- Proofs of propositions
- Additional details on experimental setup, including:
  - Additional details on evaluation metrics
  - Details on datasets
- Additional results and ablation studies:
  - Results with ϵ_{PEHE} 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

7 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 $s(r) + \zeta(\mathcal{P}, \theta^*)$, where $\theta^* = \arg\min_\theta \zeta(\mathcal{P}, \theta)$ 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 ϵ 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:

$$J(u) = \min_{\alpha_1,\ldots,\alpha_n,\theta(u),\theta} [s(u(\alpha_1, \ldots, \alpha_k)) - s(u) + \zeta(u[\alpha_1, \ldots, \alpha_k], (\theta_u, \theta))] \tag{2}$$

where $\theta(u)$ is the set of parameters of u and θ 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 parametrized by the set of parameters ω (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 $(\mathcal{P}(u), (\theta(u), \omega))$. The heuristic function value at u is now given by:

$$h(u) = \min_{\theta(u),\omega} \zeta(\mathcal{P}(u), (\theta(u), \omega)) \tag{3}$$

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 ϵ-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 ω^* for our neurosymbolic program such that for all $u, \alpha_1, \ldots, \alpha_k, \theta(u), \theta$:

$$\zeta(P(u), (\theta(u), \omega^*)) \leq \zeta(P(u[\alpha_1, \ldots, \alpha_k]), (\theta(u), \theta)) + \epsilon \tag{4}$$

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:

$$\begin{aligned} h(u) &\leq \min_{\alpha_1,\ldots,\alpha_n,\theta(u),\theta} \zeta(u[\alpha_1, \ldots, \alpha_k], (\theta_u, \theta))) + \epsilon \\ &\leq \min_{\alpha_1,\ldots,\alpha_n,\theta(u),\theta} \zeta(u[\alpha_1, \ldots, \alpha_k], (\theta_u, \theta))) + s(u(\alpha_1, \ldots, \alpha_k)) - s(u) + \epsilon \\ &= J(u) + \epsilon \end{aligned} \tag{5}$$

In other words, $h(u)$ is ϵ-admissible.

Let C denote the optimal path cost in $\mathcal{G}$. 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.

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

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 ϵ 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 $g(\cdot)$; θ_{ij} be the weight connecting i^{th} input to j^{th} hidden neuron; and θ_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:

$$y = g(\theta_1 g(\theta_{11} x_1 + \cdots + \theta_{m1} x_m) + \cdots + \theta_n g(\theta_{1n} x_1 + \cdots + \theta_{mn} x_m)) \tag{7}$$

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 `mul`, `add` represent usual multiplication and addition operations.

$$\alpha \coloneqq \texttt{g}(\alpha) \mid \texttt{mul}(\theta, \alpha) \mid \texttt{add}(\alpha, \alpha) \mid \texttt{x}_1 \mid \ldots \mid \texttt{x}_\texttt{n} \tag{8}$$

For example, if $m = 2$ and $n = 2$, the synthesized program that matches the expression for y looks like: $\texttt{g}\,(\texttt{add}\,(\texttt{mul}\,(\theta,\, \texttt{g}(\,\texttt{add}\,(\texttt{mul}\,(\theta,\, \texttt{x}_1), \texttt{mul}(\theta,\, \texttt{x}_2)))),\, \texttt{mul}(\theta,\, \texttt{g}\,(\texttt{add}\,(\texttt{mul}(\theta,\, \texttt{x}_1), \texttt{mul}(\theta,\, \texttt{x}_2))))))$.

Side note: θ is overloaded in the previous expression only for convenience and staying in line with typical program synthesis expressions. Each θ 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 ϵ 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 $\mathcal{N}$. □

8 Experimental Setup

8.1 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 (ϵ_{ATE})* and *Precision in Estimation of Heterogeneous Effect (ϵ_{PEHE})*. These are defined as follows for finite sample datasets of n data points.

$$\epsilon_{ATE} \coloneqq |\frac{1}{n}\sum_{i=1}^{n}[f(\mathbf{x}_i, 1) - f(\mathbf{x}_i, 0)] - \frac{1}{n}\sum_{i=1}^{n}[Y_i^1 - Y_i^0]| \tag{9}$$

$$\epsilon_{PEHE} \coloneqq \frac{1}{n}\sum_{i=1}^{n}[(f(\mathbf{x}_i, 1) - f(\mathbf{x}_i, 0)) - (Y_i^1 - Y_i^0)]^2 \tag{10}$$

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* (ϵ_{ATT}), which is defined as follows.

$$\epsilon_{ATT} \coloneqq |ATT^{true} - \frac{1}{|T|}\sum_{i \in T}[f(\mathbf{x}_i, 1) - f(\mathbf{x}_i, 0)]| \tag{11}$$

where ATT^{true} is defined as:

$$ATT^{true} \coloneqq \frac{1}{|T|}\sum_{i \in T} Y_i^1 - \frac{1}{|U \cap E|}\sum_{i \in U \cap E} Y_i^0 \tag{12}$$

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, $f(\mathbf{x}_i, 1)$ is exactly same as Y_i^1. If treatment value t=0, $f(\mathbf{x}_i, 0)$ 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 ϵ_{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.

8.2 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), 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 $\mathbf{x}$ (to create selection bias) as follows: $t|\mathbf{x} \sim \texttt{Bernoulli}(\texttt{sigmoid}(\mathbf{w}^T\mathbf{x} + n))$ where $\mathbf{w}^T \sim U((-0.1, 0.1)^{30\times 1})$ 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

Metrics →	$\sqrt{\epsilon_{PEHE}}$		ϵ_{ATE}	
Primitives of DSL 2	In-Sample	Out-of=Sample	In-Sample	Out-of-Sample
1-4	**.318 ± .003**	**.319 ± .000**	.0050 ± .0030	**.0063 ± .0030**
4-5	.319 ± .002	**.319 ± .000**	.0170 ± .0010	.0100 ± .0000
1-5	**.318 ± .002**	**.319 ± .000**	**.0034 ± .0026**	**.0063 ± .0033**

Table 7: Results on Twins. Primitives 1-4 alone in our proposed DSL are achieving better results compared to the primitives 4-5.

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	Input Size (Including Treatment)	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.

9 Additional Results and Ablation Studies

9.1 Results with ϵ_{PEHE} Metric

To study how NESTER performs with the ϵ_{PEHE} 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) ϵ_{PEHE} score on the IHDP dataset, even on this metric.

Dataset (Metric) →	IHDP ($\sqrt{\epsilon_{PEHE}}$)	
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
CFR_{WASS}	.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.

9.2 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 1). 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.

9.3 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 ϵ_{ATE}. For program depth of 4, we observed a better trade-off between in-sample and out-sample ϵ_{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 ϵ_{ATE}. We believe that this is because of model over-fitting for large program depths (In Figure 2 left, out-sample ϵ_{ATE} is increasing while in-sample ϵ_{ATE} is decreasing). In the Jobs dataset, we observed that almost all program depths results in similar in-sample and out-sample ϵ_{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.

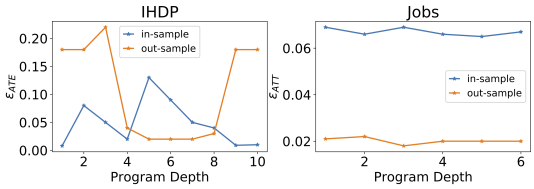

Figure 2: Program depth vs performance.

9.4 Analysis on Twins Dataset

We study the program synthesized for the Twins dataset. NESTER generates simple program ($\texttt{subset}(\mathbf{v}, [0..|\mathbf{v}|])$) for the Twins dataset (Table 4). Since the `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 ϵ_{ATE}. While this is not a surprising conclusion, the choice of the `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.

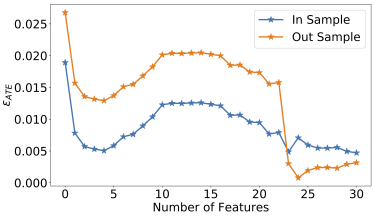

Figure 3: Twins: number of features vs ϵ_{ATE}.

10 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.

Input	Output
William Henry Charles	Charles, W.
Michael Johnson	Johnson, M.
Barack Rogers	Rogers, B.
Martha D. Saunders	Saunders, M.
Peter T Gates	Gates, P.

$$\begin{aligned}
\texttt{String e} &:= \texttt{Concat}(\texttt{f}_1, \ldots, \texttt{f}_\texttt{n}) \\
\texttt{Substring f} &:= \texttt{ConstStr}(\texttt{s}) | \texttt{SubStr}(\texttt{v}, \texttt{p}_1, \texttt{p}_\texttt{r}) \\
\texttt{Position p} &:= (\texttt{r}, \texttt{k}, \texttt{dir}) | \texttt{ConstPos}(\texttt{k}) \\
\texttt{Direction Dir} &:= \texttt{Start} | \texttt{End} \\
\texttt{Regex r} &:= \texttt{s} | \texttt{T}_1 | \ldots | \texttt{T}_\texttt{n}
\end{aligned}$$

Table 8: **Left**: An example FlashFill task where input names are automatically translated to an output format in which last name is followed by the initial of the first name; **Right**: The DSL for FlashFill task based on regular expression string transformations (Parisotto et al., 2016).

Semantics of the above DSL are as follows.

- $\texttt{Concat}(\texttt{f}_1, \ldots, \texttt{f}_\texttt{n})$ - concatenates the results of the expressions $\texttt{f}_1, \ldots, \texttt{f}_\texttt{n}$.
- $\texttt{ConstStr}(\texttt{s})$ - returns the constant string $\texttt{s}$.
- $\texttt{SubStr}(\texttt{v}, \texttt{p}_1, \texttt{p}_\texttt{r})$ - returns substring $\texttt{v}[\texttt{p}_1..\texttt{p}_\texttt{r}]$ of the string $\texttt{v}$, using position logic corresponding to $\texttt{p}_1, \texttt{p}_\texttt{r}$. We denote $\texttt{v}[\texttt{i}..\texttt{j}]$ to denote the substring of string $\texttt{v}$ starting at index $\texttt{i}$ (inclusive) and ending at index $\texttt{j}$ (exclusive), and $\texttt{len}(\texttt{v})$ denotes the length of the string $\texttt{v}$
- $\texttt{ConstPos}(\texttt{k})$ - returns $\texttt{k}$ if $\texttt{k} \geq 0$ else return $\texttt{l} + \texttt{k}$ where $\texttt{l}$ is the length of the string
- $(\texttt{r}, \texttt{k}, \texttt{Start})$ - returns the Start of $\texttt{k}^{\texttt{th}}$ match of the expression $\texttt{r}$ in $\texttt{v}$ from the beginning (if $\texttt{k} \geq 0$) or from the end (if $\texttt{k} < 0$).
- $(\texttt{r}, \texttt{k}, \texttt{End})$ - returns the End of $\texttt{k}^{\texttt{th}}$ match of the expression $\texttt{r}$ in $\texttt{v}$ from the beginning (if $\texttt{k} \geq 0$) or from the end (if $\texttt{k} < 0$).

Based on the above semantics, a program that generates the desired output given the input names in Table 8 is: $\texttt{Concat}(\texttt{f}_1, \texttt{ConstStr}(", "), \texttt{f}_2, \texttt{ConstStr}("."))$ where $\texttt{f}_1 \equiv \texttt{SubStr}(\texttt{v}, (" ", -1, \texttt{End}), \texttt{ConstPos}(-1))$ and $\texttt{f}_2 \equiv \texttt{SubStr}(\texttt{v}, \texttt{ConstPos}(0), \texttt{ConstPos}(1))$.

11 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) $\texttt{if} - \texttt{then} - \texttt{else}$ and (ii) $\texttt{affine}$ program primitives. $\texttt{affine}$ 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 $\texttt{x} = [1, 0]$ then $\texttt{affine}_{[\theta_1, \theta_2; \theta_3]}(\texttt{x}) = \theta_1 \times 1 + \theta_2 \times 0 + \theta_3 = \theta_1 + \theta_3$. The subscripts of $\texttt{affine}$ in $\texttt{affine}_{[\theta_1, \theta_2; \theta_3]}$ contain the parameters θ_1, θ_2 and bias parameter θ_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 illustation purposes. In practice, these weights are learned by training through gradient descent.

12 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, X_1, X_2, Y where: (i) X_1 causes T and Y; (ii) T causes X_2 and Y; and (iii) X_2 causes Y.

$\texttt{if affine}_{[1,1;0]}(\texttt{x}) > 0 \texttt{ then}$ $\quad \texttt{if affine}_{[1,1;-1]}(\texttt{x}) > 0 \texttt{ then}$ $\quad\quad \texttt{affine}_{[0,0;0]}(\texttt{x})$ $\quad \texttt{else}$ $\quad\quad \texttt{affine}_{[1,1;0]}(\texttt{x})$ $\texttt{else}$ $\quad \texttt{affine}_{[0,0;0]}(\texttt{x})$	$\sigma(\beta \times \texttt{affine}_{[1,1;0]}(\texttt{x})) \times$ $(\sigma(\beta \times \texttt{affine}_{[1,1;-1]}(\texttt{x})) \times \texttt{affine}_{[0,0;0]}(\texttt{x}) +$ $(1 - \sigma(\beta \times \texttt{affine}_{[1,1;-1]}(\texttt{x}))) \times \texttt{affine}_{[1,1;0]}(\texttt{x})) +$ $(1 - \sigma(\beta \times \texttt{affine}_{[1,1;0]}(\texttt{x}))) \times \texttt{affine}_{[0,0;0]}(\texttt{x})$

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

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 $\mathbb{E}[Y|do(t)]$ is $\sum_{x_1 \sim X_1} \mathbb{E}[Y|t, x_1]p(x_1)$. 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.

$$p_1 : \texttt{if subset}(\mathbf{v}, [0..1]) \texttt{ then subset}(\mathbf{v}, [0..2]) \texttt{ else subset}(\mathbf{v}, [0..2])$$

$$p_2 : \texttt{if subset}(\mathbf{v}, [0..1]) \texttt{ then subset}(\mathbf{v}, [0..3]) \texttt{ else subset}(\mathbf{v}, [0..3])$$

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.