NEUR2SP: NEURAL TWO-STAGE STOCHASTIC PROGRAMMING

A PREPRINT

Justin Dumouchelle* **Rahul Patel*** **Elias B. Khalil†** **Merve Bodur**
Department of Mechanical & Industrial Engineering, University of Toronto

May 25, 2022

ABSTRACT

Stochastic programming is a powerful modeling framework for decision-making under uncertainty. In this work, we tackle two-stage stochastic programs (2SPs), the most widely applied and studied class of stochastic programming models. Solving 2SPs exactly requires evaluation of an expected value function that is computationally intractable. Additionally, having a mixed-integer linear program (MIP) or a nonlinear program (NLP) in the second stage further aggravates the problem difficulty. In such cases, solving them can be prohibitively expensive even if specialized algorithms that exploit problem structure are employed. Finding high-quality (first-stage) solutions – without leveraging problem structure – can be crucial in such settings. We develop `Neur2SP`, a new method that approximates the expected value function via a neural network to obtain a surrogate model that can be solved more efficiently than the traditional extensive formulation approach. Moreover, `Neur2SP` makes no assumptions about the problem structure, in particular about the second-stage problem, and can be implemented using an off-the-shelf solver and open-source libraries. Our extensive computational experiments on benchmark 2SP datasets from four problem classes with different structures (containing MIP and NLP second-stage problems) show the efficiency (time) and efficacy (solution quality) of `Neur2SP`. Specifically, the proposed method takes less than 1.66 seconds across all problems, achieving high-quality solutions even as the number of scenarios increases, an ideal property that is difficult to have for traditional 2SP solution techniques. Namely, the most generic baseline method typically requires minutes to hours to find solutions of comparable quality.

Keywords Two-stage Stochastic Programming · Supervised Learning · Deep Learning

1 Introduction

Mathematical programming consists of a gamut of tools to solve optimization problems. Under perfect information, i.e., when all the data is deterministic and known, many of these problems can be solved as a linear program (LP) or mixed-integer linear program (MIP). However, in many cases, we need a modeling technique that gives us the flexibility to deal with problems with partial information. Stochastic programming is one such tool that helps us incorporate uncertainty into decision making.

In this work, we focus our attention on two-stage stochastic programs (2SPs). A 2SP involves two sets of decisions, namely the first-stage and second-stage (recourse) decisions, to be made before and after the uncertainty is realized, respectively. Given the (joint) probability distribution of the random parameters of the problem, the most common objective of 2SP is to optimize the expected value of the decisions. For example, in a two-stage stochastic facility location problem, first-stage decisions consist of which facilities should be built whereas second-stage decisions involve assigning customers to open facilities to meet their stochastic demand, and the overall objective is to minimize the sum of cost of the first-stage decisions and the expected cost of the second-stage decisions.

2SPs are usually solved via sample average approximation (SAA), which limits the future uncertainty to a finite set of possible realizations called scenarios. The obtained approximation of a 2SP, referred to as the SAA problem, is a

*These authors contributed equally.

†Corresponding author: `khalil@mie.utoronto.ca`.

reduction to an equivalent deterministic problem, and can be solved by the so-called *extensive form* (EF), the monolithic formulation where scenario copies of the second-stage decision variables are created and linked to the first-stage decisions. However, even for small 2SPs, solving the extensive form may be intractable as it requires introducing a large number of (possibly integer) variables and (possibly nonlinear) constraints. As such, specialized algorithms are required. If the second-stage problem assumes the form of an LP, then algorithms such as Benders decomposition (also known as the L-shaped method) can be leveraged to efficiently solve the problem to optimality. Unfortunately, in many practical applications of 2SP, the second-stage problem assumes the form of a MIP, for which specialized decomposition algorithms might not be efficient. The existence of continuous first-stage variables linked to the second-stage problem significantly increases the difficulty of solving such problems. This is exacerbated when the second-stage problem is nonlinear, for which no general and structure-agnostic solution strategy exists.

In this work, we propose `Neur2SP`, a framework for constructing an *easier-to-solve* surrogate optimization problem for 2SP with the use of neural networks and supervised learning. A neural network is trained to approximate the second-stage value function for a finite set of scenarios. With such a trained network, the surrogate problem is confined to optimizing *only* first-stage decisions with respect to the first-stage objective function and the neural network approximation of the second-stage objective; for a fixed first-stage solution obtained via this surrogate, an optimal second-stage decision can be obtained relatively quickly for each scenario if desired. Our main contributions can be summarized as follows:

1. `Neur2SP` is the first generic machine learning approach for deriving a heuristic solution for 2SP. We introduce a highly parallelizable data collection procedure and show two separate neural models which can be used to formulate a deterministic mixed-integer surrogate problem for 2SP;
2. `Neur2SP` can be used out-of-the-box for 2SPs with linear and nonlinear objectives and constraints as well as mixed-integer variables in both the first and second stages, all without using any problem structure, i.e., in a purely data-driven way;
3. `Neur2SP` is shown to produce high-quality solutions significantly faster than the solely applicable general baseline method, the extensive form approach, for a variety of benchmark problems, namely, stochastic facility location problem, an investment problem, a server location problem, and a pooling problem from chemical engineering.

2 Preliminaries

In this section, we briefly introduce the necessary mathematical concepts used in the remainder of the paper. For a detailed summary of the notation see Appendix A. We first introduce the 2SP setting, then describe the MIP formulation for a $ReLU$ activation function which is central to the surrogate model we propose in this work.

2.1 Two-stage Stochastic Programming

A 2SP can be generally expressed as $\min_x\{c^\intercal x + \mathbb{E}_\xi[Q(x,\xi)] : x \in \mathcal{X}\}$, where $c \in \mathbb{R}^n$ is the first-stage cost vector, $x \in \mathbb{R}^n$ represents the first-stage decisions, $\mathcal{X}$ is the first-stage feasible set, and ξ is the vector of random parameters following a probability distribution $\mathbb{P}$ with support Ξ. The *value function* $Q : \mathcal{X} \times \Xi \to \mathbb{R}$ returns the cost of optimal second-stage (recourse) decisions under realization ξ given the first-stage decisions of x. In many cases, as the $Q(x,\xi)$ is obtained by solving a mathematical program, evaluating the *expected value function*, $\mathbb{E}_\xi[Q(x,\xi)]$ is intractable.

To provide a more tractable formulation, the EF is used. Using a set of K scenarios sampled from the probability distribution $\mathbb{P}$, $\text{EF}(K) \equiv \min_x\{c^\intercal x + \sum_{k=1}^K p_k Q(x,\xi_k) : x \in \mathcal{X}\}$, where p_k is the probability of scenario ξ_k being realized. If $Q(x,\xi) = \min_y\{F(y,\xi) : y \in \mathcal{Y}(x,\xi)\}$, then $\text{EF}(K)$ can be expressed as

$$\min_{x,y}\left\{c^\intercal x + \sum_{k=1}^K p_k F(y_k,\xi_k) : x \in \mathcal{X}, y_k \in \mathcal{Y}(x,\xi_k)\,\forall k = 1,\dots,K\right\},$$

which can be solved through standard deterministic optimization techniques. However, the number of variables and constraints of the EF grows linearly with the number of scenarios. Furthermore, if $Q(\cdot,\cdot)$ is the optimal value of a MIP or an NLP, the EF model becomes significantly more challenging to solve as compared to the LP case, limiting its applicability even at small scale.

2.2 Embedding Neural Networks into MIPs

Neural networks are known to be universal function approximators [Hornik et al., 1989]. Mathematically, an ℓ-layer fully-connected neural network can be expressed as: $h^1 = \sigma(W^0\alpha + b^0)$; $h^{m+1} = \sigma(W^m h^m + b^m), m = 1,\dots,\ell-1$;

$\beta = W^\ell h^\ell + b^\ell$. Let d_0 and d_m represent the input and m^{th} layer dimentionality, respectively. Here, $\alpha \in \mathbb{R}^{d_0}$ is the input, $\beta \in \mathbb{R}$ is the prediction, $h^m \in \mathbb{R}^{d_m}$ is the m-th hidden layer, $W^m \in \mathbb{R}^{d_m \times d_{m+1}}$ is the matrix of weights from layer m to $m+1$, $b^m \in \mathbb{R}^{d_m}$ is the bias at the i-th layer, and σ is a non-linear activation function. The activation function is the rectified linear unit ($ReLU$) given by $\sigma(a) = \max\{0, a\}$.

Central to Neur2SP is the embedding of a trained neural network into a MIP. Here, we present the formulation proposed by Fischetti and Jo [2018]. For a given hidden layer m, the j-th hidden unit, h_j^m, can be written as

$$h_j^m = ReLU\left(\sum_{i=1}^{d_{m-1}} w_{ij}^{m-1} h_i^{m-1} + b_j^{m-1}\right), \tag{1}$$

where w_{ij}^m is the element at the j-th row and i-th column of W^{m-1} and b_j^{m-1} is the j-th index of b^{m-1}. To model $ReLU$ in a MIP for a single hidden unit, say the j-th unit in the m-th layer, we use the variables $\hat{h}_j^m$, $\check{h}_j^m$ and $\hat{h}_i^{m-1}$ for $i = 1, \ldots, d_{m-1}$. With the above variables, the $ReLU$activation can be modeled with the following constraints:

$$\sum_{i=1}^{d_{m-1}} w_{ij}^{m-1} \hat{h}_i^{m-1} + b_j^{m-1} = \hat{h}_j^m - \check{h}_j^m, \tag{2a}$$

$$z_j^m = 1 \Rightarrow \hat{h}_j^m \leq 0, \tag{2b}$$

$$z_j^m = 0 \Rightarrow \check{h}_j^m \leq 0, \tag{2c}$$

$$\hat{h}_j^m, \check{h}_j^m \geq 0, \tag{2d}$$

$$z_j^m \in \{0, 1\}, \tag{2e}$$

where the logical constraints in Equation (2b) and Equation (2c) are translated into big-M constraints by MIP solvers. To verify the correctness of this formulation, observe that constraints (2b) and (2c) in conjunction with the fact the binary z_j^m ensures that at most one of $\hat{h}_j^m$ and $\check{h}_j^m$ are non-zero. Furthermore, since both $\hat{h}_j^m$ and $\check{h}_j^m$ are non-negative, if $\sum_{i=1}^{d_{m-1}} w_{ij}^{m-1} \hat{h}_i^{m-1} + b_j^{m-1} > 0$, then it follows that $\hat{h}_j^m > 0$ and $\check{h}_j^m = 0$. If negative, then $\hat{h}_j^m = 0$ and $\check{h}_j^m > 0$. Thus, we have that if the left-hand side of (2a) is positive, $\hat{h}_j^m$ will be positive; if it is negative, then $\hat{h}_j^m = 0$; this is an exact representation of the $ReLU$function.

3 Related Work

A recent line of work by Cheng et al. [2017], Tjeng et al. [2017], Fischetti and Jo [2018], Serra et al. [2018] studies the problem of embedding a trained neural network with ReLU activation in the hidden layers into a MIP. To the best of our knowledge, Neur2SP is the first approach that employs this technique in the stochastic programming setting. The works of Anderson et al. [2020] and Grimstad and Andersson [2019] showcase how the bound of these MIP encodings of a neural network can be tightened further; as the bounds are based on the big-M constraints, setting the right value of M can have an impact on the solution time.

Another exciting direction of research proposes to heuristically solve 2SP based on scenario reduction. Many of these approaches Dupačová et al. [2003], Römisch [2009], Beraldi and Bruni [2014], Prochazka and Wallace [2020], Keutchayan et al. [2021] perform some form of clustering to reduce the number of scenarios and then solve a smaller surrogate problem with these scenarios. Recently, Wu et al. [2022] used a conditional variational autoencoder to learn scenario embedding and perform clustering on it for scenario reduction. To find representative scenarios they use semi-supervised learning with respect to the second-stage cost, however, these predictions are not leveraged explicitly in the downstream task, like the methodology we propose in this work.

Nair et al. [2018] directly learn to predict the solution of a 2SP for unconstrained binary first-stage decisions. They predict an initial first-stage solution and iteratively perturb it to improve. However, it cannot handle hard constraints nor general integer or continuous variables in the first stage. Bengio et al. [2020] predicts a representative scenario, for an input scenario set, and use it to form a smaller surrogate problem. They show that using the predicted representative scenario, a near-optimal first-stage decision can be obtained by solving the surrogate. However, their method relies on the problem structure to build the representative scenario for training, which requires domain expertise.

Furthermore, Dai et al. [2022] develop a model to map problem instances to piece-wise linear value functions. Their approach is applied to multi-stage problems, however, as their model is based on piece-wise linear components, it is only applicable to problems with linear next-stage sub-problems. Also, none of the above mentioned works have a non-linear second-stage problem.

4 Methodology

In this section, we present our methodology. Specifically, the learning models and tasks, the approximations to 2SP, and a data generation strategy. For an overview of our methodology, we provide a illustration and summary in Figure 1. We use the notation $[n] \coloneqq \{1, \dots, n\}$ for $n \in \mathbb{Z}_+$.

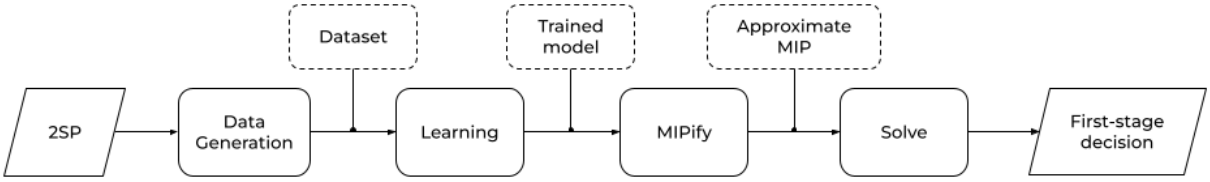

Figure 1: Overview of the methodology. The leftmost block is the input, namely, a 2SP. From the 2SP, we follow the data generation procedure from Section 4.3 to obtain a dataset. We then train one of the learning models presented in Section 4.1. The trained model is then embedded into a MIP using the procedure in Section 4.2 to obtain an approximate MIP (the "MIPify" step). Lastly, the approximate MIP is solved with an off-the-shelf MIP solver to obtain a first-stage 2SP solution.

4.1 Learning Models

We propose two separate learning models for predicting the second-stage costs. These models loosely relate to the multi-cut and single-cut Benders decomposition algorithms. Specifically, we propose a single-cut model which attempts to estimate the expected second-stage cost of *a set of scenarios* and a multi-cut model which estimates the cost of *a single second-stage scenario*.

Single-cut. In this case, the model learns a mapping from $\left(x, \{\xi_k\}_{k=1}^K\right) \to \sum_{k=1}^K p_k Q(x, \xi_k)$. In words, the model takes in a first-stage solution x and a set of K scenarios, and outputs a prediction of the second-stage cost. We embed the scenario set $\{\xi_k\}_{k=1}^K$ into a latent space by passing it through a neural network Ψ^1 and performing mean-aggregation. The aggregated embedding is passed through another network, Ψ^2, to obtain the final scenario embedding. This embedding, representing the scenario set [Zaheer et al., 2017], is appended to the input first-stage decision and passed through a feed-forward network Φ^{SC}, with $ReLU$activations, to predict the expected second-stage value. Hence, the final output is such that $\Phi^{SC}(x, \Psi^2(\oplus_{k=1}^K \Psi^1(p_k, \xi_k))) \approx \sum_{k=1}^K p_k Q(x, \xi_k)$. Note that the embedding networks, Ψ^1 and Ψ^2, can be arbitrarily complex as only Φ^{SC} is embedded into the approximate MIP. Also, although Ψ^1 is trained using K scenarios, once the networks are trained, they can be used with any finite (potentially much larger) number of scenarios.

Multi-cut. In this setting, the model learns a mapping Φ^{MC} from $(x, \xi_k) \to Q(x, \xi_k)$, where $k \in [K]$. Once the mapping Φ^{MC} is learned, we can approximate $\sum_{k=1}^K p_k Q(x, \xi_k) \approx \sum_{k=1}^K p_k \Phi^{MC}(x, \xi_k)$.

4.2 Neural Network Embedding for 2SP

We now describe the approximate MIP for both the single-cut and multi-cut learning models from the preceding section. Let Λ represent the number of predictions made by the neural network. For the single-cut case, $\Lambda = 1$ as we only predict the expected second-stage value for a set of scenarios. In the multi-cut case, $\Lambda = K$ as we predict the second-stage value for each scenario.

Let $\hat{h}_j^{m,\lambda}$ represent the $ReLU$output for the j-th hidden unit in the m-th hidden layer for output λ, for all $m \in [\ell - 1]$, $j \in [d_m]$, and $\lambda \in [\Lambda]$. Suppose $\check{h}_j^{m,\lambda}$ is a slack variable used to model the $ReLU$ output for the j-th hidden unit in the m-th hidden layer for scenario k, for all $m \in [\ell - 1]$, $j \in [d_m]$, and $\lambda \in [\Lambda]$. Let $z_j^{m,\lambda}$ be a binary variable used to ensure that at most one of $\hat{h}_j^{m,k}$ and $\check{h}_j^{m,k}$ are non-zero. This variable is defined for all $m \in [\ell - 1]$, $j \in [d_m]$, and $\lambda \in [\Lambda]$. Suppose β_λ is the λ-th prediction by the neural network, for all $\lambda \in [\Lambda]$.

With the above variables we can define an approximation to EF as given in Equation (3). The objective function (3a) minimizes the cost of the first-stage decisions plus the expected cost of the second-stage cost approximation. Constraints

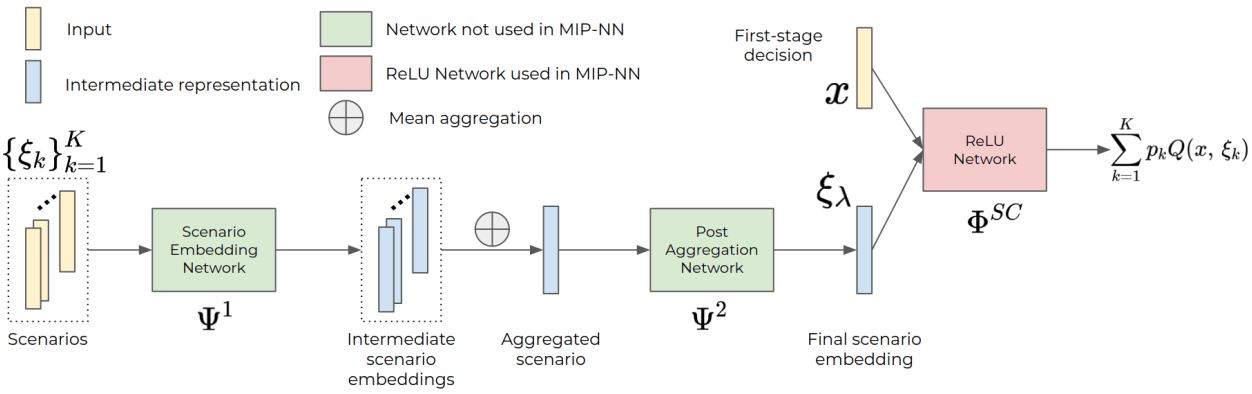

Figure 2: Neur2SP single-cut architecture.

(3b)-(3d) propagate a first-stage solution x to the output of the neural network for each scenario. Constraints (3e)-(3h) ensure the prediction of the neural network is respected. Constraint (3i) ensures the feasibility of the first-stage solution.

In this approximation, we introduce a number of additional variables and big-M constraints. Specifically, for a neural network with H hidden units, we introduce $\Lambda \cdot H$ additional binary variables for $z_j^{m,\lambda}$. In addition, we introduce $2 \cdot \Lambda \cdot H$ continuous variables for $\hat{h}_j^{m,\lambda}$ and $\check{h}_j^{m,\lambda}$. Lastly, we require an additional Λ variables for the output of the network. Although the number of variables we introduce in this approximation is quite large, we hypothesize that the resulting MIP will be easier to solve than the extensive form, in particular, when the second-stage problem is nonlinear. In the remainder of the paper, we refer to the neural network embedded MIP given in (3) as MIP-NN.

$$\min \quad c^\intercal x + \sum_{\lambda=1}^{\Lambda} p_\lambda \beta_\lambda \tag{3a}$$

$$\text{s.t.} \quad \sum_{i=1}^{d_0} w_{ij}^0 [x, \xi_\lambda]_i + b_j^0 = \hat{h}_j^{1,\lambda} - \check{h}_j^{1,\lambda} \qquad \forall\, j \in [d_1], \lambda \in [\Lambda], \tag{3b}$$

$$\sum_{i=1}^{d_{m-1}} w_{ij}^{m-1} \hat{h}_i^{m-1,\lambda} + b_j^{m-1} = \hat{h}_j^{m,\lambda} - \check{h}_j^{m,\lambda} \qquad \forall\, m \in [\ell - 1], j \in [d_m], \lambda \in [\Lambda], \tag{3c}$$

$$\sum_{i=1}^{d_\ell} w_{ij}^\ell \hat{h}_i^{\ell,\lambda} + b^\ell \leq \beta_\lambda \qquad \forall \lambda \in [\Lambda], \tag{3d}$$

$$z_j^{m,\lambda} = 1 \Rightarrow \hat{h}_j^{m,\lambda} = 0 \qquad \forall\, m \in [\ell - 1], j \in [d_m], \lambda \in [\Lambda], \tag{3e}$$

$$z_j^{m,\lambda} = 0 \Rightarrow \check{h}_j^{m,\lambda} = 0 \qquad \forall\, m \in [\ell - 1], j \in [d_m], \lambda \in [\Lambda], \tag{3f}$$

$$z_j^{m,\lambda} \in \{0, 1\} \qquad \forall\, m \in [\ell - 1], j \in [d_m], \lambda \in [\Lambda], \tag{3g}$$

$$\hat{h}_j^{m,\lambda}, \check{h}_j^{m,\lambda} \geq 0 \qquad \forall\, m \in [\ell - 1], j \in [d_m], \lambda \in [\Lambda], \tag{3h}$$

$$x \in \mathcal{X} \tag{3i}$$

4.3 Data Generation

The proposed methodology falls in the supervised learning paradigm. Hence, we need a diverse dataset of input-output pairs to train a model. To generate such dataset for a given 2SP problem we follow an iterative procedure. We begin by generating a random feasible first-stage decision. For the single-cut case, we sample a set of scenarios with random cardinality $K^{'}$ from the uncertainty distribution. Here, $K^{'}$ should be chosen to balance the trade-off between the time spent to generate a sample of second-stage values for a given first-stage solution and the time to estimate the expected

Problem	First stage	Second Stage	Objective	Constraints	Objective Sense
CFLP	Binary	Binary	Linear	Linear	Minimization
INVP	Continuous	Binary	Linear	Linear	Minimization
SSLP	Binary	Binary	Linear	Linear	Minimization
PP	Binary	Continuous	Bilinear	Bilinear	Maximization

Table 1: Problem class characteristics.

second-stage value for a set of first-stage decisions in a given time budget. Specifically, if $K^{'}$ is large, then on average more time will be spent in determining the expected value using a large number of scenarios, while for a small $K^{'}$, the first-stage decision space will be explored more since expected value estimates would be obtained faster. For a given input of a first-stage decision and set of scenarios we then generate a label by calculating the expected second-stage value $\sum_{k'=1}^{K^{'}} p_{k'} Q_{k'}(\cdot, \xi_{k'})$.

For the multi-cut case, at each iteration of the data generation procedure, we sample a single scenario from the uncertainty distribution. For a given input of a first-stage decision and scenario we generate a label by calculating its second-stage value $Q(\cdot, \cdot)$. Last, the input-output pair is added to the dataset.

This data generation procedure is fully parallelizable over the second-stage problems to be solved.

4.4 Multi-cut vs. Single-cut Trade-offs

The single-cut and multi-cut models both have significant trade-offs in terms of the learning task and the downstream optimization problem which embeds the trained model.

Learning. In data generation, both models require solving second-stage problems with a fixed first-stage solution. A sample in the multi-cut setting requires solving only a single optimization problem, whereas a sample in the single-cut setting requires solving at most $K^{'}$ second-stage problems. As this process is offline and highly parallelizable, this trade-off is easy to mitigate. As for learning, the single-cut approach operates on a subset of scenarios which makes for an exponentially larger input space. Despite the large input space, our experiments show that the single-cut model in the training converges quite well and in many cases the embedded model outperforms the multi-cut model.

Downstream Optimization Task. As the ultimate goal is embedding the trained model into a MIP, the trade-off in this regard becomes quite important. Specifically, for K scenarios, the multi-cut model will have K times more binary and continous variables than the single-cut model. For problems with a large number of scenarios, this makes the single-cut model much more appealing, smaller and likely faster to solve. Furthermore, it allows for much larger networks given that only a single copy of the network is embedded.

5 Computational Setup

All experiments were run on a computing cluster with an Intel Xeon CPU E5-2683 and Nvidia Tesla P100 GPU with 64GB of RAM (for training). Gurobi 9.1.2 [Gurobi Optimization, LLC, 2021] was used as the MIP solver. Scikit-learn 1.0.1 [Pedregosa et al., 2011] and Pytorch 1.10.0 [Paszke et al., 2019] were used for supervised learning.

2SP Problems. We evaluate our approach on four different 2SP problems that are commonly considered in the literature to test 2SP methodologies, namely, a two-stage stochastic variant of the capacitated facility location problem (CFLP) from [Cornuéjols et al., 1991], investment problem (INVP) [Schultz et al., 1998], stochastic server location problem (SSLP) [Ntaimo and Sen, 2005], and pooling problem (PP) [Audet et al., 2004, Gupte et al., 2017, Gounaris et al., 2009, Haverly, 1978]. The type of the first-stage and second-stage variables for these problems is provided in Table 1. The detailed descriptions of the problems are provided in Appendix B.

Baselines. We consider two baselines. The first is EF, which is perhaps the only generic approach that can be applied within both integer and nonlinear second-stage problems. We limit the solving time of EF to 3 hours. Additionally, we compare against an embedded trained linear regressor rather than a neural network, but defer these results to Appendix D as the solution quality is quite poor in comparison to the neural network models.

Model Selection. As with any supervised learning setting, model selection can have a significant impact on performance. As a brief summary, we use random search over 100 configurations for model selection, and observe that the overall

supervised learning performance is not greatly impacted by different configurations. We report detailed experiments for model selection in Appendix E.

Dataset Sizing. In this domain, we have control over the number of samples that we collect. However, there is a cost (time to solve the second-stage problem) associated with labelling any sample. As such, collecting just enough samples to achieve low regression error must be determined empirically. To that end, we report results for varying dataset sizes for CFLP_10_10. We use a validation set with 5000 samples and training sets with 100, 500, 1000, 5000, 10000, and 20000 samples. Model selection with random search is done for each training set size as described in the previous paragraph. Figures 3 and 4 report the results for the multi-cut and single-cut models respectively. In both cases, we can see an improvement in validation MAE with increases in the dataset sizes, however, diminishing returns start to occur when increasing the number of samples above 5000 samples.

This motivates the choice of dataset sizes which we use in the remainder of the experiments, not only for CFLP but also for the three other problem classes. Specifically, we use 10000 samples for the multi-cut case as data generation is quite fast. For the single-cut case we limit the number of samples to 5000 as we only see a small improvement of %4 in validation MAE at the cost of doubling the compute time.

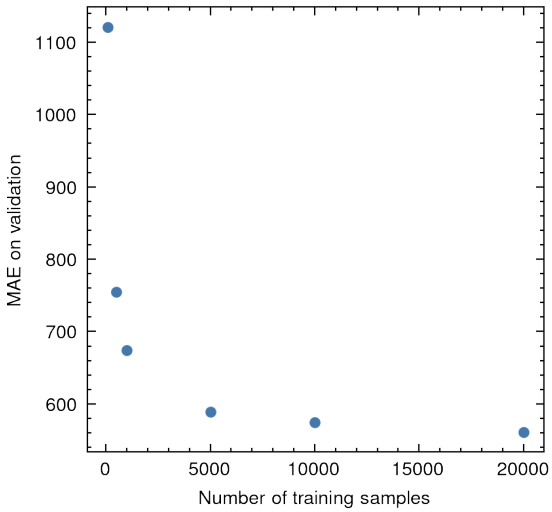

Figure 3: MC dataset sizing results

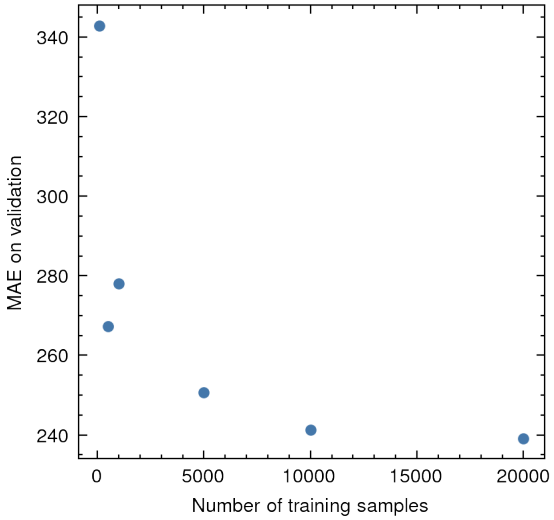

Figure 4: SC dataset sizing results

Data Generation & Supervised training times : As the data generation and training can be done offline and are both parallelizable, we defer specific timings to Appendix C. We note that for data generation, a single sample can be obtained in less than two seconds for all instances, and in many cases much faster. There training times are within the range of 120 to 2100 seconds. For the single-cut data generation, we choose $K^{'} = 100$, a number of scenarios which was quick to label while exposing the model to a reasonably large number of scenarios in some cases. The combined time for data generation and model training is typically always less than 3 hours (i.e., the time given to EF) and depending on the problem, may be much less.

6 Results & Discussion

In this section, we report the results of the embedded model across the four problem settings. As is standard in 2SP, we evaluate a single “base" instance across varying scenarios sets and sizes. For example, in CFLP, and for a “base" instance with 10 facilities and 10 customers, one can generate an instance by specifying the number of scenarios and then sampling as many. An important advantage of our approach is that we can apply a single trained model to an instance with an arbitrary number of scenarios with no additional offline time required for training or data generation. For example, the same trained model is used for CFLP_10_10_{100,500,1000}. We abbreviate the single-cut and multi-cut models to SC and MC, respectively.

Tables 2 through 5 report the gaps between approaches, solving times, and the time which EF takes to achieve the same solution quality as SC and MC. In addition, we include supplementary results with the objective values in Appendix D. For SSLP and CFLP, each row represents mean statistics across 11 and 10 instances generated by sampling different

Problem	Obj. Difference (%)		Solving Time			EF time to			
	EF-SC	EF-MC	SC	MC	EF	SC		MC	
CFLP_10_10_100	2.58	1.65	**0.38**	8.28	4,410.60	8.87	(0)	12.69	(0)
CFLP_10_10_500	2.41	0.94	**0.60**	206.30	10,800.17	415.89	(0)	2,034.73	(0)
CFLP_10_10_1000	0.94	-0.67	**0.64**	856.77	10,800.87	580.50	(0)	7,551.00	(8)
CFLP_25_25_100	-0.75	-0.75	**0.44**	4.86	10,800.06	-	(10)	-	(10)
CFLP_25_25_500	-3.62	-3.62	**0.54**	26.41	10,800.14	-	(10)	-	(10)
CFLP_25_25_1000	-1.32	-1.32	**0.58**	54.45	10,800.36	-	(10)	-	(10)
CFLP_50_50_100	-0.43	-1.29	**1.66**	21.10	10,800.05	5,637.98	(6)	2,334.04	(9)
CFLP_50_50_500	-9.58	-10.71	**1.25**	173.63	10,806.15	-	(10)	-	(10)
CFLP_50_50_1000	-16.62	-17.50	**1.44**	572.12	10,805.82	-	(10)	-	(10)

Table 2: CFLP results: each row represents an average over ten 2SP instances with varying scenario sets. "Obj. Difference" for method EF-{SC,MC} is the percent relative objective value of {SC,MC} to EF; a negative (positive) value of $-g\%$ ($g\%$) indicates that {SC,MC} finds a solution that is $g\%$ better (worse) than EF's for the minimization problem. "Solving Time" is the time in which {SC,MC,EF} are solved to optimality. A time of $\sim$10,800 implies that the solving limit was reached. "EF time to" is the time in which EF achieves a solution of the same quality as {SC,MC}. To the right in parentheses is the number of instances for which EF failed to find a solution that is as good as {SC,MC}. If EF did not find any feasible solution, then the entry is left as "-". All times in seconds.

Problem	Obj. Difference (%)		Solving Time			EF time to			
	EF-SC	EF-MC	SC	MC	EF	SC		MC	
SSLP_10_50_50	0.00	0.00	**0.11**	5.83	10,800.48	228.06	(0)	228.06	(0)
SSLP_10_50_100	-0.00	-0.00	**0.11**	13.09	10,800.21	145.35	(0)	145.35	(0)
SSLP_10_50_500	-0.00	-0.00	**0.14**	129.44	10,802.82	7,359.85	(4)	7,359.85	(4)
SSLP_10_50_1000	-55.21	-55.21	**0.13**	466.38	10,800.47	-	(11)	-	(11)
SSLP_10_50_2000	-102.69	-102.69	**0.14**	2,182.31	10,800.17	-	(11)	-	(11)
SSLP_15_45_5	3.10	18.71	**0.32**	0.34	2.54	0.75	(0)	0.12	(0)
SSLP_15_45_10	2.98	18.47	**0.31**	0.58	1,976.62	2.72	(0)	0.20	(0)
SSLP_15_45_15	2.53	18.90	**0.33**	0.86	2,052.76	1.84	(0)	0.34	(0)
SSLP_5_25_50	0.12	1.78	**0.20**	1.14	2.24	1.94	(0)	1.97	(0)
SSLP_5_25_100	0.02	1.60	**0.18**	1.83	8.43	8.04	(0)	7.75	(1)

Table 3: SSLP results: each row represents an average over eleven 2SP instances with varying scenario sets, one of which being the instance from Ahmed et al. [2015]. Columns are as in Table 2.

scenario sets of a given size, respectively. However, for INVP and PP, each row represents the statistics across 1 instance. Originally, both these problems have infinite support as the uncertainty distributions are assumed to be continuous. To manage the complexity, these distributions are transformed to have a finite support by uniformly sampling equidistant points over the continuous domain in the literature. We adopt the same procedure, leading to a static set of scenarios for a given scenario set size.

In Table 6, we report optimality gaps and solving times on the publicly available SSLP SIPLib instances. In contrast with the other instances in this paper, this set of 12 instances is widely used as a benchmark in the stochastic programming community since they were proposed in Ntaimo and Sen [2005] The optimal values for these instances were obtained from the SIPLIB Ahmed et al. [2015]. The results for SC and MC, in Table 6, are directly comparable to those of other methods reported in the literature.

6.1 Discussion

Tables 2–5 show that SC is significantly faster than other approaches, with a minimum and maximum solving time of 0.11s and 1.66s respectively, across all problems. This highlights the scalability of the SC in terms of problem size and type, which is expected as the size of the resulting MIP is independent of the number of scenarios. Also, the objective difference is less than 5% in most cases, with a minimum of -102% and a maximum of 13.78%. These differences are inversely proportional to the scenario set size, which indicates that the SC is able to generalize on larger scenario sets, even though it was trained with a maximum of 100 scenarios per data point. EF takes significantly higher time to reach a solution quality similar to SC, often on the order of minutes to 3 hours. For many larger CFLP and SSLP instances, EF finds worse solutions than SC even after 3 hours. Not only is SC as good or better in solution quality, but also orders of magnitude faster.

Problem	Obj. Difference (%)		Solving Time			EF time to	
	EF-SC	EF-MC	SC	MC	EF	SC	MC
INVP_B_E_4	9.54	3.01	0.36	0.34	**0.02**	0.02	0.02
INVP_B_E_9	7.54	2.00	0.31	0.53	**0.04**	0.03	0.03
INVP_B_E_36	2.72	4.96	0.30	9.53	**0.08**	0.02	0.02
INVP_B_E_121	1.37	2.42	**0.33**	86.42	1.69	0.06	0.02
INVP_B_E_441	2.80	2.43	**0.37**	4,342.19	117.59	0.78	1.15
INVP_B_E_1681	1.36	-	**0.34**	-	10,800.01	17.41	0.00
INVP_B_E_10000	-1.48	-	**0.36**	-	10,803.98	-	0.00
INVP_B_H_4	8.81	9.50	0.46	0.25	**0.01**	0.01	0.01
INVP_B_H_9	5.04	5.04	0.30	0.57	**0.03**	0.02	0.02
INVP_B_H_36	1.61	1.61	**0.26**	6.79	1.29	0.01	0.01
INVP_B_H_121	1.77	1.77	**0.33**	45.89	34.69	0.01	0.01
INVP_B_H_441	2.13	5.50	**0.28**	1,870.42	217.46	2.21	0.21
INVP_B_H_1681	-0.71	-	**0.36**	-	10,800.01	-	0.00
INVP_B_H_10000	-2.72	-	**0.36**	-	10,800.03	-	0.00
INVP_I_E_4	12.83	0.00	0.38	0.23	**0.01**	0.01	0.01
INVP_I_E_9	7.40	2.64	0.27	0.35	**0.06**	0.01	0.02
INVP_I_E_36	5.48	5.17	0.27	1.39	**0.04**	0.01	0.01
INVP_I_E_121	5.30	4.49	**0.29**	49.51	1.65	0.02	0.03
INVP_I_E_441	3.00	0.68	**0.26**	2,049.93	46.92	0.08	0.10
INVP_I_E_1681	1.31	3.08	**0.26**	10,834.53	10,800.00	0.41	0.41
INVP_I_E_10000	-1.35	-	**0.30**	-	10,800.10	-	0.00
INVP_I_H_4	13.78	12.16	0.35	0.21	**0.02**	0.01	0.01
INVP_I_H_9	9.12	0.81	0.37	0.31	**0.03**	0.01	0.02
INVP_I_H_36	4.97	3.44	**0.36**	1.99	1.27	0.03	0.03
INVP_I_H_121	4.01	4.99	**0.32**	23.10	7.43	0.07	0.07
INVP_I_H_441	3.15	3.15	**0.32**	1,231.48	10,800.00	0.33	0.33
INVP_I_H_1681	-0.34	0.11	**0.33**	10,816.89	10,800.03	-	252.70
INVP_I_H_10000	-1.60	-	**0.38**	-	10,802.10	-	0.00

Table 4: INVP results: each row represents a single instances. See Table 2 for description of columns.

Problem	Obj. Difference (%)		Solving Time			EF time to	
	EF-SC	EF-MC	SC	MC	EF	SC	MC
PP_125	3.25	36.64	**1.51**	144.08	10,800.00	10,717.05	2.48
PP_216	9.06	40.14	**1.47**	254.94	364.98	59.79	3.59
PP_343	0.67	40.85	**1.46**	570.64	10,800.00	1,450.54	9.12
PP_512	8.69	39.77	**1.60**	1,200.37	10,800.01	167.77	13.80
PP_729	1.38	37.98	**1.62**	3,440.19	10,800.01	5,867.67	36.34
PP_1000	5.92	41.32	**1.49**	10,853.59	10,800.00	1,596.22	210.48

Table 5: PP results: each row represents a single instances. See Table 2 for description of columns.

For the MC, we can observe that the solving time is directly proportional to the size of the problem. However, for the largest INVP instances, it times out without even generating a feasible solution (Table 4). This is expected as we need to embed the trained neural network once per scenario, limiting scalability. In terms of objective differences, for CFLP and SSLP, we can observe that the difference improves with the increase in instance size. For INVP, no clear trend is visible, however, they do not exceed objective difference of 5% in most cases. For PP, the objective difference is around 40%, indicating that the MC is not able to generalize, whereas SC performs very well.

As for the results in Table 6, we can see that both SC and MC do quite well in terms of finding solutions, especially in the larger scenario case where they obtain optimal first-stage solutions. Perhaps, the most impressive results here is that SC is able to obtain optimal results for many instances in $\sim$0.1 seconds.

Problem	Gap to Optimal (%)			Solving Time		
	SC	MC	EF	SC	MC	EF
SSLP_10_50_50	**0.00**	**0.00**	**0.00**	**0.11**	4.83	10,801.27
SSLP_10_50_100	**0.00**	**0.00**	**0.00**	**0.11**	11.66	10,800.04
SSLP_10_50_500	**0.00**	**0.00**	0.00	**0.11**	107.88	10,818.23
SSLP_10_50_1000	**0.00**	**0.00**	28.64	**0.12**	383.51	10,800.26
SSLP_10_50_2000	**0.00**	**0.00**	51.24	**0.13**	4,523.19	10,800.20
SSLP_15_45_5	0.46	19.59	**0.00**	0.32	**0.28**	4.17
SSLP_15_45_10	1.57	18.23	**0.00**	**0.25**	0.40	3.71
SSLP_15_45_15	0.53	16.51	**0.00**	**0.41**	0.72	4.74
SSLP_5_25_50	**0.00**	2.15	**0.00**	**0.26**	0.92	2.35
SSLP_5_25_100	**0.00**	1.40	**0.00**	**0.18**	1.68	8.87

Table 6: SSLP SIPLib gap and time comparison among methods. Optimal SIPLib instances values obtained from Ahmed et al. [2015]. “Gap to Optimal" is the percent gap to the optimal solution. “Solving Time" is the solving to of the approximate MIP and EF. All times in seconds.

7 Conclusion

Two-stage stochastic programming is a powerful modeling framework for decision-making under uncertainty. These problems are hard to solve in practice, especially when the second-stage problem is a MIP or NLP. Finding good feasible solutions quickly thus becomes of extremely important.

To that end, we proposed Neur2SP, a learning-based, general, and structure-agnostic approach which approximates the second-stage value function to form an easy-to-solve surrogate problem. The four problem classes we have tackled are (1) widely used in the literature, (2) vary in the types of first and second-stage problems, and (3) span a wide range in terms of number of variables, constraints, and scenarios. Through our experiments, we show that Neur2SP achieves high-quality solutions quickly, especially for larger instances. In 1–2 seconds, Neur2SP can find solutions of the same or better quality than the most generic method in the literature, EF, with the latter taking minutes to hours.

In terms of future work, this methodology can be extended in many directions. Further innovations in the SC model architecture may improve our already positive results. Another direction is the extension of the general idea of embedding trained models into other complex optimization problems, such as bi-level optimization or multi-stage stochastic programming. Lastly, in this work, we propose MC and SC, however, a natural middle ground between these models is a clustering approach which embeds a trained model for a subset of representative scenarios, rather than a single or the entire scenario set at evaluation time.

References

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

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309, 2018.

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 251–268. Springer, 2017.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*, pages 4558–4566. PMLR, 2018.

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, pages 1–37, 2020.

Bjarne Grimstad and Henrik Andersson. Relu networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131:106580, 2019.

Jitka Dupačová, Nicole Gröwe-Kuska, and Werner Römisch. Scenario reduction in stochastic programming. *Mathematical programming*, 95(3):493–511, 2003.

Werner Römisch. Scenario reduction techniques in stochastic programming. In *International Symposium on Stochastic Algorithms*, pages 1–14. Springer, 2009.

Patrizia Beraldi and Maria Elena Bruni. A clustering approach for scenario tree reduction: an application to a stochastic programming portfolio optimization problem. *Top*, 22(3):934–949, 2014.

Vit Prochazka and Stein W Wallace. Scenario tree construction driven by heuristic solutions of the optimization problem. *Computational Management Science*, 17(2):277–307, 2020.

Julien Keutchayan, Janosch Ortmann, and Walter Rei. Problem-driven scenario clustering in stochastic optimization. *arXiv preprint arXiv:2106.11717*, 2021.

Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning scenario representation for solving two-stage stochastic integer programs. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=06Wy2BtxXrz`.

Vinod Nair, Dj Dvijotham, Iain Dunning, and Oriol Vinyals. Learning fast optimizers for contextual stochastic integer programs. In *UAI*, pages 591–600, 2018.

Yoshua Bengio, Emma Frejinger, Andrea Lodi, Rahul Patel, and Sriram Sankaranarayanan. A learning-based algorithm to quickly compute good primal solutions for stochastic integer programs. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 99–111. Springer, 2020.

Hanjun Dai, Yuan Xue, Zia Syed, Dale Schuurmans, and Bo Dai. Neural stochastic dual dynamic programming. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=aisKPsMM3fg`.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3391–3401, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html`.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. https://www.gurobi.com.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Gérard Cornuéjols, Ranjani Sridharan, and Jean-Michel Thizy. A comparison of heuristics and relaxations for the capacitated plant location problem. *European journal of operational research*, 50(3):280–297, 1991.

Rüdiger Schultz, Leen Stougie, and Maarten H Van Der Vlerk. Solving stochastic programs with integer recourse by enumeration: A framework using gröbner basis. *Mathematical Programming*, 83(1):229–252, 1998.

Lewis Ntaimo and Suvrajeet Sen. The million-variable "march" for stochastic combinatorial optimization. *Journal of Global Optimization*, 32(3):385–400, 2005.

Charles Audet, Jack Brimberg, Pierre Hansen, Sébastien Le Digabel, and Nenad Mladenović. Pooling problem: Alternate formulations and solution methods. *Management science*, 50(6):761–776, 2004.

Akshay Gupte, Shabbir Ahmed, Santanu S Dey, and Myun Seok Cheon. Relaxations and discretizations for the pooling problem. *Journal of Global Optimization*, 67(3):631–669, 2017.

Chrysanthos E Gounaris, Ruth Misener, and Christodoulos A Floudas. Computational comparison of piecewise- linear relaxations for pooling problems. *Industrial & Engineering Chemistry Research*, 48(12):5742–5766, 2009.

Co A Haverly. Studies of the behavior of recursion for the pooling problem. *Acm sigmap bulletin*, (25):19–28, 1978.

Shabbir Ahmed, R Garcia, N Kong, L Ntaimo, G Parija, F Qiu, and S Sen. Siplib: A stochastic integer programming test problem library. *See http://www2. isye. gatech. edu/˜ sahmed/siplib*, 2015.

Niels van der Laan and Ward Romeijnders. A loose benders decomposition algorithm for approximating two-stage mixed-integer recourse models. *Mathematical Programming*, 190(1):761–794, 2021.

Xiang Li, Emre Armagan, Asgeir Tomasgard, and Paul I Barton. Stochastic pooling problem for natural gas production network design and operation under uncertainty. *AIChE Journal*, 57(8):2120–2135, 2011.

A Symbols

List of symbols used in the paper with their brief description.

Two-stage stochastic program	
x	First-stage decision vector
c	First-stage objective coefficient vector
K	EF scenario set size
k	Scenario index
ξ_k	k^{th} scenario realization
p_k	Probability of scenario k
n	Dimension of x
$Q(x, \xi)$	Second-stage sub-problem for first-stage decision x and scenario ξ
$F(x, \xi)$	Second-stage cost for first-stage decision x and scenario ξ
$\mathcal{X}$	Constraint set exclusively on the first-stage decision
$\mathcal{Y}(x, \xi)$	Scenario-specific constraint set for first-stage decision x and scenario ξ
Neural Network	
ℓ	Number of layers in the network
m	Index over the neural network layers
d^0	Dimensionality of input layer
d^m	Dimensionality of layer m
α	Input to the neural network
β	Output of the neural network
W	Weight matrix
b	Bias
σ	Activation function
h^m	m^{th} hidden layer
i	Index over the column of weight matrix
j	Index over the row of weight matrix
Φ^1	Scenario-encoding network
Φ^2	Post scenario-aggregation network
Ψ^{SC}	Scenario-embedding network for single-cut
Ψ^{MC}	Scenario-embedding network for multi-cut
MIP-NN	
$\hat{h}$	Non-negative $ReLU$ input
$\check{h}$	Negative $ReLU$ input
z	Indicator variables
Λ	Number of predictions used in embedding

Table 7: Symbols summary

B Stochastic Programming Problems

B.1 Capacitated Facility Location (CFLP)

The CFLP is a decision-making problem in which a set of facility opening decisions must be made in order to meet the demand of a set of customers. Typically this is formulated as a minimization problem, where the amount of customer demand satisfied by each facility cannot exceed its capacity. The two-stage stochastic CFLP arises when facility opening decisions must be made prior to knowing the actual demand. For this problem, we generate instances following the procedure described in [Cornuéjols et al., 1991] and create a stochastic variant by simply generating the first-stage costs and capacities, then generate scenarios by sampling K demand vectors using the distributions defined in [Cornuéjols et al., 1991]). To ensure relatively complete recourse, we introduce additional variables with prohibitively expensive objective costs in the case where customers cannot be served. In the experiments a CFLP with n facilities, m customers, and s scenarios is denoted by CFLP_n_m_s.

B.2 Investment Problem (INVP)

The INVP is a 2SP problem studied in [Schultz et al., 1998]. This 2SP has a set of continuous first-stage decisions which yield an immediate revenue. In the second stage, after a set of random variables are realized, a set of binary decisions can be made to receive further profit. In this work, we specifically consider the instance described in the example 7.3. of [Schultz et al., 1998]. This problem has 2 continuous variables in the first stage with the domain $[0, 5]$, and 4 binary variables in the second stage. The scenarios are given by two random discrete variables which are defined with equal probability over the range $[5, 15]$. Specifically, for K scenarios, each random variable can take an equally spaced value in the range. Although the number of variables is quite small, the presence of continuous first-stage decision has made this problem relevant within the context of other recent work such as the heuristic approach proposed in [van der Laan and Romeijnders, 2021]. As a note, we reformulate the INVP as an equivalent minimization problem in the remainder of this work. In the experiments an INVP instance is denoted by INVP_v_t, where v indicates the type of second-stage variable (B for binary and I for integer) and t indicates the type of technology matrix (E for identity and H for $[[2/3, 1/3], [1/3, 2/3]]$).

B.3 Stochastic Server Location Problem (SSLP)

The SSLP is a 2SP, where in the first stage a set of decisions are made to decide which servers should be utilized and a set of second-stage decisions assigning clients to servers. In this case, the random variables take binary values, which represent a client with a request occurring in the scenario or not. A more detailed description of the problem can be found in [Ntaimo and Sen, 2005]. In this work, we directly use the instances provided in SIPLIB [Ahmed et al., 2015]. In the experiments a SSLP with n servers, m clients, and s scenarios is denoted by SSLP_n_m_s.

B.4 Pooling Problem (PP)

The pooling problem is a well-studied problem in the field of mixed-integer nonlinear programming Audet et al. [2004], Gupte et al. [2017], Gounaris et al. [2009], Haverly [1978]. It can be formulated as a mixed-integer quadratically constrained quadratic program, making it the hardest problem class in our experiments.

We are given a directed graph, consisting of three disjoint sets of nodes, called the source, pool and terminal nodes. We need to produce and send some products from the source to the terminal nodes, using the given arcs, such that the product demand and quality constraints on the terminal nodes, along with the arc capacity constraints, are satisfied. The pool nodes can be used to mix products with different qualities in appropriate quantities to generate a desired quality product. The goal is to decide the amount of product to send on each arc such that the total profit from the operations is maximized. We consider a stochastic version of the problem as described in the case study of Li et al. [2011]. Here, in the first stage, we need to design the network by selecting nodes and arcs from the input graph, without knowing the quality of the product produced on source nodes and the exact demand on the terminal nodes. Once the uncertainty is revealed, in the second stage, we make the recourse decisions about the amount of product to be sent on each arc, such that demand and quality constraints on the terminal nodes are satisfied. In our case, we have 16 binary variables in the first stage and 11 continuous variables per scenario in the second stage. An instance of this problem is referred to as PP_s, where s is the number of scenarios.

C Data Generation & Supervised Learning Times

In this section, we report details of the data generation and training times for all problem settings in Tables 8 and 9, respectively. For training, we split the # samples into an 80%-20% train validation set, and select the best model on the validation set in the given number of epochs.

D Objective Results

In this section, we report the objective for the first-stage solutions obtained by each approximate MIP and the objective of EF (either optimal or at the end of the solving time). In addition, we report the objective of the approximate MIP. See Tables 10 through 13 for results. As mentioned in the main paper, the results from linear regressor (LR) are quite poor, with a significantly worse objective in almost every instance. This is unsurprising a linear function will not likely have the capacity to estimate the integer and non-linear second-stage objectives. For both SC and MC we can see that the true objective and the approximate-MIP objective are relatively close for all of the problem settings, further indicating that the neural network embedding is a useful approximation to the second-stage expected cost.

Problem	SC			MC		
	# samples	Time per sample	Total time	# samples	Time per sample	Total time
CFLP_10_10	5,000	0.36	1,823.07	10,000	0.00	13.59
CFLP_25_25	5,000	0.83	4,148.83	10,000	0.01	112.83
CFLP_50_50	5,000	1.54	7,697.91	10,000	0.01	135.57
SSLP_10_50	5,000	0.19	942.10	10,000	0.00	22.95
SSLP_15_45	5,000	0.19	929.27	10,000	0.00	16.35
SSLP_5_25	5,000	0.17	860.74	10,000	0.00	13.18
INVP_B_E	5,000	1.79	8,951.27	10,000	0.00	4.17
INVP_B_H	5,000	1.84	9,207.90	10,000	0.00	4.22
INVP_I_E	5,000	1.75	8,759.83	10,000	0.00	4.34
INVP_I_H	5,000	1.79	8,944.65	10,000	0.00	3.32
PP	5,000	0.24	1,202.11	10,000	0.00	14.86

Table 8: Data generation samples and times. Data was generated in parallel with 43 processes. All times in seconds.

	SC	MC	LR
CFLP_10_10	667.28	127.12	0.53
CFLP_25_25	2,205.23	840.07	0.28
CFLP_50_50	463.71	128.11	0.75
SSLP_10_50	708.86	116.17	0.63
SSLP_15_45	1,377.21	229.42	0.57
SSLP_5_25	734.02	147.44	0.05
INVP_B_E	344.87	1,000.14	0.02
INVP_B_H	1,214.54	607.49	0.02
INVP_I_E	2,115.25	680.93	0.02
INVP_I_H	393.82	174.26	0.02
PP	576.08	367.25	0.05

Table 9: Training times. All times in seconds.

E Model Selection

For the supervised learning task, we implement linear regression using Scikit-learn 1.0.1 [Pedregosa et al., 2011]. In this case we use the base estimator with no regularization. The multisingle-cut neural models are all implemented using Pytorch 1.10.0 [Paszke et al., 2019]. For model selection, we use random search over 100 configurations for each problem setting. For multi-cut and single-cut we sample configurations from Table 14. For both cases we limit the ReLU layers to a single layer with a varying hidden dimension. In the multi-cut case the choice of the ReLU hidden dimension is limited since a large number of predictions each with a large hidden dimension can lead to MILPs which are prohibitively expensive to solve. For the single-cut specific hidden dimensions, we have 3 layers, with Embed hidden dimension 1 and Embed hidden dimension 2 corresponding to layers before the aggregation and Embed hidden dimension 3 being a final hidden layer after the aggregation.

In Tables 15 and 16 we report the best parameters for each problem setting for the multi and single cut models, respectively. In addition, we report the validation MSE across all 100 configurations for each problem in box plots in Figures 5 to 8. From the box plots we can observe that lower validation MAE configurations are quite common as the medians are typically not too far from the lower tails of the distributions. This indicates that hyperparameter selection can be helpful when attempting to improve the second-stage cost estimates, however, the gains are marginal in most cases.

Problem	True objective				Approximate-MIP objective		
	SC	MC	LR	EF	SC	MC	LR
CFLP_10_10_100	7,174.57	7,109.62	10,418.87	**6,994.77**	7,102.57	7,046.37	5,631.00
CFLP_10_10_500	7,171.79	7,068.91	10,410.19	**7,003.30**	7,102.57	7,084.46	5,643.68
CFLP_10_10_1000	7,154.60	**7,040.70**	10,406.08	7,088.56	7,102.57	7,064.36	5,622.40
CFLP_25_25_100	**11,773.01**	**11,773.01**	23,309.73	11,864.83	11,811.39	12,100.73	10,312.21
CFLP_25_25_500	**11,726.34**	**11,726.34**	23,310.34	12,170.67	11,811.39	12,051.51	10,277.01
CFLP_25_25_1000	**11,709.90**	**11,709.90**	23,309.85	11,868.04	11,811.39	12,041.12	10,263.37
CFLP_50_50_100	25,236.33	**25,019.64**	45,788.45	25,349.21	26,309.43	26,004.88	18,290.63
CFLP_50_50_500	25,281.13	**24,964.33**	45,786.97	28,037.66	26,287.48	25,986.50	18,209.77
CFLP_50_50_1000	25,247.77	**24,981.70**	45,787.18	30,282.41	26,309.43	26,002.78	18,217.14

Table 10: CFLP detailed objective results: each row represents an average over 10 2SP instance with varying scenario sets. “True objective" for {SC,MC,LR} is the cost of the first-stage solution obtained from the approximate MIP evaluated on the second-stage scenarios. For EF it is the objective at the solving limit. “Approximate-MIP objective" is objective from the approximate MIP for {SC,MC,LR}. All times in seconds.

Problem	True objective				Approximate-MIP objective		
	SC	MC	LR	EF	SC	MC	LR
SSLP_10_50_50	-354.96	-354.96	-63.00	**-354.96**	-350.96	-339.42	-294.69
SSLP_10_50_100	**-345.86**	**-345.86**	-49.62	-345.86	-350.96	-328.54	-283.96
SSLP_10_50_500	**-349.54**	**-349.54**	-54.68	-349.54	-350.96	-332.82	-288.02
SSLP_10_50_1000	**-350.07**	**-350.07**	-55.45	-235.22	-350.96	-333.46	-288.55
SSLP_10_50_2000	**-350.07**	**-350.07**	-54.72	-172.73	-350.96	-332.87	-288.19
SSLP_15_45_5	-247.27	-206.83	-249.51	**-255.55**	-238.44	-259.11	-58.28
SSLP_15_45_10	-249.58	-209.49	-252.89	**-257.41**	-238.44	-265.92	-64.01
SSLP_15_45_15	-251.10	-208.86	-254.58	**-257.68**	-238.44	-267.01	-66.71
SSLP_5_25_50	-125.22	-123.15	14.50	**-125.36**	-121.64	-110.18	-119.98
SSLP_5_25_100	-120.91	-119.03	19.87	**-120.94**	-121.64	-109.59	-117.79

Table 11: SSLP detailed objective results: each row represents an average over eleven 2SP instance with varying scenario sets. See Table 10 for a detailed description of the columns.

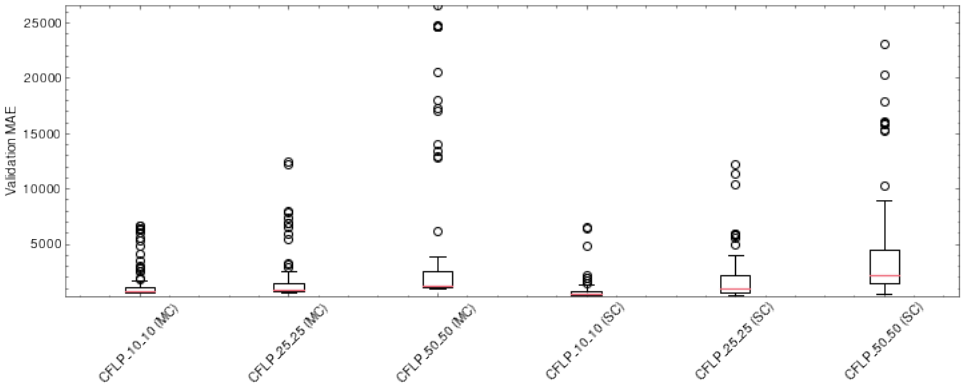

Figure 5: CFLP validation MAE over random search configurations for multi-cut (MC) and single-cut (SC) models.

Problem	True objective				Approximate-MIP objective		
	SC	MC	LR	EF	SC	MC	LR
INVP_B_E_4	-51.56	-55.29	-46.25	**-57.00**	-58.59	-52.15	-63.67
INVP_B_E_9	-54.86	-58.15	-53.11	**-59.33**	-58.81	-55.33	-63.67
INVP_B_E_36	-59.55	-58.19	-58.86	**-61.22**	-59.38	-57.92	-63.67
INVP_B_E_121	-61.44	-60.78	-61.06	**-62.29**	-59.60	-58.91	-63.67
INVP_B_E_441	-59.60	-59.83	-59.91	**-61.32**	-59.91	-58.51	-63.67
INVP_B_E_1681	-59.81	-	-59.30	**-60.63**	-59.94	-	-63.67
INVP_B_E_10000	**-59.85**	-	-58.68	-58.98	-59.95	-	-63.67
INVP_B_H_4	-51.75	-51.36	-51.75	**-56.75**	-58.12	-52.41	-61.24
INVP_B_H_9	-56.56	-56.56	-56.56	**-59.56**	-61.78	-56.67	-61.24
INVP_B_H_36	-59.31	-59.31	-59.31	**-60.28**	-59.38	-59.52	-61.24
INVP_B_H_121	-59.93	-59.93	-59.93	**-61.01**	-60.22	-60.54	-61.24
INVP_B_H_441	-60.14	-58.07	-60.14	**-61.44**	-60.23	-58.13	-61.24
INVP_B_H_1681	**-60.47**	-	**-60.47**	-60.04	-60.57	-	-61.24
INVP_B_H_10000	**-60.53**	-	**-60.53**	-58.93	-60.65	-	-61.24
INVP_I_E_4	-55.35	**-63.50**	-52.50	**-63.50**	-66.79	-58.96	-71.57
INVP_I_E_9	-61.63	-64.80	-61.89	**-66.56**	-66.70	-61.70	-71.57
INVP_I_E_36	-66.03	-66.25	-67.08	**-69.86**	-67.39	-65.18	-71.57
INVP_I_E_121	-67.35	-67.92	-69.07	**-71.12**	-67.39	-66.70	-71.57
INVP_I_E_441	-67.55	-69.16	-67.39	**-69.64**	-67.63	-67.43	-71.57
INVP_I_E_1681	-67.95	-66.73	-66.52	**-68.85**	-67.69	-67.62	-71.57
INVP_I_E_10000	**-67.94**	-	-65.67	-67.04	-67.82	-	-71.57
INVP_I_H_4	-54.75	-55.78	-54.75	**-63.50**	-65.31	-59.99	-66.07
INVP_I_H_9	-59.78	-65.25	-59.78	**-65.78**	-64.15	-61.08	-66.07
INVP_I_H_36	-63.78	-64.80	-63.78	**-67.11**	-66.79	-63.76	-66.07
INVP_I_H_121	-65.03	-64.37	-65.03	**-67.75**	-65.38	-64.64	-66.07
INVP_I_H_441	-65.12	-65.12	-65.12	**-67.24**	-67.13	-65.16	-66.07
INVP_I_H_1681	**-65.63**	-65.34	**-65.63**	-65.41	-65.87	-65.03	-66.07
INVP_I_H_10000	**-65.66**	-	**-65.66**	-64.63	-66.45	-	-66.07

Table 12: INVP detailed objective results: each row represents single instance. See Table 10 for a detailed description of the columns.

Problem	True objective				Approximate-MIP objective		
	SC	MC	LR	EF	SC	MC	LR
PP_125	264.30	173.10	-10.00	**273.19**	189.75	177.12	70.75
PP_216	200.29	131.83	-10.00	**220.25**	189.75	168.10	70.75
PP_343	206.38	122.90	-10.00	**207.77**	189.75	172.17	70.75
PP_512	204.41	134.83	-10.00	**223.86**	189.75	162.54	70.75
PP_729	219.42	137.97	-10.00	**222.48**	189.75	167.55	70.75
PP_1000	202.50	126.30	-10.00	**215.25**	189.75	165.59	70.75

Table 13: PP detailed objective results: each row represents single instance. See Table 10 for a detailed description of the columns.

Parameter	MC	SC
Batch size	$\{16, 32, 64, 128\}$	$\{16, 32, 64, 128\}$
Learning rate	$[1e^{-5}, 1e^{-1}]$	$[1e^{-5}, 1e^{-1}]$
L1 weight penalty	$[1e^{-5}, 1e^{-1}]$	$[1e^{-5}, 1e^{-1}]$
L2 weight penalty	$[1e^{-5}, 1e^{-1}]$	$[1e^{-5}, 1e^{-1}]$
Optimizer	{Adam, Adagrad, RMSprop}	{Adam, Adagrad, RMSprop}
Dropout	$[0, 0.5]$	$[0, 0.5]$
# Epochs	1000	2000
ReLU hidden dimension	$\{32, 64\}$	$\{64, 128, 256, 512\}$
Embed hidden dimension 1	-	$\{64, 128, 256, 512\}$
Embed hidden dimension 2	-	$\{16, 32, 64, 128\}$
Embed hidden dimension 3	-	$\{8, 16, 32, 64\}$

Table 14: Random search parameter space for multi-cut and single-cut models. For values in {}, we sample with equal probability for each discrete choice. For values in [], we sample a uniform distribution with the given bounds. For single values, we keep it fixed across all configurations. A value of - means that parameter is not applicable for the given model type.

Parameter	CFLP_10_10	CFLP_25_25	CFLP_50_50	SSLP_5_25	SSLP_10_50	SSLP_15_45	INVP_B_E	INVP_B_H	INVP_I_E	INVP_I_H	PP
Batch size	128	16	128	128	128	64	16	32	32	128	64
Learning rate	0.05029	0.05217	0.00441	0.03385	0.07015	0.08996	0.00435	0.00521	0.06613	0.01614	0.0032
L1 weight penalty	0.07512	0.00551	0.08945	0.03232	0.07079	0.09105	0.08321	0.05754	0.01683	0. 01859	0
L2 weight penalty	0.08343	0.02846	0.08602	0.0	0.01792	0.0	0.01047	0.02728	0	0	0.0361
Optimizer	Adam	Adam	Adam	RMSprop	RMSprop	RMSprop	RMSProp	RMSProp	Adam	Adam	Adam
Dropout	0.02198	0.02259	0.05565	0.00914	0.01944	0.02257	0.17237	0.13698	0.04986	0.0859	0.05945
ReLU hidden dimension	64	32	64	32	64	32	64	64	64	32	64

Table 15: MC best configurations from random search.

Parameter	CFLP_10_10	CFLP_25_25	CFLP_50_50	SSLP_5_25	SSLP_10_50	SSLP_15_45	INVP_B_E	INVP_B_H	INVP_I_E	INVP_I_H	PP
Batch size	32	16	128	64	64	32	128	32	16	128	64
Learning rate	0.0263	0.06571	0.02906	0.08876	0.07633	0.03115	0.01959	0.00846	0.02841	0.02867	0.08039
L1 weight penalty	0.02272	0.06841	0.05792	0.0	0.04529	0.07182	0.0	0.0	0.00022	0	0
L2 weight penalty	0.05747	0.0	0.04176	0.03488	0.0	0.0	0	0.09007	0.02272	0.01882	0
Optimizer	RMSprop	Adam	Adam	Adam	RMSprop	Adam	Adagrad	Adam	Adagrad	Adagrad	Adam
Dropout	0.01686	0.0028	0.03318	0.00587	0.00018	0.0088	0.08692	0.04096	0.01854	0.01422	0.0072
ReLU hidden dimension	128	256	256	256	64	256	256	256	256	256	512
Embed hidden dimension 1	512	128	256	64	128	512	256	512	64	256	512
Embed hidden dimension 2	16	64	64	16	32	64	16	16	32	32	128
Embed hidden dimension 3	16	16	8	32	64	16	32	16	8	64	16

Table 16: SC best configurations from random search.

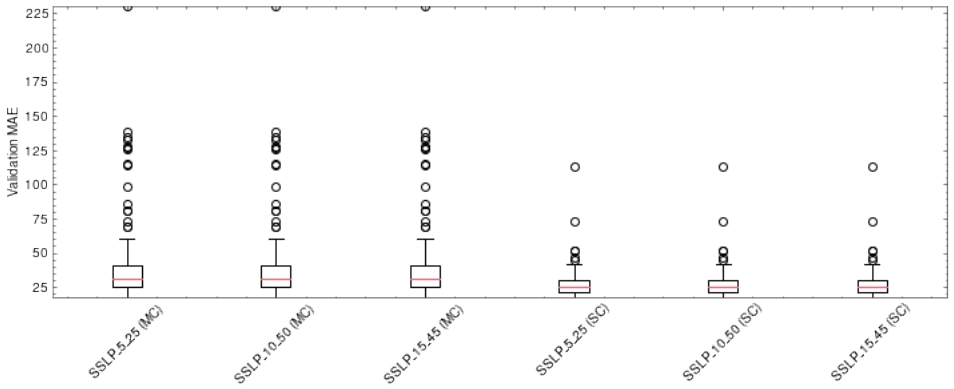

Figure 6: SSLP validation MAE over random search configurations for multi-cut (MC) and single-cut (SC) models.

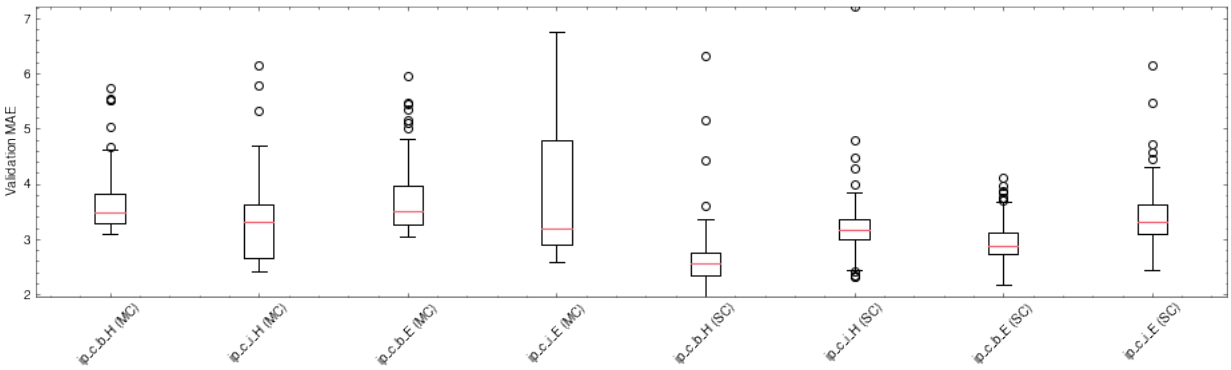

Figure 7: INVP validation MAE over random search configurations for multi-cut (MC) and single-cut (SC) models.

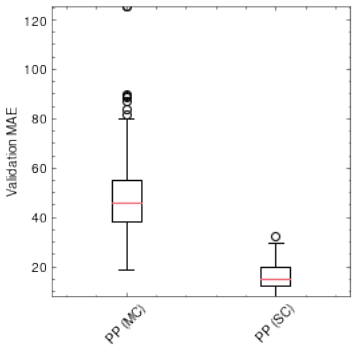

Figure 8: PP validation MAE over random search configurations for multi-cut (MC) and single-cut (SC) models.