ESTIMATING TREATMENT EFFECTS USING NEUROSYMBOLIC PROGRAM SYNTHESIS

Abbavaram Gowtham Reddy & Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology Hyderabad, India
{cs19resch11002,vineethnb}@iith.ac.in

ABSTRACT

Estimating treatment effects from observational data is a central problem in causal inference. Methods to solve this problem exploit inductive biases and heuristics from causal inference to design multi-head neural network architectures and regularizers. In this work, we propose to use neurosymbolic program synthesis, a data-efficient, and interpretable technique, to solve the treatment effect estimation problem. We theoretically show that neurosymbolic programming can solve the treatment effect estimation problem. By designing a Domain Specific Language (DSL) for treatment effect estimation problem based on the inductive biases used in literature, we argue that neurosymbolic programming is a better alternative to treatment effect estimation than traditional methods. Our empirical study reveals that our method, which implicitly encodes inductive biases in a DSL, achieves better performance on benchmark datasets than the state-of-the-art methods.

1 INTRODUCTION

Treatment effect (also referred to as causal effect) estimation estimates the effect of a treatment variable on an outcome variable (e.g., the effect of a drug on recovery). Randomized Controlled Trials (RCTs) are widely considered as the gold standard approach for treatment effect estimation (Chalmers et al., 1981; Pearl, 2009). In RCTs, individuals are randomly split into the treated group and the control (untreated) group. This random split removes the spurious correlation between treatment and outcome variables before the experiment so that estimated treatment effect is unbiased. However, RCTs are often: (i) unethical (e.g., in a study to find the 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 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 & Stiles, 2016; Pearl et al., 2016). These limitations leave us with observational data to compute treatment effects.

Observational data, similar to RCTs, suffers from the fundamental problem of causal inference (Pearl, 2009), which states that for any individual, we cannot observe all potential outcomes at the same time (e.g., once we record a person’s medical condition after taking a medicinal drug, we cannot observe the same person’s medical condition with 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 & Mahoney, 1996). For these reasons, estimating unbiased treatment effects from observational data can be challenging (Hernan & Robins, 2019; Farajtabar et al., 2020). However, due to the many use cases in the real-world, estimating treatment effects from observational data is one of the long-standing central problems in causal inference (Rosenbaum & Rubin, 1983; 1985; Brady et al., 2008; Morgan & Winship, 2014; Shalit et al., 2017; Yoon et al., 2018; Shi et al., 2019; Yao et al., 2018; Zhang et al., 2021).

Earlier methods that estimate treatment effects from observational data are based on matching techniques that compare data points from treatment and control groups that are similar w.r.t. a metric (e.g., Euclidean distance in nearest-neighbor matching, or propensity score matching) (Brady et al., 2008; Morgan & Winship, 2014). Recent methods exploit inductive biases and heuristics from causal inference to design multi-head neural network (NN) models and regularizers (Hill, 2011; Farajtabar et al., 2020; Shi et al., 2019; Schwab et al., 2020; Chu et al., 2020; Shalit et al., 2017; Alaa & van der Schaar, 2017; Yoon et al., 2018; Bica, 2021).
Multi-head NN models are typically used when treatment variables are single-dimensional and categorical (Shi et al., 2019; Shalit et al., 2017; Farajtabar et al., 2020; Schwab et al., 2020), and regularizers therein enforce constraints such as controlling for propensity score instead of pre-treatment covariates, i.e. covariates that are not caused by the treatment variable in the underlying causal data generating graph (Shi et al., 2019; Rosenbaum & Rubin, 1983). However, each such model is well-suited to a certain kind of causal graph, and may not apply to all causal data generating processes. For example, as shown in Fig 1, CFRNet (Shalit et al., 2017), a popular NN-based treatment estimation model, controls pre-treatment covariates using a regularizer based on an Integral Probability Metric (IPM). It requires the representations of non-treatment covariates (denoted as $x$ in Fig 1) with and without treatment to be similar. This is relevant for causal model A in the figure, but does not work for causal model B, where non-treatment covariates are caused by the treatment $t$ and hence could vary for different values of $t$. One would ideally need a different regularizer or architecture to address causal model B (the same observation holds for TARNet (Shalit et al., 2017) too). In practice, one may not be aware of the underlying causal model, making this more challenging. In this work, we instead propose to use a neurosymbolic program synthesis technique to compute treatment effect, which does not require such explicit regularizers or architecture redesign for each causal model. Such a technique learns to automatically synthesize differentiable programs that satisfy a given set of input-output examples (Shah et al., 2020; Parisotto et al., 2016), and can hence learn the sequence of operations to estimate treatment effect for this set. We call our method as the NEuroSYmbolic Treatment Effect EstimatoR (or NESTER). Neurosymbolic program synthesis is known to have the flexibility to synthesize different programs for different data distributions to optimize a performance criterion, while still abiding by the inductive biases studied in treatment effect estimation literature (see Sec 4.1 for more details). To describe further, one could view CFRNet/TARNet as implementing one $\text{if } - \text{then } - \text{else}$ program primitive with its two-headed NN architecture. NESTER will instead automatically synthesize the sequence of program primitives (from a domain-specific language of primitives) for a given set of observations from a causal model, and can thus generalize to different distributions.

Program synthesis methods, in general, enumerate a set of programs and select (from the enumeration) a set of feasible programs that satisfy given input-output examples so that the synthesized programs generalize well to unseen inputs (see Appendix for an example) (Biermann, 1978; Gulwani, 2011; Parisotto et al., 2016; Valkov et al., 2018; Shah et al., 2020). Usually, a Domain-Specific Language (DSL) (e.g., a specific context-free grammar) is used to synthesize relevant programs for a given domain and task. Recently, various NN-based techniques have been proposed to perform neurosymbolic program synthesis (Parisotto et al., 2016; Valkov et al., 2018; Gaunt et al., 2017; Bosnjak et al., 2017). We use the neurosymbolic program synthesis paradigm where each program primitive (e.g., $\text{if } - \text{then } - \text{else}$, $\alpha_1 + \alpha_2$) is a differentiable module (Parisotto et al., 2016; Shah et al., 2020). Such differentiable programs simultaneously optimize program primitive parameters while learning the overall program structure and flow. The set of possible programs that can be synthesized using a DSL is often large (Parisotto et al., 2016). Many methods have been proposed to search through the vast search space of programs efficiently (Gulwani et al., 2012; Parisotto et al., 2016; Valkov et al., 2018; Shah et al., 2020). We use Neural Admissible Relaxation (Shah et al., 2020) in this work, which uses neural networks as relaxations of partial programs while searching the program space using informed search algorithms such as $A^*$ (Hart et al., 1968). The final program can be obtained by training using gradient descent algorithms. Our key contributions are:

- We study the use of neurosymbolic program synthesis as a practical approach for solving treatment effect estimation problems. To the best of our knowledge, this is the first such effort that applies neurosymbolic program synthesis to estimate treatment effects.
• We propose a Domain-Specific Language (DSL) for treatment effect estimation, where each program primitive is motivated from basic building blocks of models for treatment effect estimation in literature.
• We theoretically show that our neurosymbolic program synthesis approach can approximate a continuous function up to an arbitrary precision. This result enables us to solve the treatment effect estimation problem by assuming a continuous function relating the treatment and outcome variables.
• We perform comprehensive empirical studies on multiple benchmark datasets (including additional results in the Appendix) where we outperform existing state-of-the-art models. We also show the interpretability of such a neurosymbolic approach on synthetic as well as real-world datasets, thus highlighting the usefulness of our approach over traditional treatment effect estimation methods.

2 RELATED WORK

Traditional Methods for Treatment Effect Estimation. Early methods of treatment effect estimation from observational data are largely based on matching techniques (Brady et al., 2008; Morgan & Winship, 2014; Stuart, 2010) where similar data points in treatment and control groups are compared using methods such as nearest neighbor matching and propensity score matching to estimate treatment effects. In nearest neighbor matching (Stuart, 2010), for each data point in the treatment group, the nearest points from the control group w.r.t. Euclidean distance are identified, and the difference in potential outcomes between treatment and corresponding control data points is estimated as the treatment effect. In propensity score matching (Rosenbaum & Rubin, 1983), a model is trained to predict the treatment effect value using all data points from both treatment and control groups. Using this model, points from treatment and control groups that are close w.r.t. the model’s output are compared, and the difference in potential outcomes of these points is estimated as treatment effect. However, such matching techniques are known to not scale to high-dimensional or large-scale data (Abadie & Imbens, 2006).

Another family of methods estimates treatment effects using the idea of backdoor adjustment (Pearl, 2009; Rubin, 2005). Under the assumption of availability of a sufficient adjustment set (Pearl, 2009), these models rely on fitting conditional probabilities given the treatment variable and a sufficient adjustment set of covariates. However, such models are known to suffer from high variance in the estimated treatment effects (Shalit et al., 2017). Covariate balancing is another technique to control for the confounding bias to estimate treatment effects. Weighting techniques perform covariate balancing by assigning weights to each instance based on various techniques (e.g., weighting each instance using propensity score in the inverse probability weighting technique) (Rosenbaum & Rubin, 1983; Assaad et al., 2021; CRUMP et al., 2009; Li & L, 2013; Diamond & Sekhon, 2013; Li & 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 delivered significantly better performance in recent years.

Learning-based Methods for Treatment Effect Estimation. Recent methods to estimate treatment effects have largely been based on multi-headed NN models equipped with regularizers (Hill, 2011; Farajtabar et al., 2020; Shi et al., 2019; Schwab et al., 2020; Chu et al., 2020; Shalit et al., 2017; Yoon et al., 2018; Bica et al., 2020). To find treatment effects under 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. CFRNet (Shalit et al., 2017) proposed a two-headed NN architecture with a regularizer that forced representations of treatment and control groups to be close to each other, in order to adjust for confounding features before forwarding the representation to treatment-specific heads. Extending CFRNet architecture, (Farajtabar et al., 2020) proposed an additional regularizer to adjust for confounding by forcing both treatment-specific heads to have same baseline outcomes (i.e., for any data point, both treatment-specific heads should output same value). In Dragonnet (Shi et al., 2019), along with two heads for predicting treatment-specific (potential) outcomes, an additional head to predict treatment value was also used; this allowed pre-treatment covariates to be used in predicting potential outcomes. Assuming that potential outcomes are strongly related, (Curth & van der Schaar, 2021) proposed techniques that improve existing models using the structural similarities between potential outcomes. All of these methods, however, have a fixed architecture design and can hence address observational data from certain causal models. Our approach is also NN-based but uses a neurosymbolic approach to automatically synthesize...
an architecture (or a flow of program primitives), thereby providing it a capability to work across observational data from different causal models conveniently.

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) have also been used to learn the interventional distribution (Yoon et al., 2018; Bica et al., 2020) from observed data in both categorical and continuous treatment variable settings to estimate treatment effects. By disentangling confounding variables from instrumental variables, (Zhang et al., 2021) proposed a variational inference method for treatment effect estimation that uses only confounding variables. However, generative modeling requires a large amount of data to be useful, which is often not practical in treatment effect estimation tasks. (Yao et al., 2018) proposed a method to learn representations by leveraging local similarities and thereby estimate treatment effect. Ensemble models such as causal forests (Wager & Athey, 2018), and Bayesian additive regression trees (Chipman et al., 2010) have also been considered for interval estimation. As stated earlier, our work is however very different from these efforts, and seeks to build a flexible yet powerful framework for treatment effect estimation using neurosymbolic program synthesis.

Neurosymbolic Program Synthesis. Program synthesis, viz. automatically learning a program that satisfies a given set of input-output examples (Biermann, 1978; Gulwani, 2011; Parissetto et al., 2016; Valkov et al., 2018; Shah et al., 2020), has been shown to be 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, evolutionary algorithms, and recently, node pruning with neural admissible relaxation have been proposed to efficiently search for optimal programs from a vast search space (Gulwani et al., 2012; Parissetto et al., 2016; Valkov et al., 2018; Shah et al., 2020). We use the idea of node pruning with neural admissible relaxation (Shah et al., 2020) in this work as it gives near-optimal solutions with fast convergence. This is the first use of neurosymbolic program synthesis for treatment effect estimation, to the best of our knowledge.

### 3 Background and Problem Formulation

#### Treatment Effect Estimation

Let \( D = \{ (x_i, t_i, y_i) \}_{i=1}^{n} \) be an observational dataset of \( n \) triplets. Each triplet \((x_i, t_i, y_i)\) is a sample drawn from the true data distribution \( p(X, T, Y) \), where \( X \), \( Y \) and \( T \) are the corresponding random variables (described herein). \( x_i \in \mathbb{R}^d \) denotes the \( d \)-dimensional covariate vector, \( t_i \in \mathbb{R} \) denotes the treatment value \( t_i \) is not a part of \( x_i \), and \( y_i \in \mathbb{R} \) denotes the corresponding outcome. To explain treatment effects, consider a simple setting where treatment is binary-valued i.e., \( t \in \{ 0, 1 \} \). 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 \( t_i; x_i \). The observed outcome \( y_i \) can be expressed in terms of true potential outcomes as: \( y_i = t_i Y^{i,t_i} + (1 - t_i) Y^{i,1-t_i} \).

One of the goals in treatment effect estimation from observational data is to learn the estimator \( f(x, t) \) such that the difference between estimated potential outcomes (i.e., under \( t = 1 \) and \( t = 0 \)): \( f(x, 1) - f(x, 0) \) is as close as possible to the difference in true potential outcomes: \( Y^{1} - Y^{0} \) \( \forall i \). This difference for a specific individual \( i \) is known as Individual Treatment Effect (ITE) (Pearl, 2009).

Extending the discussion on ITE to an entire population, our goal is to estimate the Average Treatment Effect (ATE) of the treatment variable \( T \) on the outcome variable \( Y \) which is defined as:

\[
ATE_Y^T = \mathbb{E}[Y|do(T=1)] - \mathbb{E}[Y|do(T=0)]
\]

where the \( do(.) \) notation denotes external intervention to the treatment variable (Pearl, 2009), i.e. \( \mathbb{E}[Y|do(T = t)] \) refers to the expected value of the outcome \( Y \) when every individual in the population is administered with the treatment \( t \). (Note that if treatment is not binary-valued, treatment effects are calculated w.r.t. a baseline treatment value (Pearl, 2009), and the right term in Eqn 1 would compute the interventional expectation at the baseline.) Assuming \( X \) satisfies the backdoor criterion relative to the treatment effect of \( T \) on \( Y \) (Pearl, 2009), we can write \( \mathbb{E}[Y|do(T = t)] = \mathbb{E}_{x\sim X}[\mathbb{E}[Y|T = t, X = x]] \). Using this, a simple technique to estimate \( \mathbb{E}[Y|T = t, X = x] \) (and thus \( \mathbb{E}[Y|do(T = t)] \)), the ATE is to fit a model for \( Y \) given \( T, X \). These models are the basic building blocks of most methods for treatment effect estimation. We use the finite sample approximation of ATE by taking the average of ITEs. Following (Shalit et al., 2017; Lechner, 2001; Imbens, 2020; Schwab et al., 2020; Zhang et al., 2021), we make the following assumptions which are sufficient to guarantee the identifiability (Pearl, 2009) of treatment effects from observational data.
• **Ignorability:** This is also referred to as no unmeasured confounding assumption. For a given set of pre-treatment covariates, treatment is randomly assigned. Mathematically, in a binary treatment setting, conditioned on a set of pre-treatment covariates \( X \), treatment \( T \) is independent of the outcomes \( Y_0, Y_1 \) (i.e., \( Y_0, Y_1 \perp T \mid X \)).

• **Positivity:** Treatment assignment for each individual is not deterministic, and it must be possible to assign all treatment values to each individual, i.e. \( 0 < p(t \mid x) < 1 \ \forall t, x \).

• **Stable Unit Treatment Value Assumption (SUTVA):** The observed outcome of any individual under treatment must be independent of the treatment assignment to other individuals.

**Neurosymbolic Program Synthesis:** Following [Shah et al., 2020], let \( (P, \theta) \) be a neurosymbolic program where \( P \) denotes the program structure and \( \theta \) denotes the program parameters. \((P, \theta)\) is differentiable in \( \theta \) (see Appendix for an example neurosymbolic program). \( P \) is synthesized using a Context-Free Grammar (CFG). A CFG consists of a set of rules of the form \( \alpha \rightarrow \sigma_1, \ldots, \sigma_n \) where \( \alpha \) is a non-terminal and \( \sigma_1, \ldots, \sigma_n \) are either non-terminals or terminals. Program synthesis starts with an initial non-terminal, then iteratively applies the CFG 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 are considered as nodes in a program graph. The process continues until no non-terminals are left, i.e., we have synthesized a program. The leaf nodes of the resultant program graph contain structures that consist of only terminals. Let \( \theta \) be the set of parameters of such a leaf node structure \( P \). Let \( s(r) \) be the cost incurred in using the rule \( r \) while generating a program structure. The structural cost of \( P \) is \( s(P) = \sum_{r \in R(P)} s(r), \) where \( R(P) \) is the set of rules used to create the structure \( P \). In this paper, we set \( s(r) \) to a constant real number for all production rules (e.g., \( s(r) = 1 \ \forall r \in R(P) \)).

The program learning problem is thus usually formulated as a graph search problem, i.e., starting with an empty graph, the graph is expanded by creating new partial structures (internal nodes of the graph) and structures (leaf nodes of the graph). When searching for an optimal program, parameters of the program (and program structures) are updated simultaneously along with the synthesis of the programs (Shah et al., 2020).

For a synthesized program \( (P, \theta) \), we define \( \zeta(P, \theta) = E_{(x, t, y) \sim D}[((P, \theta)(x, t) - y)^2] \) as the error/cost incurred by \( (P, \theta) \) in estimating treatment effects. The overall goal of neurosymbolic program synthesis is then to find a structurally simple program (that can also be human-interpretable) with low prediction error, i.e. to solve the optimization problem: \( (P^*, \theta^*) = \arg \min_{(P, \theta)} (s(P) + \zeta(P, \theta)) \). We now describe our methodology.

### 4 Neurosymbolic Treatment Effect Estimator: Methodology

The overall idea of our methodology is to design a Domain-Specific Language (DSL) for treatment effect estimation that is fairly general, followed by the use of the standard A* search algorithm to synthesize programs given observational data from a specific causal model. We begin by discussing the DSL we design, followed by the program synthesizer. Note that one could view each primitive of our DSL as modules of existing learning-based treatment effect estimators such as TARNet or CFRNet (Shalit et al., 2017). We also theoretically analyze the usefulness of the search-based neurosymbolic program synthesizer for the given task.

#### 4.1 Domain Specific Languages for Treatment Effect Estimation

Since a program synthesizer requires as input a set of input-output examples, unsurprisingly, we can pose the problem of treatment effect estimation as the problem of mapping a set of inputs to corresponding outputs. Concretely, given observational data \( D \), the set of pairs \( \{(t_i, x_i)\}_{i=1}^n \) act as inputs and the set of outcomes \( \{y_i\}_{i=1}^n \) act as outputs. For simplicity, let \( v_i = [t_i; x_i] \) (concatenation of treatment and covariates) denote the \( i \)th input. A synthesized program learns to estimate the potential outcomes for unseen inputs by learning a mapping between given input-output examples. To bring interpretability to synthesized programs and to leverage the inductive biases...
considered in treatment effect estimation literature, we develop a DSL (Table 2) based on well-known program primitives that have connections to ideas used in literature for treatment effect estimation (illustrated in Table 1). (We later state Prop 4.2 that guarantees the existence of a DSL for treatment effect estimation task). As discussed earlier, existing treatment effect estimation methods introduce inductive biases into machine learning models either through regularizers or through changes in NN architectures. On a similar note, one could view a DSL as a set of inductive biases based on learnable program primitives (Shah et al., 2020; Chaudhuri et al., 2021). The proposed DSL is based on inductive biases used in treatment effect estimation literature. We next describe the connections between program primitives in our DSL in Table 2 and inductive biases used in traditional treatment effect estimation methods.

Connection between multi-head neural network architectures and if−then−else, subset: Recall that, in treatment effect estimation, our goal is to estimate the quantity $E[Y|T=t, X=x]$. If a single model is used to estimate both $E[Y|T=0, X=x]$ and $E[Y|T=1, X=x]$, it is often the case that $X$ is very high dimensional and hence the treatment $T$, which is often one dimensional, may be discarded by the model when making predictions. This will result in the estimated treatment effect being biased towards 0 (Künzel et al., 2019). To account for this, two separate models can be used to estimate $E[Y|T=0, X=x]$ and $E[Y|T=1, X=x]$. However, this method suffers from high variance in estimated treatment effect due to limited data in treatment-specific sub-groups and selection bias (Shah et al., 2017).

In order to mitigate this problem, (Shalit et al., 2017) and subsequent efforts by (Shi et al., 2019; Schwab et al., 2020; Farajtabar et al., 2020) use an NN architecture in which two separate heads are spanned from a latent representation layer to predict treatment specific outcomes and thus achieve better treatment effect estimate with lower variance. Such two-head NNs can be implemented using a combination of if−then−else and subset program primitives. For example, to implement a two-head NN architecture, a neurosymbolic program synthesizer can perform the following: if $\alpha_1 = \text{subset}(v, [0.1]) = 1$, subset([v, [a, b]) takes a vector $v$ as input and returns a real number as output (as explained later in this section), the program synthesizer executes $\alpha_2$ else it executes $\alpha_3$ where $\alpha_2, \alpha_3$ are two different sub-structures that act as two heads of the overall architecture.

Note that each $\alpha$ in the primitive: “if $\alpha$ then $\alpha$ else $\alpha$” returns a real number and hence the output of “if $\alpha$ then $\alpha$ else $\alpha$” is also a real number. For e.g., as discussed above, $\alpha$ here can be either subset([v, [0.1]) or transform([v, $\mu, \sigma$]) too. Here both subset([v, [0.1]) and transform([v, $\mu, \sigma$]) take a vector $v$ as input and return a real number as output.

To avoid discontinuities and to enable backpropagation, following (Shah et al., 2020), we implement a smooth approximation of if−then−else. For example, smooth approximation of if $a > 0$ then $b$ else $c$ can be written as $\sigma(\beta \cdot a) \cdot b + (1 - \sigma(\beta \cdot a)) \cdot c$, where $\sigma$ is the sigmoid function and $\beta$ is a temperature parameter. As $\beta \rightarrow 0$, the approximation approaches the usual if−then−else. It is now easy to see that multi-head NN architectures can be implemented using multiple if−then−else and subset primitives. It is important to note that we do not hard-code/pre-define the network architecture. Instead, the program synthesizer learns to generate programs such that the primitives are composed in any order it deems to be effective in minimizing the loss value during training (see Sec. 5 and Appendix for examples).

Connection between IPM regularization and transform: To improve the results from two head NN architectures (e.g., TARNNet), CFRNet (Shalit et al., 2017) proposes to use IPM regularization (e.g., Maximum Mean Discrepancy (Gretton et al., 2012), Wasserstein distance (Cuturi & Doucet, 2014)) on a latent layer representation. This enforces the encoded distribution of treatment $(p(x|t=1))$ and control $(p(x|t=0))$ groups to be close to each other. Minimizing IPM between $p(\phi(x)|t=1)$ and $p(\phi(x)|t=0)$ (where $\phi$ is the learned representation) is then the same as ensuring that treatment and covariates are independent (i.e., $T \perp \perp X$) thus mimicking RCTs (Shalit et al., 2017). To introduce this kind of inductive bias, we introduce a program primitive called transform($\alpha, \mu, \sigma$) that transforms a given input vector $\alpha$ into $\phi(\alpha)$ using two other vectors $\mu, \sigma$ as $\phi(\alpha) = \frac{\alpha - \mu}{\sigma}$, where $\mu, \sigma$ are mean and standard deviations of the observational data $D$.

If transform($\alpha, \mu, \sigma$) is applied to the entire dataset $D$, the transformed data now has mean 0 and standard deviation 1 (where 0 and 1 are vectors of 0s and 1s respectively). When the two subpopulations $p(x|t=0)$ and $p(x|t=1)$ are distributed similarly to $D$ (which would also satisfy the ignorability assumption), the means and standard deviations of $p(\phi(x)|t=0)$ and $p(\phi(x)|t=1)$ will also be approximately equal to 0 and 1 respectively (note that $\phi(x) = \frac{x-\mu}{\sigma}$, and the input to
A DSL for the Treatment Effect Estimation Task

\[ a := \text{if } a \text{ then } a \text{ else } a \mid \text{transform}(a, \mu, \sigma) \mid \text{subset}(a, [a..b]) \mid \text{const}(S) \mid \text{⊙} (a, \alpha) \mid v \]

| Program Primitive | Description |
|-------------------|-------------|
| 1. if \( a \) then \( a \) else \( a \) | Simple if – then – else condition. To avoid evaluating conditions, and to enable back-propagation, we implement smooth approximation of if – then – else. |
| 2. transform\( (a, \mu, \sigma) \) | Transform the input vector \( a \) into \( \phi(a) = \frac{a - \mu}{\sigma} \) where \( \mu \) and \( \sigma \) are mean and standard deviation of observational data. Feed \( \phi(a) \) into a NN to get a real number as output. |
| 3. subset\( (a, [a..b]) \) | Select a set of features from the start index \( a \) (including) to end index \( b \) (excluding) from the input \( a \). Other features are set to 0. Feed this vector into a NN to get a real number as output. |
| 4. const\( (S) \) | Learn a set of constants of shape \( S \). |
| 5. \( \text{⊙} (a, \alpha) \) | Parameterized algebraic functions (e.g., \( \alpha_1 + \alpha_2 \alpha_3 \alpha_4; \alpha_1, \alpha_2 \in \mathbb{R} \)). |

Table 2: A DSL for the treatment effect estimation task in Backus-Naur form [Winik 1993] and its semantics. \( v \) represents input from \( D \). More details of each primitives is provided below.

\text{transform}(a, \mu, \sigma) \) is the vector \([t, x] \in D \). Then, the Maximum Mean Discrepancy between \( p(\phi(x)|t = 0) \) and \( p(\phi(x)|t = 1) \) will go towards zero when matching the first two moments of \( p(\phi(x)|t = 0) \) and \( p(\phi(x)|t = 1) \) for treatment effect estimation. The transformed vector \( \phi(x) \) is subsequently fed into a multi-layer perceptron to produce a real number as output, which is then fed into other program primitives.

\text{transform}(a, \mu, \sigma) \) appears to be similar to data standardization, a data pre-processing step. However, unlike the fixed architecture in traditional NNs, program synthesis has the flexibility to choose when to use \( \text{transform}(a, \mu, \sigma) \) (for e.g, \( x \) could be the output of the 3rd program primitive, not input). Besides, though the reason to introduce \( \text{transform}(a, \mu, \sigma) \) is to mimic IPM regularization (specifically Maximum Mean Discrepancy), it is evident from our DSL that the program synthesizer can use \( \text{transform}(a, \mu, \sigma) \) multiple times in a program (see Appendix for examples).

\text{Connection between pre-treatment covariate selection and subset:} Under the ignorability assumption, pre-treatment covariates are controlled to find estimates of treatment effects. For e.g., (Shi et al. [2019]) controls pre-treatment covariates via controlling propensity score [Rosenbaum & Rubin 1983]. However, it is not required to control all the covariates in the input. In order to identify a minimal set of pre-treatment covariates to control, we use \( \text{subset}(a, [a..b]) \) primitive. If we do not know which indices to select, multiple instances of \( \text{subset}(a, [a..b]) \) can be used by assigning different values to \( a, b \) in each instance. Program synthesizer then selects appropriate \( \text{subset}(a, [a..b]) \) for some \( a, b \). \( \text{subset}(a, [a..b]) \) also helps to identify the most important dimensions from a given input/hiden representation as explained in the if – then – else example earlier. Finally, the chosen vector is fed into a multi-layer perceptron to produce a real number as output which will subsequently be used by other primitives.

The other two simple program primitives – \( \text{const}(S) \), \( \text{⊙} (a, \alpha) \) – whose semantics are given in Table 2 are included for giving additional flexibility to the program synthesizer and combining various program primitives effectively to achieve better results. \( \text{⊙} (a, \alpha) \) takes two real numbers as inputs and returns a real number as output after performing the algebraic operation \( \text{⊙} \). Using the proposed DSL, we now present the algorithm to synthesize neurosymbolic programs that estimate treatment effects.

4.2 Neurosymbolic Program Synthesis for Treatment Effect Estimation

We use the \( A^* \) informed search algorithm [Hart et al. 1968] to implement the proposed NESTER method. The heuristic function \( h \) we use in our method is defined as follows. For any partial structure \( \mathcal{P}(u) \) in a node \( u \), NNs with adequate capacity (enough width and depth) are used to replace the non-terminals. The training loss of the resultant program \( \mathcal{P}(u, \theta(u)) \) on \( D \) then acts as the heuristic value \( h(u) \) at the node \( u \) [Shah et al. 2020]. Using this heuristic function, we run \( A^* \) algorithm to find the programs that estimate treatment effects. We outline our approach in Algorithm 1. We now study the theoretical guarantees of neurosymbolic program synthesis in estimating treatment effects.

**Definition 4.1. (Admissible Heuristics [Harris 1974, Pearl 1984])** In an informed search algorithm, a heuristic function \( h(u) \) that estimates the cost to reach goal node from a node \( u \) is said to be admissible if \( h(u) \leq h^*(u), \forall u \) where \( h^*(u) \) is the actual/true cost to reach the goal node from \( u \). \( h(u) \) is said to be \( \epsilon \)-admissible if \( h(u) \leq h^*(u) + \epsilon, \forall u \).
Proposition 4.1. In an informed search algorithm, let the cost of the leaf edge \((u_i, u_j)\) (edge connecting internal node \(u_i\) to leaf node \(u_j\)) be \(s(v) + \zeta(P, \theta^*)\), where \(\theta^* = \arg\min_{\theta \in \Theta} \zeta(P, \theta)\) and \(r\) is the rule used to create \(u_i\) from \(u_j\). If NNs \(N\) parameterized by their capacity (architecture width and height) are used to substitute the non-terminals in the partial structure of \(u_i\), the resultant program's training loss is equal to the \(\epsilon\)-admissible heuristic value at the node \(u_i\). Such an \(\epsilon\)-admissible heuristic returns a solution whose path cost is at most an additive constant \(\epsilon\) away from the path cost of the optimal solution (Shah et al., 2020).

Proposition 4.2. Given an \(\epsilon\)-admissible heuristic, for any trained 1-hidden layer NN \(N\) with \(m\) inputs, \(n\) hidden neurons, and one output, there exist a Domain Specific Language \(L\) such that the error/loss incurred by the synthesized program \((P, \theta)\) is \(\epsilon\)-close to the error/loss incurred by \(N\) in approximating any continuous function.

Proofs of the above propositions are in the Appendix. The universal approximation theorem (Hornik et al., 1989) states that we can increase the number of hidden layer neurons of a 1-hidden layer NN \(\hat{N}\) to approximate any continuous function \(f\) with a certain error, say \(\epsilon\). Proposition 4.2 states that there exists a neurosymbolic program \((P, \theta)\) whose error in approximating \(\hat{N}\) is \(\epsilon\). Equivalently, there exists a neurosymbolic program \((P, \theta)\) whose error in approximating \(f\) is \((\epsilon + \epsilon)\). That is, if the relationship between treatment and effect is a continuous function, neurosymbolic programming is a viable candidate for estimating treatment effects.

5 EXPERIMENTS AND RESULTS

We perform experiments to showcase the usefulness of NESTER in estimating treatment effects when coupled with our proposed DSL. Our code along with instructions for reproducibility of results is in the supplementary material. To permit interpretability, we limit the program depth to utmost 5 for the main experiments (see Appendix for experiments with other depths).

Datasets and Baselines: Evaluating treatment effect estimation methods requires all potential outcomes to be available, which is impossible due to the fundamental problem of causal inference. Thus, following (Shalit et al., 2017; Yoon et al., 2018; Shi et al., 2019; Farajtabar et al., 2020), we experiment on two semi-synthetic datasets—Twins (Almond et al., 2005), IHDP (Hill, 2011)—that are derived from real-world RCTs (see Appendix 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. Each dataset is split 64/16/20% into train/validation/test sets, similar to earlier efforts.

We compare NESTER with Ordinary Least Squares with treatment as a feature (OLS-1), OLS with two regressors for two treatments (OLS-2), k-Nearest Neighbors (k-NN), balancing linear regression (BLR) (Johansson et al., 2016), Bayesian additive regression trees (BART) (Chipman et al., 2010), random forest (Breiman, 2001), causal forest (Wager & Athey, 2018), balancing neural network (BNN) (Johansson et al., 2016), treatment-agnostic representation network (TARNet) (Shalit et al., 2017), multi-head network (MHNET) (Farajtabar et al., 2020), Generative Adversarial Nets for inference of individualized treatment effects (GANITE) (Yoon et al., 2018), counterfactual regression with Wasserstein distance (CFR\(_{WASS}\)) (Shalit et al., 2017), Dragonnet (Shi et al., 2019) and multi-task Gaussian process (CMGP) (Alaa & van der Schaar, 2017).

Evaluation Metrics: For the experiments on IHDP and Twins datasets where we have access to both potential outcomes, following (Shalit et al., 2017; Yoon et al., 2018; Shi et al., 2019; Farajtabar et al., 2020), we use the evaluation metrics—Error in estimation of Average Treatment Effect (\(\epsilon_{ATE}\)) and Precision in Estimation of Heterogeneous Effect (\(\epsilon_{PEHE}\)). \(\epsilon_{ATE}\) is a global measure in the sense that it measures the error in the estimation of average treatment effect in a population. \(\epsilon_{PEHE}\) is a local measure as it operates on the error in the estimation of individual treatment effects. For
Appendix.

(iii) Use a specific program primitive multiple times. This flexibility allows

This paper presents a new neurosymbolic programming approach for treatment effect estimation,

Twins, Jobs compared to complex models).

Table 3: Results on IHDP, Twins, and Jobs datasets. Lower is better. The best numbers are in bold. Second best numbers are underlined. Simple machine learning models, ensemble models, and neural network based models are separated using horizontal lines. See Appendix for further analysis on k-NN results.

the experiment on the Jobs dataset where we observe only one potential outcome per data point, following (Shalit et al. 2017) Yoon et al. 2018, Shi et al. 2019, Farajtabar et al. 2020, we use the metric Error in estimation of Average Treatment Effect on the Treated ($\epsilon_{ATT}$). Mathematical definitions and details of these metrics are provided in the Appendix. Following (Shalit et al. 2017) Shi et al. 2019, Yoon et al. 2018, we report both in-sample and out-of-sample performance w.r.t. $\sqrt{\text{PEHE}}, \epsilon_{ATE}, \epsilon_{ATT}$ in our results. The in-sample evaluation is non-trivial since we do not observe counterfactual outcomes (all potential outcomes) even during training.

From the results in Table 3 except w.r.t. in-sample $\epsilon_{ATT}$ score in Jobs dataset, NESTER either outperforms or is competitive with the best alternative methods. Our method has the flexibility to learn both complex models that are required for small and complex datasets such as IHDP (complex models such as CMGP outperforms simple models such as OLS on IHDP) and to learn simple models to solve large and simple datasets such as Twins and Jobs (OLS, k-NN often perform better on Twins, Jobs compared to complex models).

Flexibility in Applying Inductive Bias and Program Primitives as Regularizers: Inductive biases, a set of assumptions we make to solve a ML problem, have a significant impact on the ML model performance at test time (Mitchell 1980). For a given task, inductive biases are chosen based on the intuition that a particular way of problem-solving is better than others. These intuitions either come from domain knowledge or from data analysis. As discussed earlier, these inductive biases are implicit in program primitives in the proposed DSL. Using the proposed DSL, for each dataset, NESTER has the flexibility to: (i) Choose or not choose a specific program primitive; (ii) Decide order in which the program primitives are used; and (iii) Use a specific program primitive multiple times. This flexibility allows NESTER to use inductive biases differently for different datasets to perform better. Table 4 shows the best programs synthesized by NESTER for IHDP, Twins, and Jobs datasets. Unlike traditional fixed architectures (e.g., IPM regularization followed by two head network in CFRNet), NESTER synthesizes path flows (equivalent to different architectures) to solve each dataset. Additional experimental details including analysis on the depth of synthesized programs, impact of the choice of DSL are provided in the Appendix.

6 Conclusions

This paper presents a new neurosymbolic programming approach for treatment effect estimation, and also studies why neurosymbolic programming is a good choice for solving such a problem. By making an analogy between parameterized program primitives and the basic building blocks of machine learning models in the literature on treatment effect estimation, we propose a Domain

| Datasets | IHDP | Twins | Jobs |
| --- | --- | --- | --- |
| Metrics | $\epsilon_{ATE}$ | $\epsilon_{ATE}$ | $\epsilon_{ATT}$ |
| 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 | .0029 ± .0025 | .0070 ± .0059 | .01 ± .01 | .08 ± .03 |
| BLR | .72 ± .04 | .93 ± .05 | .0057 ± .0036 | .0334 ± .0092 | .01 ± .01 | .08 ± .03 |
| k-NN | .14 ± .01 | .90 ± .05 | .0028 ± .0021 | .0051 ± .0039 | .21 ± .01 | .13 ± .05 |
| BART | .23 ± .01 | .34 ± .02 | .1206 ± .0236 | .1265 ± .0234 | .02 ± .00 | .08 ± .03 |
| R Forest | .73 ± .05 | .96 ± .06 | .0049 ± .0034 | .0080 ± .0051 | .03 ± .01 | .09 ± .04 |
| C 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 | .04 ± .00 |
| CMGP | .11 ± .10 | .13 ± .12 | .0124 ± .0051 | .0143 ± .0116 | .06 ± .06 | .09 ± .07 |
| NESTER | .06 ± .04 | .09 ± .07 | .0034 ± .0026 | .0063 ± .0033 | .06 ± .00 | .02 ± .01 |

Table 4: Sample programs learned by NESTER. $|v|$ = size of vector $v$. 

| Datasets | IHDP | Twins | Jobs |
| --- | --- | --- | --- |
| Metrics | $\epsilon_{ATE}$ | $\epsilon_{ATE}$ | $\epsilon_{ATT}$ |
| 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 | .0029 ± .0025 | .0070 ± .0059 | .01 ± .01 | .08 ± .03 |
| BLR | .72 ± .04 | .93 ± .05 | .0057 ± .0036 | .0334 ± .0092 | .01 ± .01 | .08 ± .03 |
| k-NN | .14 ± .01 | .90 ± .05 | .0028 ± .0021 | .0051 ± .0039 | .21 ± .01 | .13 ± .05 |
| BART | .23 ± .01 | .34 ± .02 | .1206 ± .0236 | .1265 ± .0234 | .02 ± .00 | .08 ± .03 |
| R Forest | .73 ± .05 | .96 ± .06 | .0049 ± .0034 | .0080 ± .0051 | .03 ± .01 | .09 ± .04 |
| C 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 | .04 ± .00 |
| CMGP | .11 ± .10 | .13 ± .12 | .0124 ± .0051 | .0143 ± .0116 | .06 ± .06 | .09 ± .07 |
| NESTER | .06 ± .04 | .09 ± .07 | .0034 ± .0026 | .0063 ± .0033 | .06 ± .00 | .02 ± .01 |
Specific Language on which program synthesis is rooted. Our results and analysis on benchmark datasets with several baselines show the usefulness of the proposed approach. Exploring new program primitives corresponding to unexplored heuristics for the treatment effect estimation task is an interesting future direction.

REFERENCES

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

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

Ahmed M. Alaa and Mihaela van der Schaar. Bayesian inference of individualized treatment effects using multi-task gaussian processes. In *NIPS*, 2017.

Douglas Almond, Kenneth Y. Chay, and David S. Lee. The costs of low birth weight. *The Quarterly Journal of Economics*, 120(3):1031–1083, 2005.

Serge Assaad, Shuxi Zeng, Chenyang Tao, Shounak Datta, Nikhil Mehta, Ricardo Henao, Fan Li, and Lawrence Carin Duke. Counterfactual representation learning with balancing weights. In *AISTATS*, 2021.

Ioana Bica, James Jordon, and Mihaela van der Schaar. Estimating the effects of continuous-valued interventions using generative adversarial networks. In *NeurIPS*, 2020.

Alan W. Biermann. The inference of regular lisp programs from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(8):585–600, 1978.

Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable forth interpreter. In *ICML*, 2017.

Henry Brady, David Collier, and Jasjeet Sekhon. The neyman-rubin model of causal inference and estimation via matching methods. *The Oxford Handbook of Political Methodology*, 01 2008.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

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

Thomas C. Chalmers, Harry Smith, Bradley Blackburn, Bernard Silverman, Biruta Schroeder, Dinah Reitman, and Alexander Ambroz. A method for assessing the quality of a randomized control trial. *Controlled Clinical Trials*, 2(1):31–49, 1981.

Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, and Yisong Yue. Neurosymbolic programming. *Foundations and Trends® in Programming Languages*, 7(3):158–243, 2021.

Victor Chernozhukov, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1):C1–C68, 2018.

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

Zhixuan Chu, Stephen L. Rathbun, and Sheng Li. Matching in selective and balanced representation space for treatment effects estimation. In *CIKM*, 2020.

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

RICHARD K. CRUMP, V. JOSEPH HOTZ, GUIDO W. IMBENS, and OSCAR A. MITNIK. Dealing with limited overlap in estimation of average treatment effects. *Biometrika*, 96(1):187–199, 2009.
Alicia Curth and Mihaela van der Schaar. On inductive biases for heterogeneous treatment effect estimation. In *Advances in Neural Information Processing Systems*, 2021.

Marco Cuturi and Arnaud Doucet. Fast computation of wasserstein barycenters. In *ICML*, 2014.

Alexis Diamond and Jasjeet S Sekhon. 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, 2013.

Vincent Dorie. Npci: Non-parametrics for causal inference., 2016. URL [https://github.com/vdorie/npci](https://github.com/vdorie/npci).

Mehrdad Farajtabar, Andrew Lee, Yuanjian Feng, Vishal Gupta, Peter Dolan, Harish Chandran, and Martin Szummer. Balance regularized neural network models for causal effect estimation. *CoRR*, abs/2011.11199, 2020.

Alexander L. Gaunt, Marc Brockschmidt, Nate Kushman, and Daniel Tarlow. Differentiable programs with neural libraries. In *ICML*, 2017.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *JMLR*, 13(25):723–773, 2012.

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

Sumit Gulwani, William R. Harris, and Rishabh Singh. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105, 2012.

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

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

M.A. Hernan and J.M. Robins. *Causal Inference*. Taylor & Francis, 2019.

Jennifer L. Hill. Bayesian nonparametric modeling for causal inference. *Journal of Computational and Graphical Statistics*, 20(1):217–240, 2011.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

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

Fredrik Johansson, Uri Shalit, and David Sontag. Learning representations for counterfactual inference. In *ICML*, 2016.

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

Li L and Greene T. A weighting analogue to pair matching in propensity score analysis. *Int J Biostat*, 9(2):215–234, 2013.

Robert J. LaLonde. Evaluating the econometric evaluations of training programs with experimental data. *The American Economic Review*, 76(4):604–620, 1986.

Michael Lechner. Identification and estimation of causal effects of multiple treatments under the conditional independence assumption. In *Econometric evaluation of labour market policies*, pp. 43–58. Springer, 2001.
Sheng Li and Yun Fu. Matching on balanced nonlinear representations for treatment effects estimation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), Advances in Neural Information Processing Systems, 2017.

Tom M Mitchell. The need for biases in learning generalizations. Department of Computer Science, Laboratory for Computer Science Research . . . , 1980.

Stephen L. Morgan and Christopher Winship. Counterfactuals and Causal Inference: Methods and Principles for Social Research. Analytical Methods for Social Research. Cambridge University Press, 2 edition, 2014.

Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. arXiv preprint arXiv:1611.01855, 2016.

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

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

Judea Pearl. Causality. Cambridge university press, 2009.

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

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

Donald B Rubin. Causal inference using potential outcomes. Journal of the American Statistical Association, 100(469):322–331, 2005.

Robert William Sanson-Fisher, Billie Bonevski, Lawrence W. Green, and Cate D’Este. Limitations of the randomized controlled trial in evaluating population-based health interventions. American Journal of Preventive Medicine, 33(2):155–161, 2007.

Patrick Schwab, Lorenz Linhardt, Stefan Bauer, Joachim M Buhmann, and Walter Karlen. Learning counterfactual representations for estimating individual dose-response curves. In AAAI, 2020.

Ameesh Shah, Eric Zhan, Jennifer Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. Learning differentiable programs with admissible neural heuristics. In NeurIPS, 2020.

Uri Shalit, Fredrik D Johansson, and David Sontag. Estimating individual treatment effect: generalization bounds and algorithms. In ICML, 2017.

Claudia Shi, David Blei, and Victor Veitch. Adapting neural networks for the estimation of treatment effects. In NeurIPS, 2019.

Armando Solar Lezama. Program Synthesis By Sketching. PhD thesis, EECS Department, University of California, Berkeley, Dec 2008.

Armando Solar-Lezama, Rodric Rabbah, Rastislav Bodik, and Kemal Ebcioglu. Programming by sketching for bit-streaming programs. SIGPLAN Not., 40(6):281–294, 2005.

Elizabeth A Stuart. 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, 2010.

Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. Houdini: Lifelong learning as program synthesis. In NeurIPS, 2018.

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

Glynn Winskel. The Formal Semantics of Programming Languages: An Introduction. MIT Press, Cambridge, MA, USA, 1993.
Liuyi Yao, Sheng Li, Yaliang Li, Mengdi Huai, Jing Gao, and Aidong Zhang. Representation learning for treatment effect estimation from observational data. In NeurIPS, 2018.

Jinsung Yoon, James Jordon, and Mihaela van der Schaar. GANITE: estimation of individualized treatment effects using generative adversarial nets. In ICLR, 2018.

Weijia Zhang, Lin Liu, and Jiuyong Li. Treatment effect estimation with disentangled latent factors. In AAAI, 2021.
APPENDIX

In this appendix, we include the following additional information, which we could not fit in the main paper due to space constraints:

- Proofs of propositions
- Additional details on experimental setup, including:
  - Additional details on evaluation metrics
  - Details on datasets
- Additional results and ablation studies:
  - Results with $\epsilon$-admissible metric
  - Analysis on the choice of domain specific language
  - Analyzing the depth of synthesized program structures
  - Analysis on Twins dataset
- Example of program synthesis application - FlashFill
- Example of a neurosymbolic program - solving XOR problem
- Interpretability of Synthesized Programs - A real-world example

A PROOFS OF PROPOSITIONS

Proposition 4.1. In an informed search algorithm, let the cost of the leaf edge $(u_i, u_l)$ (edge connecting internal node $u_i$ to leaf node $u_l$) be $s(r) + \zeta(P, \theta^*)$, where $\theta^* = \arg \min_{\theta} \zeta(P, \theta)$ and $r$ is the rule used to create $u_l$ from $u_i$. If NNs $\mathcal{N}$ parameterized by their capacity (architecture width and height) are used to substitute the non-terminals in the partial structure of $u_i$, the resultant program’s training loss is equal to the $\epsilon$-admissible heuristic value at the node $u_i$. Such an $\epsilon$-admissible heuristic returns a solution whose path cost is at most an additive constant $\epsilon$ away from the path cost of the optimal solution (Shah et al., 2020).

Proof. Let $\mathcal{G}$ denote the program graph that is being generated by an informed search algorithm. At any node $u$ in $\mathcal{G}$, let $s(u)$ be the structural cost of $u$, i.e., the sum of costs of rules used to construct $u$. Now, let $u[\alpha_1, \ldots, \alpha_k]$ be any structure (that is not partial) obtained from $u$ by using the rules $\alpha_1, \ldots, \alpha_k$. Then the cost to reach goal node from $u$ is given by:

$$J(u) = \min_{\alpha_1, \ldots, \alpha_k, \theta(u)} \left[ s(u[\alpha_1, \ldots, \alpha_k]) - s(u) + \zeta(u[\alpha_1, \ldots, \alpha_k], (\theta_u, \theta)) \right]$$

where $\theta(u)$ is the set of parameters of $u$ and $\theta$ is the set of parameters of $\alpha_1, \ldots, \alpha_k$. Now, let the heuristic function value $h(u)$ at $u$ be obtained as follows: substitute the non-terminals in $u$ with neural networks parameterized by the set of parameters $\omega$ (these networks are type-correct— for example, if a non-terminal is supposed to generate sub-expressions whose inputs are sequences, then the neural network used in its place is recurrent). Now, let us denote the program obtained by this construction with $(P(u), (\theta(u), \omega))$. The heuristic function value at $u$ is now given by:

$$h(u) = \min_{\theta(u), \omega} \zeta(P(u), (\theta(u), \omega))$$

In practice, neural networks may only form an approximate relaxation of the space of completions and parameters of architectures; also, the training of these networks may not reach global optima. To account for these issues, consider an approximate notion of admissibility (Harris, 1974; Pearl, 1984). For a fixed constant $\epsilon > 0$, let an $\epsilon$-admissible heuristic be a function $h^*(u)$ over architectures such that $h^*(u) \leq J(u) + \epsilon, \forall u$.

As neural networks with adequate capacity are universal function approximators, there exist parameters $\omega^*$ for our neurosymbolic program such that for all $u, \alpha_1, \ldots, \alpha_k, \theta(u), \theta$:

$$\zeta(P(u), (\theta(u), \omega^*)) \leq \zeta(P(u[\alpha_1, \ldots, \alpha_k]), (\theta(u), \theta)) + \epsilon$$

If $s(r) > 0; \forall r \in \mathcal{L}$ (where $\mathcal{L}$ is the DSL under consideration), then $s(u) \leq s(u[\alpha_1, \ldots, \alpha_k])$, which implies:

$$h(u) \leq \min_{\alpha_1, \ldots, \alpha_k, \theta(u), \theta} \zeta(u[\alpha_1, \ldots, \alpha_k], (\theta_u, \theta)) + \epsilon$$

$$\leq \min_{\alpha_1, \ldots, \alpha_k, \theta(u), \theta} \zeta(u[\alpha_1, \ldots, \alpha_k], (\theta_u, \theta)) + s(u[\alpha_1, \ldots, \alpha_k]) - s(u) + \epsilon$$

$$= J(u) + \epsilon$$
In other words, $h(u)$ is $\epsilon$-admissible.

Let $C$ denote the optimal path cost in $G$. If an informed search algorithm returns a node $u_g$ as the goal node that does not have the optimal path cost $C$, then there must exist a node $u'$ on the frontier (nodes to explore) that lies along the optimal path but has not yet explored. Let $g(u_g)$ denote the path cost at $u_g$ (note that path cost includes the prediction error of the program at $u_g$). This lets us establish an upper bound on the path cost of $u_g$.

$$g(u_g) \leq g(u') + h(u') \leq g(u') + J(u') + \epsilon \leq C + \epsilon. \quad (6)$$

In an informed search algorithm, the heuristic estimate at the goal node $h(u_g)$ is 0. That is, the path cost of the optimal program returned by the informed search algorithm is at most an additive constant $\epsilon$ away from the path cost of the optimal solution.

**Proposition 4.2.** Given an $\epsilon$-admissible heuristic, for any trained 1-hidden layer NN $\mathcal{N}$ with $m$ inputs, $n$ hidden neurons, and one output, there exist a Domain Specific Language $\mathcal{L}$ such that the error/loss incurred by the synthesized program $(\mathcal{P}, \theta)$ is $\epsilon$-close to the error/loss incurred by $\mathcal{N}$ in approximating any continuous function.

**Proof.** Consider a trained 1-hidden layer neural network $\mathcal{N}$ with $m$ inputs $x_1, \ldots, x_m$, $n$ hidden neurons $h_1, \ldots, h_n$, and output $y$. Let the activation function used in hidden and output layers be $g(\cdot)$; $\theta_{ij}$ be the weight connecting $i^{th}$ input to $j^{th}$ hidden neuron; and $\theta_j$ be the weight connecting $j^{th}$ hidden neuron to output $y$. The output $y$ of $\mathcal{N}$ can be expressed in terms of inputs, activations, and parameters as:

$$y = g(\theta_1 g(\theta_{11} x_1 + \cdots + \theta_{1m} x_m) + \cdots + \theta_n g(\theta_{1n} x_1 + \cdots + \theta_{mn} x_m)) \quad (7)$$

Since the expression for $y$ consists of additions, multiplications, and a known activation function $g$, we can synthesize the same expression (Equation (7)) using the following DSL $\mathcal{L}$ where $\text{mul}$, $\text{add}$ represent usual multiplication and addition operations.

$$\alpha := g(\alpha) \mid \text{mul}(\alpha, \alpha) \mid \text{add}(\alpha, \alpha) \mid x_1 \mid \ldots \mid x_n \quad (8)$$

For example, if $m = 2$ and $n = 2$, the synthesized program that matches the expression for $y$ looks like: $g(\text{add}(\text{mul}(\theta, g(\text{add}(\text{mul}(\theta, x_1), \text{mul}(\theta, x_2))))), \text{mul}(\theta, g(\text{add}(\text{mul}(\theta, x_1), \text{mul}(\theta, x_2))))).$

**Side note:** $\theta$ is overloaded in the previous expression only for convenience and staying in line with typical program synthesis expressions. Each $\theta$ is however updated independently while training the above program using gradient descent.

It is clear that the expression for $y$ can be synthesized using $\mathcal{L}$ for any given $m, n$. Now, as part of our construction, set $s(r) = 0; \forall r \in \mathcal{L}$ to synthesize programs of arbitrary depth and width without worrying about structural cost of the synthesized program. Now the path cost $p$ of a node $u$ returned by the synthesizer contains only the prediction error value of the program at the node $u$. Using Proposition 4.1, $p$ is at most $\epsilon$ away from the path cost of the optimal solution (node with the expression for $y$, the output of $\mathcal{N}$). Since path cost of any node only contains the prediction error values, we conclude that the error/loss incurred by the synthesized program is $\epsilon$-close to the error/loss incurred by $\mathcal{N}$. \hfill $\square$

## B Experimental Setup

### B.1 Additional Details on Evaluation Metrics

For the experiments on IHDP and Twins datasets where we have access to both potential outcomes, following [Shalit et al. 2017] [Yoon et al. 2018] [Shi et al. 2019] [Farajtabar et al. 2020], we use the evaluation metrics: *Error in estimation of Average Treatment Effect $(\epsilon_{ATE})$* and *Precision in Estimation of Heterogeneous Effect $(\epsilon_{PEHE})*. These are defined as follows for finite sample datasets.
of $n$ data points.

$$\epsilon_{ATE} := \frac{1}{n} \sum_{i=1}^{n} [f(x_i, 1) - f(x_i, 0)] - \frac{1}{n} \sum_{i=1}^{n} [Y_i^1 - Y_i^0]$$

$$\epsilon_{PEHE} := \frac{1}{n} \sum_{i=1}^{n} [(f(x_i, 1) - f(x_i, 0)) - (Y_i^1 - Y_i^0)]^2$$

For the experiment on the Jobs dataset where we observe only one potential outcome per data point, following (Shalit et al., 2017; Yoon et al., 2018; Shi et al., 2019; Farajtabar et al., 2020), we use the metric Error in estimation of Average Treatment Effect on the Treated ($\epsilon_{ATT}$), which is defined as follows.

$$\epsilon_{ATT} := |ATT^{true} - \frac{1}{|T|} \sum_{i \in T} [f(x_i, 1) - f(x_i, 0)]|$$

where $ATT^{true}$ is defined as:

$$ATT^{true} := \frac{1}{|T|} \sum_{i \in T} Y_i^1 - \frac{1}{|U \cap E|} \sum_{i \in U \cap E} Y_i^0$$

and $T$ is the treated group, $U$ is control group, and $E$ is the set of data points from a randomized experiment (Shalit et al., 2017) (see description of Jobs dataset below for an example of $E$, $T$, and $U$).

In k-NN where $k=1$, if treatment value $t=1$, $f(x_i, 1)$ is exactly same as $Y_i^1$. If treatment value $t=0$, $f(x_i, 0)$ is exactly same as $Y_i^0$ because of the way k-NN works during test time on in-sample data. For this reason, the estimated value of $\epsilon_{ATE}$ is biased towards 0. This bias exists even for higher values of $k$ in k-NN while taking the average outputs of $k$ nearest data points. However, we do not observe such bias w.r.t. out-sample data. Hence, following earlier work [63], we only consider K-NN results for out-sample performance. We updated Table 3 caption to clarify this.

### B.2 Details on Datasets

**IHDP:** Infant Health and Development Program (IHDP) is a randomized control experiment on 747 low-birth-weight, premature infants. The treatment group consists of 139 children, and the control group has 608 children. The treatment group received additional care such as frequent specialist visits, systematic educational programs, and pediatric follow-up. The Control group only received pediatric follow-up. (Hill, 2011) created the semi-synthetic version of IHDP dataset by synthesizing both potential outcomes. Following (Hill, 2011; Shalit et al., 2017; Yoon et al., 2018; Shi et al., 2019; Farajtabar et al., 2020), we use simulated outcomes of the IHDP dataset from NPCI package (Dorie, 2016). This experiment aims to estimate the effect of treatment on the IQ score of children at the age of 3.

**Twins:** The Twins dataset is derived from all births in the USA between 1989-1991 (Almond et al., 2005). Considering twin births in this period, for each child, we estimate the effect of birth weight on 1-year mortality rate. Treatment $t = 1$ refers to the heavier twin and $t = 0$ refers to the lighter twin. Following (Yoon et al., 2018), for each twin-pair, we consider 30 features relating to the parents, the pregnancy, and the birth. We only consider twins weighing less than 2kg and without missing features. The final dataset has 11,400 pairs of twins whose mortality rate for the lighter twin is 17.7%, and for the heavier 16.1%. In this setting, for each twin pair we observed both the case $t = 0$ (lighter twin) and $t = 1$ (heavier twin) (that is, since all other features such as parent’s race, health status, gestation weeks prior to birth, etc. are same except the weight of each twin, the choice of twin (lighter vs heavier) is associated with the treatment ($t = 0$ vs $t = 1$)); thus, the ground truth of individualized treatment effect is known in this dataset. In order to simulate an observational study from these 11,400 pairs, following (Yoon et al., 2018), we selectively observe one of the two twins using the feature information $x$ (to create selection bias) as follows: $t|x \sim \text{Bernoulli}(\text{sigmoid}(w^T x + n))$ where $w^T \sim U((-0.1, 0.1)^{30 \times 1})$ and $n \sim N(0, 0.1)$.

**Jobs:** The Jobs dataset is a widely used real-world benchmark dataset in causal inference. In this dataset, the treatment is job training, and the outcomes are income and employment status after job training. The dataset combines a randomized study based on the National Supported Work Program in the USA (we denote the set of observations from this randomized study with $E$) with observational
data (A. Smith & E. Todd, 2005). Each observation contains 18 features such as age, education, previous earnings, etc. Following (Shalit et al., 2017; Yoon et al., 2018), we construct a binary classification task, where the goal is to predict unemployment status given a set of features. The Jobs dataset is the union of 722 randomized samples (t = 1 : 297, t = 0 : 425) and 2490 observed samples (t = 1 : 0, t = 0 : 2490). The treatment variable is job training (t = 1 if trained for job else t = 0), and the outcomes are income and employment status after job training. In Equations 11-12, we then have |T| = 297, |C| = 2915, |E| = 722. Since all the treated subjects T were part of the original randomized sample E, we can compute the true ATT (Equation 12) and hence can study the precision in estimation of ATT (Equation 11).

Table 5 summarizes the dataset details. All experiments were conducted on a computing unit with a single NVIDIA GeForce 1080Ti.

| Dataset | Number of Data points (Including Treatment) | Input Size | Batch Size | Training Epochs | Train/Valid/Test Split (%) |
|---------|-----------------------------------------------|------------|------------|-----------------|----------------------------|
| IHDP    | 747 (1000 such instances)                     | 26         | 16         | 100             | 64/16/20                   |
| Twins   | 11400                                         | 31         | 128        | 7               | 64/16/20                   |
| Jobs    | 3212                                          | 18         | 64         | 10              | 64/16/20                   |

Table 5: Dataset details. ‘Input Size’ includes treatment variable.

C.1 RESULTS WITH ϵPEHE METRIC

To study how NESTER performs with the ϵPEHE metric, we empirically captured the performance of NESTER comprehensively against all baselines on the IHDP dataset. From Table 6, NESTER achieves strong out-sample (out-of-sample) ϵPEHE score on the IHDP dataset, even on this metric.

| Dataset (Metric) | IHDP (ϵPEHE) |
|-----------------|--------------|
| Methods ↓       | In-Sample    | Out-Sample  |
| OLS-1           | 5.80 ± .30   | 5.80 ± .30  |
| OLS-2           | 2.50 ± .10   | 2.50 ± .10  |
| BLR             | 5.80 ± .30   | 5.80 ± .30  |
| k-NN            | 2.10 ± .10   | 4.10 ± .20  |
| BART            | 2.10 ± .10   | 2.3 ± .10   |
| R Forest        | 4.20 ± .20   | 6.60 ± .30  |
| C Forest        | 3.80 ± .20   | 3.80 ± .20  |
| BNN             | 2.20 ± .10   | 2.10 ± .10  |
| TARNet          | .88 ± .02    | .95 ± .02   |
| MHNET           | 1.54 ± .70   | 1.89 ± .52  |
| GANITE          | 1.90 ± .40   | 2.40 ± .40  |
| CFRWASS         | .71 ± .02    | .76 ± .02   |
| Dragonnet       | 1.37 ± 1.57  | 1.42 ± 1.67 |
| CMGP            | .65 ± .44    | .77 ± .11   |
| NESTER          | .73 ± .19    | .76 ± .20   |

Table 6: Results on IHDP dataset. Lower is better.

C.2 CHOICE OF DSL

The choice of DSL has a huge impact on the performance of NESTER. We argue that the success of NESTER is because of the specific program primitives in the proposed DSL and their connection to the causal inference literature (Table I). Specifically, we study the usefulness of the primitives if − then − else, transform, subset. We conduct an ablation study where the DSL only contains the subset of primitives from the set of primitives 1-5 in the original DSL (Table 2). When we remove the primitives 1-3 from the DSL, we observe the degradation in the performance (Table 7). Results improved when we added the primitives 1-3 in the DSL.
Table 7: Results on Twins. Primitives 1-4 alone in our proposed DSL are achieving better results compared to the primitives 4-5.

### C.3 Analysis on Depth of Synthesized Program Structures

We study the effect of program depth on the estimated treatment effects while keeping all other hyperparameters fixed. Figure 2 shows the results on IHDP and Jobs datasets for various values of program depth. Since IHDP dataset contains 1000 realizations of simulated outcomes (Hill, 2011), we take the first instance and verify the effect of program depth on $\epsilon_{ATE}$. For program depth of 4, we observed a better trade-off between in-sample and out-sample $\epsilon_{ATE}$. Any depth smaller than 4 and higher than 4 results in degradation of performance w.r.t. one of in-sample or out-sample $\epsilon_{ATE}$. We believe that this is because of model over-fitting for large program depths (In Figure 2 left, out-sample $\epsilon_{ATE}$ is increasing while in-sample $\epsilon_{ATE}$ is decreasing). In the Jobs dataset, we observed that almost all program depths result in similar in-sample and out-sample $\epsilon_{ATT}$. Hence, in this case it is advisable to limit the program depth to be a small number as it helps to interpret the results better. On Twins dataset, as stated in the main paper (L 395), we observed that simple models give best results. It is observed that, even though we set the hyperparameter that controls the depth of the program graph to be a large value, the resultant optimal program always ends up to be of depth 1, again supporting our claim that simple models work better for the Twins dataset.

### C.4 Analysis on Twins Dataset

We study the program synthesized for the Twins dataset. NESTER generates simple program ($\text{subset}(\mathbf{v}, [0..|\mathbf{v}|])$) for the Twins dataset (Table 4). Since the $\text{subset}$ primitive allows us check the performance w.r.t. different subsets of covariates, we empirically verified the effect of choosing a subset of input covariates (other covariates are set to 0) on the predicted $ATE$. Results in Figure 3 show the performance of NESTER as the number of covariates are increased from 1 to 31 (starting with treatment variable, adding one covariate at a time). We observe that the model with all features included gives the best in-sample and out-sample $\epsilon_{ATE}$. While this is not a surprising conclusion, the choice of the $\text{subset}$ primitive allows us such an analysis. Also, this simple program synthesized by NESTER supports the fact that simpler models perform better on the Twins dataset. This can be observed from first three rows and final row of Table 3.

### D FlashFill Task and Semantics of its DSL

Following our discussion in Section 1 (L 75), for better understanding of symbolic program synthesis, we provide an example of a symbolic program application called FlashFill (Parisotto et al., 2016). Examples of the FlashFill task and a DSL to synthesize programs that solve FlashFill task are given in Table 8.
We expect that each program primitive in a domain-specific language has a semantic meaning; hence, right which last name is followed by the initial of the first name; left x T,X on the output? etc. sequence of program primitives mean for the problem? what is the effect of each program primitive using various aspects such as: which program primitives are used and why? what does the learned interpretability in program synthesis refers to understanding the decision of a synthesized program training through gradient descent.

Unlike symbolic programs, neurosymbolic programs are differentiable and can be trained using gradient descent. Program primitives in a neurosymbolic program have trainable parameters associated with them. The program shown in Table 9 (left) is constructed using (i) if – then – else and (ii) affine program primitives. affine primitive takes a vector as input and returns a scalar that is the sum of dot product of parameters with the input and a bias parameter. For example, if x = [1, 0] then affine_{\theta_1, \theta_2, \theta_3}(x) = \theta_1 \times x + \theta_2 \times 0 + \theta_3 = \theta_1 + \theta_3. The subscripts of affine in affine_{\theta_1, \theta_2, \theta_3} contain the parameters \theta_1, \theta_2 and bias parameter \theta_3 separated by semi colon (;). The smooth approximation of this program, to enable backpropagation, is shown in Table 9 (right). The parameter values are hard-coded for illustration purposes. In practice, these weights are learned by training through gradient descent.

### E Neurosymbolic Program Example: Solving XOR Problem

Following our discussion in Section 3 (L 194), for better understanding of the internal workings of a neurosymbolic program, we provide an example on solving the XOR problem i.e., predicting the output of XOR operation given two binary digits.

Unlike symbolic programs, neurosymbolic programs are differentiable and can be trained using gradient descent. Program primitives in a neurosymbolic program have trainable parameters associated with them. The program shown in Table 9 (left) is constructed using (i) if – then – else and (ii) affine program primitives. affine primitive takes a vector as input and returns a scalar that is the sum of dot product of parameters with the input and a bias parameter. For example, if x = [1, 0] then affine_{\theta_1, \theta_2, \theta_3}(x) = \theta_1 \times x + \theta_2 \times 0 + \theta_3 = \theta_1 + \theta_3. The subscripts of affine in affine_{\theta_1, \theta_2, \theta_3} contain the parameters \theta_1, \theta_2 and bias parameter \theta_3 separated by semi colon (;). The smooth approximation of this program, to enable backpropagation, is shown in Table 9 (right). The parameter values are hard-coded for illustration purposes. In practice, these weights are learned by training through gradient descent.

### F Interpretability of Synthesized Programs: A Real World Example

We expect that each program primitive in a domain-specific language has a semantic meaning; hence, interpretability in program synthesis refers to understanding the decision of a synthesized program using various aspects such as: which program primitives are used and why? what does the learned sequence of program primitives mean for the problem? what is the effect of each program primitive on the output? etc.

We explain more clearly with an example. Consider a causal model consisting of variables T, X₁, X₂, Y where: (i) X₁ causes T and Y; (ii) T causes X₂ and Y; and (iii) X₂ causes Y.
A real-world scenario depicted by this causal model could be where \( T \) is the average distance walked by a person in a day, \( X_1 \) is age, \( X_2 \) is metabolism, and \( Y \) is blood pressure. In this example, our goal is to estimate the effect of walking \( (T) \) on blood pressure \( (Y) \). In this case, the ideal estimator for the quantity \( \mathbb{E}[Y|do(t)] = \sum_{x_1} \mathbb{E}[Y|t, x_1] p(x_1) \). However, NESTER has access to only observational data and is unaware of the underlying causal process. Now consider the following two possible programs \( p_1, p_2 \) that are synthesized by NESTER to estimate the treatment effect of \( T \) on \( Y \). Let \( \mathbf{v} = [t, x_1, x_2] \) be an input data point.

\[
\begin{align*}
p_1 &: \text{if subset}(\mathbf{v}, [0..1]) \text{ then subset}(\mathbf{v}, [0..2]) \text{ else subset}(\mathbf{v}, [0..2]) \\
p_2 &: \text{if subset}(\mathbf{v}, [0..1]) \text{ then subset}(\mathbf{v}, [0..3]) \text{ else subset}(\mathbf{v}, [0..3])
\end{align*}
\]

The only difference between \( p_1 \) and \( p_2 \) is the set of indices used in subset primitives. \( p_1 \) uses only \( T, X_1 \) (indicated by [0..2] in \( p_1 \)) to predict \( Y \); while \( p_2 \) uses \( T, X_1, X_2 \) (indicated by [0..3] in \( p_2 \)) to predict \( Y \). In this case, we would ideally observe \( p_1 \) to perform better than \( p_2 \) because \( p_1 \) controls for the correct set of confounding variables (\( \{X_1\} \) in this case). Conversely, observing a strong performance for \( p_1 \) tells us that \( \{X_1\} \) is the confounder, without knowledge of the causal model.

Observing the generated program and primitives gives us insights about the underlying data generating process such as which features are the potential causes of treatment (e.g., age affects the average distance a person can walk), which features should not be controlled (e.g., we need the effect of walking on blood pressure irrespective of the metabolism rate of a person), etc. Such information encoded in a synthesized program can also be validated with domain experts if available. Our experimental results and ablation studies discussed above show other ways of interpreting programs.

Table 9: **Left:** A neurosymbolic program to solve XOR problem. **Right:** Smooth approximation of the program on the left where \( \sigma \) is sigmoid function. \( \beta \) is a temperature parameter. As \( \beta \to 0 \), the approximation approaches usual if \( \sim \) then \( \approx \) else (Section 4.1).