of labour market policies*, 43–58. Springer.

Li, S.; and Fu, Y. 2017. Matching on Balanced Nonlinear Representations for Treatment Effects Estimation. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *NIPS*.

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; and Murphy, K. 2018. Progressive Neural Architecture Search. In *ECCV*.

Maathuis, M. H.; Colombo, D.; Kalisch, M.; and Bühlmann, P. 2010. Predicting causal effects in large-scale systems from observational data. *Nature methods*, 7(4): 247–248.

Mooij, J. M.; Peters, J.; Janzing, D.; Zscheischler, J.; and Schölkopf, B. 2016. Distinguishing cause from effect using observational data: methods and benchmarks. *The Journal of Machine Learning Research*, 17(1): 1103–1204.

Morgan, S. L.; and Winship, C. 2014. *Counterfactuals and Causal Inference: Methods and Principles for Social Research*. Analytical Methods for Social Research. Cambridge University Press, 2 edition.

Parisotto, E.; rahman Mohamed, A.; Singh, R.; Li, L.; Zhou, D.; and Kohli, P. 2017. Neuro-Symbolic Program Synthesis. In *ICLR*.

Pearl, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc.

Pearl, J. 2009. *Causality*. Cambridge university press.

Pearl, J.; Glymour, M.; and Jewell, N. 2016. *Causal Inference in Statistics: A Primer*. Wiley.

Pham, H.; Guan, M.; Zoph, B.; Le, Q.; and Dean, J. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *ICML*.

Rosenbaum, P. R.; and Rubin, D. B. 1983. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1): 41–55.

Rosenbaum, P. R.; and Rubin, D. B. 1985. Constructing a control group using multivariate matched sampling methods that incorporate the propensity score. *The American Statistician*, 39(1): 33–38.

Rubin, D. B. 1974. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of educational Psychology*, 66(5): 688.

Sanson-Fisher, R. W.; Bonevski, B.; Green, L. W.; and D'Este, C. 2007. Limitations of the Randomized Controlled Trial in Evaluating Population-Based Health Interventions. *American Journal of Preventive Medicine*, 33(2): 155–161.

Schwab, P.; Linhardt, L.; Bauer, S.; Buhmann, J. M.; and Karlen, W. 2020. Learning counterfactual representations for estimating individual dose-response curves. In *AAAI*.

Shah, A.; Zhan, E.; Sun, J.; Verma, A.; Yue, Y.; and Chaudhuri, S. 2020. Learning Differentiable Programs with Admissible Neural Heuristics. In *NeurIPS*.

Shalit, U.; Johansson, F. D.; and Sontag, D. 2017. Estimating individual treatment effect: generalization bounds and algorithms. In *ICML*.

Shi, C.; Blei, D.; and Veitch, V. 2019. Adapting Neural Networks for the Estimation of Treatment Effects. In *NeurIPS*.

Shi, K.; Dai, H.; Ellis, K.; and Sutton, C. 2022. CrossBeam: Learning to Search in Bottom-Up Program Synthesis. In *ICLR*.

Solar Lezama, A. 2008. *Program Synthesis By Sketching*. Ph.D. thesis, EECS Department, University of California, Berkeley.

Solar-Lezama, A.; Rabbah, R.; Bodík, R.; and Ebcioğlu, K. 2005. Programming by Sketching for Bit-Streaming Programs. *SIGPLAN Not.*, 40(6): 281–294.

Stuart, E. A. 2010. Matching methods for causal inference: A review and a look forward. *Statistical science: a review journal of the Institute of Mathematical Statistics*, 25(1): 1.

Tang, H.; and Ellis, K. 2022. From Perception to Programs: Regularize, Overparameterize, and Amortize. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, 30–39.

Valkov, L.; Chaudhari, D.; Srivastava, A.; Sutton, C.; and Chaudhuri, S. 2018. HOUDINI: Lifelong Learning as Program Synthesis. In *NeurIPS*.

Wager, S.; and Athey, S. 2018. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 113(523): 1228–1242.

Winskel, G. 1993. *The Formal Semantics of Programming Languages: An Introduction*. Cambridge, MA, USA: MIT Press.

Yao, L.; Li, S.; Li, Y.; Huai, M.; Gao, J.; and Zhang, A. 2018. Representation Learning for Treatment Effect Estimation from Observational Data. In *NeurIPS*.

Yoon, J.; Jordon, J.; and van der Schaar, M. 2018. GANITE: Estimation of Individualized Treatment Effects using Generative Adversarial Nets. In *ICLR*.

Zhang, W.; Liu, L.; and Li, J. 2021. Treatment Effect Estimation with Disentangled Latent Factors. In *AAAI*.

Zoph, B.; and Le, Q. 2017. Neural Architecture Search with Reinforcement Learning. In *ICLR*.