

A Joint Learning and Optimization Approach for the Two-Stage Stochastic Service Network Design Problem

Ahmad Baubaid

baubaid@kfupm.edu.sa

King Fahd University of Petroleum and Minerals

Article

Keywords: stochastic service network design, learning-based heuristic, two-stage formulation

Posted Date: October 29th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11274651/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

A Joint Learning and Optimization Approach for the Two-Stage Stochastic Service Network Design Problem

Ahmad Baubaid^{1,2*}

¹Department of Industrial and Systems Engineering, King Fahd University of Petroleum and Minerals, Dhahran, 31261, Saudi Arabia.

²Interdisciplinary Research Center for Smart Mobility and Logistics, King Fahd University of Petroleum and Minerals, Dhahran, 31261, Saudi Arabia.

Corresponding author(s). E-mail(s): baubaid@kfupm.edu.sa;

Abstract

Freight consolidation carriers must plan shipping capacity for an upcoming operating season under uncertainty in shipment volumes. This often takes the form of an optimization problem: a two-stage Stochastic Service Network Design (SSND) problem, that is computationally intractable for realistic network sizes. We propose a neural network-augmented optimization framework for such a problem with general integer first-stage variables. The approach replaces the computationally expensive second-stage recourse cost subproblem with a feedforward neural network embedded directly into a mixed-integer programming formulation, which is solved as a surrogate model. Unlike prior work adopting such an approach, which assumes (bounded) binary first-stage, a challenge for effective learning in this setting is generating training data that adequately covers a bigger feasible solution space. Thus, we introduce and compare three data generation strategies: a random approach, a novel perturbation-based approach that seeds training samples from solutions of single-scenario problems, and a hybrid of the two. Experiments across 18 instances show that the choice of data generation method impacts solution quality and that the approach, overall, is a viable heuristic for this class of problems. The perturbation-based and hybrid approaches both outperform the random approach on a variety of metrics, and they result in better service network designs. The perturbation and hybrid approaches are then further benchmarked against Sample Average Approximation (which is often used to solve this problem) and results show comparable solution quality while offering

complementary trade-offs in computational effort and scalability, thus making the framework a viable alternative for freight consolidation carriers.

Keywords: stochastic service network design, learning-based heuristic, two-stage formulation

1 Introduction

Freight transportation is an essential component of modern supply chains, enabling the movement of goods between producers, distributors, retailers, and consumers. Among freight transportation carriers, consolidation-based carriers play an important role, as a large proportion of commercial shipments are too small to justify a dedicated vehicle and must instead be grouped with other freight to move economically. To achieve economies of scale, consolidation-based carriers, such as less-than-truckload (LTL) carriers, parcel and express delivery companies, and intermodal container shipping carriers route freight through intermediate terminals at each of which shipments from different origins are sorted and combined ([1]).

Every operating season, consolidation carriers must make a range of planning decisions regarding their services for that upcoming season. A *service* refers to a transportation operation that moves shipments between two terminals. Decisions made by carriers include which services to operate (including their capacities), how frequently to operate them, how freight should be routed via services through the network, the assignment of crews and vehicles during the operating season, and empty vehicle repositioning strategies. The objective is to provide a required level of service while minimizing overall transportation and operating costs. These decisions are generally modeled within the context of the service network design (SND) problem ([2]). The SND is thus a tactical-level transportation planning problem that is faced by and solved by consolidation-based transportation carriers.

A substantial body of research has focused on deterministic versions of the SNDP, where all problem parameters are assumed to be known with certainty. However, real-world transportation networks operate in highly uncertain environments. Freight demand is unknown when planning for an operating season and also fluctuates over time; travel times are affected by congestion and disruptions; and operational costs may vary because of changes in fuel prices, labor availability, and equipment utilization. As a result, service network designs that perform well under forecasted (expected) conditions may become inefficient or infeasible when actual operating conditions differ from forecasts.

To better capture these realities, the stochastic service network design (SSND) problem explicitly incorporates uncertainty into the planning process. By representing uncertain parameters through probability distributions and/or discrete scenarios, SSND models enable decision makers to evaluate network design alternatives across a range of possible future conditions, and to produce designs that perform well under multiple scenarios. As described in Section 2, a common SSND structure is a two-stage

stochastic program. For instance, first-stage decisions typically involve these selection of services that will operate between the terminals in the carrier’s network, and, after the uncertain demand is revealed, the second stage involves routing shipments from origins to destinations using the selected services. Excess demand can either be assumed unmet (with a penalty), or may trigger the acquisition of additional spot market services (as needed) in the second stage at a higher cost. In such models, demand is often represented in the form of scenarios obtained either from historical data or sampled from probability distributions.

However, solving such two-stage SSNDs presents significant computational challenges. The resulting models are often large-scale mixed-integer stochastic programs, and may feature integer variables in both stages. Furthermore, while increasing the number of demand scenarios can help make the model more representative, doing so causes problem sizes to grow rapidly. Given that the deterministic SND is intractable even for modest problem sizes ([3]), the SSND is even more so. Consequently, as outlined in Section 2, considerable research has been devoted to developing efficient exact and heuristic solution methods capable of addressing this problem.

With the rise of hybridized machine-learning and optimization approaches, learning-assisted approaches are being proposed for a variety of complex optimization problems. In this work, we propose and investigate a learning-assisted approach for the SSND with uncertain demand. In particular, we extend the work of [4]. While that study introduced the problem and presented initial results, this work includes a significantly expanded literature review, more detailed methodology and computational experiments sections, an exploration of additional data generation approaches, and more extensive computational experiments including benchmarking against a technique commonly-used for this problem, namely Sample Average Approximation (SAA).

Specifically, this paper makes the following contributions to the literature:

1. We develop a learning-based approach for the two-stage stochastic Service Network Design problem, one of a few such approaches for this class of problems, and the first for the general integer first-stage variant. Our approach makes use of a trained neural network as a surrogate model for the second-stage subproblem, and thus, simplifies the two-stage stochastic program.
2. We propose different data generation techniques to generate data for training the neural network. This includes a random-based approach (similar to those used in previous studies adopting a similar solution technique), a novel perturbation-based approach, and a hybrid of these two approaches.
3. We compare the proposed data generation approaches against each other using 18 instances. Specifically, we compare the approaches in terms of accuracy metrics on a testing dataset (i.e., MAE, MSE, and MAPE), as well as in terms of the actual realized cost of their resulting first-stage designs. Results show that the perturbation-based and hybrid approaches for data generation enable more effective learning and yield results that are better than the random-based approach.

4. We demonstrate the efficacy of our approach by comparing it against a Sample Average Approximation algorithm. Results show that the perturbation-based and hybrid approaches for data generation yield competitive designs with SAA. We also discuss the trade-off that our approach offers in terms of computational effort and solution quality.

The remainder of this paper is structured as follows. Section 2 reviews the relevant literature. Section 3 presents the mathematical model for the two-stage SSND considered in this study. Section 4 describes the proposed learning-based approach for the SSND, as well as how labeled training data is generated as part of the approach. Section 5 presents and discusses numerical experiments. Finally, section 6 concludes the paper and discusses a few future research directions.

2 Literature Review

2.1 Stochastic Service Network Design

The Service Network Design (SND) problem is a tactical planning problem commonly faced by freight consolidation carriers (less-than-truckload, rail, or parcel delivery carriers). SND involves carriers deciding the *services* to be offered in their network of terminals for the upcoming operating season. Decisions to be made include the routes, frequencies, and modes of the offered services, as well as how freight with a given origin and destination will be routed through this network (often referred to as a *freight flow plan* or a *load plan*). For an overview of service network design, we refer the reader to [1], [2], and, more recently, [5].

Service network design problems are often formulated as network design problems either on a spatial or “flat” network represented by the physical locations of the terminals, or on a time-expanded network which also models the specific schedules of services and the timed movement of freight. The latter is often referred to as the “scheduled service network design” problem ([2]). In particular, they are usually formulated as Multi-commodity Capacitated Network Design (MCND) problems either with binary or general integer design variables (representing service selection or offering) depending on the context; hence, they are typically mixed-integer programs ([6]). Typically, each origin-destination pair represents a commodity with a given demand value. Additional constraints may be added to control the freight flows from origins to destinations, e.g., a commodity must follow exactly one path from origin to destination. A variety of approaches, both exact and heuristic, have been proposed in the literature to tackle the SND in its various forms (see [7–11]). SNDs are known to be intractable except for small-sized instances ([3, 6]).

When planning, carriers often face uncertainty in commodity volumes, thus giving rise to the Stochastic Service Network Design (SSND) problem. In the SSND, demand values for each commodity are treated as random variables with a known probability distribution. SSNDs are typically formulated as two-stage stochastic programs where the first stage involves deciding the design of the network (service selection and flow plans, if applicable), and the second stage is for routing commodities through the selected services after demand information is revealed (typically with some recourse

decisions or penalties for unmet demand). The objective is to minimize the total expected cost: the design costs as well as the expected second-stage routing costs. Recourse decisions typically take the form of adding outsourced services (e.g., from the spot market) if design capacity is not sufficient. The deterministic SND model has been tackled with a variety of exact and heuristic algorithms, and several variants have been proposed and studied (the reader is referred to the aforementioned references for an overview on SND and their solution approaches); however, as the focus of this work is on the SSND and its solution approaches, we briefly highlight the research on the stochastic variant hereafter. A more in-depth overview on stochastic network design and, more specifically, on the SSND can be found in [12] and [5], respectively.

To the best of our knowledge, the first study of the SSND was conducted by [13], who investigated the impact of uncertainty on service network design problems by comparing the structure of stochastic solutions to their deterministic counterparts and analyzing the structural properties that help a service network design hedge against uncertainty.

A common approach in SSND modeling is to replace the expected second-stage cost in the objective with a sample average taken over a number of demand scenarios; most solution algorithms are also designed around this type of scenario-based model. The resulting problem is a deterministic mixed-integer linear program, and has been tackled using a variety of solution approaches most of which are heuristics. For instance, [14] apply a Variable Neighborhood Search inspired heuristic to solve instances of the scheduled stochastic service network design problem. A progressive hedging meta-heuristic (a Lagrangian-based decomposition technique for multi-scenario stochastic programs) with scenario decomposition was developed to solve such problems in [15]. In a follow-up work, [16] developed an approach for grouping scenarios that results in multi-scenario subproblems, and that this provides computational advantages provided the grouping of scenarios is done well. [17], on the other hand, applied an exact approach to solve the problem utilizing a number of acceleration strategies to enhance the convergence of a Bender's decomposition method. [18] develop a more general SND model that, in addition to the tactical decisions of SND, also takes into account strategic decisions (e.g., fleet sizing and resource acquisition). To solve this problem, they present and compare two column-generation-based matheuristics. [19] study flexibility in service network designs, and utilize a sample-average approximation method to solve a variant of the SND problem found in the context of less-than-truckload carriers. [20] study a variant of the SSND containing stochastic arc capacities (in addition to stochastic demands), and develop a partial Benders decomposition algorithm for the problem. Their approach involves integrating scenario information from the subproblems into the master problem. [21] develop a soft clustering scenario-grouping method to be used in a progressive hedging algorithm. In their approach, a single scenario can belong to more than one group depending on the scenario's probabilistic membership score to each of the groups. They argue and demonstrate that having some degree of overlap between the scenario groups helps in the convergence of the progressive hedging algorithm. [22] then show that dualizing the non-anticipativity constraint resulting from this group-wise decomposition yields a Lagrange bound that is greater than or equal to that of the scenario-wise decomposition case. [23] investigate the use

of reduced cost information from a deterministic SND problem in guiding the solution for the corresponding stochastic version, and utilize this in a matheuristic that iteratively solves restricted subproblems.

In summary, the SSND has been tackled using a variety of exact and heuristic approaches. Approaches used were Benders decomposition, progressive hedging, upgrading deterministic solutions, Sample Average Approximation (SAA), and metaheuristics. While there have been a few papers presenting learning-based approaches for the SSND, we defer their discussion to Section 2.3.

2.2 Leveraging machine learning to solve stochastic programs

Due to the difficulty of solving many combinatorial optimization problems, there has been more recent attention given to integrating machine learning techniques to help in solving difficult optimization problems. [24], [25], and [26] provide overviews of research that leverages machine learning to solve optimization problems. While we focus the remaining two sections of our review on works that leverage machine learning to solve stochastic programming problems, we also wish to highlight [27] which combines machine learning and local search for the *deterministic* multi-commodity capacitated fixed-charge network design problem, which can be seen as a special case of the more general SND problem.

While the use of machine learning in solving stochastic optimization problems is fairly old (e.g., approximating value functions is a common approach in the fields of approximate dynamic programming, control theory, and reinforcement learning), the use of machine learning to solve stochastic *programming* problems is relatively newer. In the context of stochastic programming, machine learning algorithms used to augment optimization algorithms typically focus either on scenario reduction, predicting *partial* solutions to (or other attributes of) problems to aid in solving them, i.e., these typically augment classical optimization approaches, or directly predicting *complete* solutions to problems. We provide brief reviews of each of these in turn.

A common approach in utilizing machine learning methods to tackle stochastic programming problems involves clustering or grouping scenarios to help make two-stage problems more tractable. For example, [28] develop a multivariate clustering approach and utilize recurrent neural networks to group scenarios in a renewable energy context. [29] develop a learning-based approach to predict a representative scenario that when used as the sole scenario in the deterministic equivalent of the stochastic program provides a high-quality solution for the two-stage stochastic program. [30] propose a deep learning approach for graph-based two-stage stochastic programs that can do both scenario reduction as well as prediction of the objective value. [31] combine sample average approximation with machine learning (the latter is used to cluster scenarios) to solve a stochastic humanitarian relief problem. [32] use a reinforcement learning algorithm and a hierarchical graph neural network to learn representations of scenarios, and select ones that are representative.

A number of research papers focus on directly learning and predicting partial solutions or objective function values of combinatorial optimization problems with the goal of using machine learning to aid in solving these problems (usually through a combination of machine learning and classical optimization). For instance, to tackle multistage

stochastic programming problems, [33] introduce a hybrid approach that combines scenario-tree-based stochastic programming and supervised machine learning by training the latter on a family of policies generated by solving scenario tree approximations. The machine learning model is used to infer the values of the decision variables for the scenario trees, and a feasibility restoration procedure is applied to ensure feasibility of the inferred decisions. The best policy is then selected using a validation step. Using neural networks, [34] develop a similar approach which learns a single general policy by leveraging all the data collected from the scenario trees, making the inferred policy more robust and providing computational advantages. [35] develop an approach that utilizes Support Vector Regression to predict optimal values for instances of an offshore wind farm layout optimization problem from a real-world dataset of (near optimal) solutions. This is done to aid decision-making and quickly estimate the potential of a site without having to determine that through explicit optimization. [36] develop a machine learning method to predict second stage solutions conditional on the first-solutions. This is used to accelerate the solution of a two-stage load planning problem since only a rough idea of a set of operational decisions/consequences (of a given set of tactical decisions) is needed to determine good first-stage values. [37] propose a learning-based approach that can help predict which parts of a solution to a routing problem will not change when the problem is solved repeatedly with relatively small changes to help speed up solve times. [38] present a framework that combines machine learning, optimization, and heuristics to predict solutions to multi-stage programs. A main paper of interest, whose method we adopt in our work, is [39] which presents Neur2SP, an approach that uses a ReLU neural network to approximate the second-stage expected costs in two-stage stochastic programs (especially ones where the second stage subproblems are already themselves intractable, e.g., mixed-integer programs). Thus, instead of approximating the expected second stage cost by a sample average obtained by a set of scenarios, a neural network approximator is embedded into the mathematical formulation, which helps obtain a surrogate problem that is more tractable than the traditional scenario-based form. Building on Neur2SP, [40] propose to instead utilize the distribution of the second stage objective making use of a quantile neural network. Also building on Neur2SP, [41] show that alternating between optimizing the first-stage decision variables and re-training the neural network can provide benefits. [42] present a matheuristic approach which combines the L-shaped method with a machine learning approach that predicts second stage solutions. [43] develop an approach that is similar to that of [39] for a stochastic single-source capacitated facility location problem. [44] hybridize supervised machine learning and an integer L-shaped approach to solve a risk-averse two-stage stochastic model for making train rescheduling decisions in a metro system.

In terms of directly predicting complete solutions to optimization problems, [45] develop a reinforcement learning algorithm that learns an iterative local search algorithm in the space of the first-stage variables of a two-stage mixed-integer stochastic program. The algorithm is able to learn (from a dataset of instances) an initialization policy and a local move policy to find good solutions to the studied problem. [46] propose machine learning techniques to predict solutions to learn operational decisions for a two-stage stochastic program for transshipping blood in a hospital network. [47]

develop an end-to-end learning approach for an online route-planning problem. [48] present a deep reinforcement learning framework for solving two-stage stochastic programs using two agents (trained sequentially) where each agent solves a stage of a problem. [49] provide a tutorial on “Neural Combinatorial Optimization”, i.e., the use of DRL as a heuristic technique to predict solutions to optimization problems.

Since most of the above approaches require a large dataset of instance solutions (or optimal values), and since solving problem instances can result in datasets that are not amenable to learning approaches (due to symmetries, alternative optimal solutions, etc.), this makes the supervised learning process more difficult. [50] study the data generation process for these learning-based approaches, and propose a method for data/instance generation that results in a dataset that is more amenable to supervised learning.

2.3 Learning approaches in SSND and related problems

In this section, we highlight a few papers that apply learning-based approaches to SSNDs and related problems. To the best of our knowledge, there have been very few such works, and our work contributes to this emerging line of research.

[51] develop a learning-based matheuristic approach for the two-stage stochastic program variant of the multicommodity capacitated fixed-charge network design problem, which is related to the SSND. Their approach learns potential good first-stage (binary) variables to fix from deterministic solutions to demand scenarios, motivated by the observation that solutions to stochastic problems typically share some commonalities with their deterministic counterparts. Their proposed approach alternates between predicting a set of promising first-stage variables (from scenarios and combinations of scenarios), and partially optimizing the reduced subproblem formed by fixing a certain percentage of variables in this promising set. This work was then extended in [52] in which the authors develop the “Integrated Learning and Progressive Hedging (ILPH)” which integrates the previous predict-and-optimize approach with progressive hedging. Their approach was also demonstrated on the two-stage stochastic variant of the multicommodity capacitated fixed-charge network design problem. [53] develop a multiple scenario approach that utilizes a k -means clustering algorithm to form group scenarios for a two-stage SSND. In this SSND variant, some vehicles have fixed routes during the entirety of the planning horizon, while some have realization-dependent routes. Each subproblem is then solved using a column-generation based matheuristic.

3 Mathematical Model

In this section, we present the mathematical formulation of the two-stage Stochastic Service Network Design (SSND) problem considered in this study.

In this SSND, a consolidation carrier has to plan for the transportation capacity to offer on each connection of its network of terminals given uncertain demand. We model the network of terminals as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. The carrier will transport a set of shipments or commodities, denoted by $\mathcal{K}$, whose origins and destinations are terminals in the network. We denote the origin and destination of commodity $k \in \mathcal{K}$ as $o(k)$ and $d(k)$, respectively. On its way from $o(k)$ to $d(k)$, a commodity k can

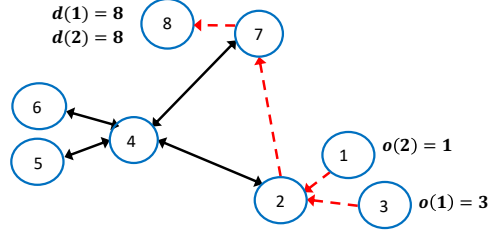


Fig. 1: Illustrative example of a consolidation carrier network (from [4]). There are two commodities with origins $o(1) = 3$ and $o(2) = 1$, and with a common destination $d(1) = d(2) = 8$. The services that the commodities might take through the network are shown as dashed lines, with the two shipments consolidated together from node 2 onwards. The black arrows represent other services operating within the network.

utilize other terminals as transshipment or consolidation points. In Figure 1, we show an example of a consolidation carrier's network and illustrate how two commodities might be transported using services (the arrows) in this network.

We define $\Omega \subseteq \mathbb{R}_+^{|\mathcal{K}|}$ to be a set of possible values for the random commodity demand shipment volumes, and refer to a particular realization using $\omega \in \Omega$. We denote the demand volume of commodity k in a realization ω as q_k^ω .

The SSND primarily involves deciding the *design* of the network. The design (or first-stage) decisions take the form of how many vehicles should be operated on each arc $a \in \mathcal{A}$ of the network, i.e., the planned capacity of the network. For simplicity, we assume that all vehicles have an identical capacity of Q . Operating a vehicle on an arc $a \in \mathcal{A}$ incurs a fixed cost f_a . These constitute the *first-stage* decisions of the problem. Depending on the commodity demand realization, if sufficient planned capacity is not available to meet its total demand, then additional vehicles obtained from the spot market (each also with capacity Q) can be added to an arc at a higher cost, $\bar{f}_a > f_a$. This latter set of decisions constitutes the *recourse* decisions in the problem. The objective of the SSND is to minimize the total expected cost.

To model the SSND, we make use of the following decision variables:

- r_{ij} first-stage/design (integer) variable for the number of vehicles to operate on arc $(i, j) \in \mathcal{A}$,
- x_{ijk}^ω second-stage variable for flow on arc $(i, j) \in \mathcal{A}$ for commodity $k \in \mathcal{K}$ in demand realization $\omega \in \Omega$, and
- $\bar{r}_{ij}^\omega$ second-stage (integer) variable for the number of outsourced vehicles to operate on arc $(i, j) \in \mathcal{A}$ in demand realization $\omega \in \Omega$.

The SSND model is then defined as follows:

$$\min \sum_{(i,j) \in \mathcal{A}} f_{ij} r_{ij} + \mathbb{E}_\omega [\mathcal{Q}(\mathbf{r}, \omega)], \quad (1a)$$

$$\text{s.t. } r_{ij} \in \mathbb{Z}_{\geq 0}, \quad \forall (i, j) \in \mathcal{A}, \quad (1b)$$

where

$$\mathcal{Q}(\mathbf{r}, \omega) = \min \left\{ \sum_{(i,j) \in \mathcal{A}} \bar{f}_{ij} \bar{r}_{ij}^\omega : (\bar{\mathbf{r}}^\omega, \mathbf{x}^\omega) \in \mathcal{P}(\mathbf{r}, \omega) \right\} \quad (1c)$$

and

$$\mathcal{P}(\mathbf{r}, \omega) = \left\{ (\bar{\mathbf{r}}^\omega, \mathbf{x}^\omega) : \sum_{k \in \mathcal{K}} x_{ijk}^\omega \leq Q(r_{ij} + \bar{r}_{ij}^\omega), \quad \forall (i, j) \in \mathcal{A}, \right. \quad (1d)$$

$$\left. \sum_{(i,j) \in \delta^+(i)} x_{ijk}^\omega - \sum_{(j,i) \in \delta^-(i)} x_{jik}^\omega = \begin{cases} q_k^\omega, & \text{if } i = o(k), \\ -q_k^\omega, & \text{if } i = d(k), \\ 0, & \text{otherwise,} \end{cases} \quad \forall i \in \mathcal{V}, k \in \mathcal{K}, \right. \quad (1e)$$

$$x_{ijk}^\omega \geq 0, \quad \forall (i, j) \in \mathcal{A}, k \in \mathcal{K}, \quad (1f)$$

$$\bar{r}_{ij}^\omega \in \mathbb{Z}_{\geq 0}, \quad \forall (i, j) \in \mathcal{A}. \quad (1g)$$

where $\delta^+(i)$ and $\delta^-(i)$ denote the sets of arcs in $\mathcal{A}$ leaving and entering node i , respectively.

The objective (1a) minimizes the total expected cost which is composed of the design variables' cost, i.e., the cost of the planned capacity, and a term representing the expected cost of the second stage subproblem taken over demand realizations. Constraints (1b) ensure integrality of the design variables. The second stage subproblem is given by $\mathcal{Q}(\mathbf{r}, \omega)$ in equation (1c), and its objective is to minimize the cost of the outsourced vehicles. The feasible region of the subproblem is defined by constraints (1d) – (1g). Constraints (1d) ensure that total commodity flow on an arc (i, j) respects its combined (planned and outsourced) capacity. Constraints (1e) are standard flow balance constraints that ensure the correct commodity quantities leave their origins and arrive at their destinations for the given realization (potentially using other terminals as transshipment/consolidation points). Finally, constraints (1f) and (1g) describe the domain of the second-stage/recourse variables.

Often, model (1) is converted to a deterministic equivalent model by using a finite set of scenarios, $\mathcal{S}$. When all scenarios $s \in \mathcal{S}$ are assumed to have identical probabilities, the deterministic equivalent of model (1) can be expressed as follows:

$$\min \quad \sum_{(i,j) \in \mathcal{A}} f_{ij} r_{ij} + \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \sum_{(i,j) \in \mathcal{A}} \bar{f}_{ij} \bar{r}_{ij}^s \quad (2a)$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{K}} x_{ijk}^s \leq Q(r_{ij} + \bar{r}_{ij}^s), \quad \forall s \in \mathcal{S}, \forall (i, j) \in \mathcal{A}, \quad (2b)$$

$$\sum_{j \in \mathcal{V}: (i,j) \in \mathcal{A}} x_{ijk}^s - \sum_{j \in \mathcal{V}: (j,i) \in \mathcal{A}} x_{jik}^s = \begin{cases} q_k^s, & \text{if } i = o(k), \\ -q_k^s, & \text{if } i = d(k), \\ 0, & \text{otherwise,} \end{cases} \quad \forall s \in \mathcal{S}, \forall i \in \mathcal{V}, \forall k \in \mathcal{K}, \quad (2c)$$

$$x_{ijk}^s \geq 0, \quad \forall s \in \mathcal{S}, \forall (i, j) \in \mathcal{A}, \forall k \in \mathcal{K}, \quad (2d)$$

$$\bar{r}_{ij}^s \in \mathbb{Z}_{\geq 0}, \quad \forall s \in \mathcal{S}, \forall (i, j) \in \mathcal{A}, \quad (2e)$$

$$r_{ij} \in \mathbb{Z}_{\geq 0}, \quad \forall (i, j) \in \mathcal{A}. \quad (2f)$$

We make use of the above deterministic equivalent in our solution approach for solving the SSND in lieu of model (1).

4 Solution Approach

In this section, we describe our learning-based method for solving the two-stage SSND model. We begin in Section 4.1 by giving a brief overview of (artificial) neural networks (NNs). Then, in Section 4.2 we present the details of the solution approach including how the NN is used to develop a MIP surrogate model. In Section 4.3, we describe the procedures used for generating labeled data for training the NN. Finally, in Section 4.4, we conclude this section with a description of a Sample Average Approximation (SAA) algorithm which serves as a benchmark in our computational experiments.

4.1 Neural networks

(Artificial) Neural Networks (NNs) are a class of machine learning models inspired by the structure and function of the biological brain. They consist of multiple layers of nodes, or “neurons”, typically including an input layer, one or more hidden layers, and an output layer. In a feedforward NN architecture, each neuron receives inputs from several others in the preceding layer, forming a network of weighted connections that determine the strength of the transmitted signals. In addition, each neuron has an associated bias term that allows the model to shift activation thresholds. The neuron then transforms its weighted inputs using a nonlinear activation function, producing an output that becomes the input to neurons in the next layer.

Through an iterative *training* process, the network adjusts its internal parameters (weights and biases) to minimize a loss/error function and improve its predictive accuracy on labeled data in a supervised learning framework. This enables the NN to learn complex mappings from an input space to an output space. An example of a feedforward NN architecture can be seen in Figure 2. This NN is *fully-connected* since every neuron (except those in the input layer) receives signals from all neurons of the previous layer.

Mathematically, a feedforward neural network maps an input vector $\mathbf{x} \in \mathbb{R}^n$ to an output vector $\hat{\mathbf{y}} \in \mathbb{R}^m$ through a series of hidden layers. Each layer performs an affine transformation of inputs of the previous layer followed by the application of a nonlinear activation function. For a neural network with L layers, the NN can be expressed mathematically using the following relationships:

$$\mathbf{h}^{(0)} = \mathbf{x}, \quad (3)$$

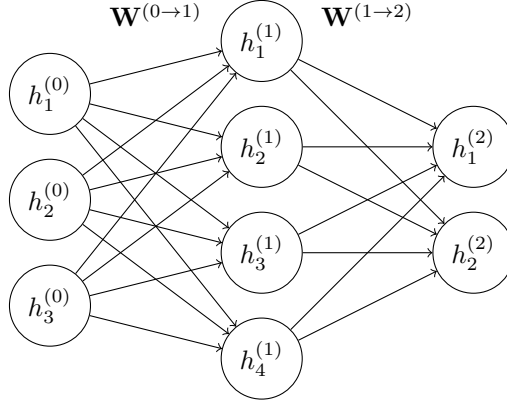


Fig. 2: Example feedforward neural network with one input layer, one hidden layer, and one output layer. The node post-activation outputs are labeled as $h_i^{(l)}$ and the weight matrices $\mathbf{W}^{(0 \rightarrow 1)}$, $\mathbf{W}^{(1 \rightarrow 2)}$ are indicated.

$$\mathbf{h}^{(l)} = \sigma\left(\mathbf{W}^{(l-1 \rightarrow l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l-1 \rightarrow l)}\right), \quad \forall l = 1, 2, \dots, L-1, \quad (4)$$

$$\hat{\mathbf{y}} = \mathbf{W}^{(L-1 \rightarrow L)} \mathbf{h}^{(L-1)} + \mathbf{b}^{(L-1 \rightarrow L)}. \quad (5)$$

In the above, $\mathbf{W}^{(l-1 \rightarrow l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ denotes the weight matrix connecting layers $l-1$ and l , where n_{l-1} and n_l are the respective numbers of neurons in those layers. The bias vector $\mathbf{b}^{(l-1 \rightarrow l)} \in \mathbb{R}^{n_l}$ allows for shifting the activation thresholds of the neurons in layer l . The function $\sigma(\cdot)$ represents the nonlinear activation function, and $\mathbf{h}^{(l)}$ denotes the post-activation output of layer l . The output (prediction) of the neural network is given by $\hat{\mathbf{y}}$; this can generally be a vector, although for our proposed solution approach, it is a scalar. Equation (3) defines the input layer. Equations (4) describe the transformations of the hidden layers: an affine combination of the outputs of the previous layer followed by a non-linear activation. Equation (5) represents the output layer, which produces the network's final prediction $\hat{\mathbf{y}}$.

The overall mapping can be denoted compactly as $f(\mathbf{x}; \Theta)$, where $\Theta = \{\mathbf{W}^{(l-1 \rightarrow l)}, \mathbf{b}^{(l-1 \rightarrow l)}\}_{l=1}^L$ is the set of tunable parameters of the NN. During training, the parameters are optimized in a supervised manner by minimizing a loss function $\mathcal{L}(f(\mathbf{x}; \Theta), \mathbf{y})$ where $\mathbf{y}$ represents the true target output.

In our implementation, we use the Rectified Linear Unit (ReLU) activation function, defined as $\text{ReLU}(x) = \max\{0, x\}$ for $x \in \mathbb{R}$. This choice facilitates embedding the trained neural network within the mixed-integer programming (MIP) surrogate model described in the following section.

4.2 Surrogate MIP for SSND

In our approach, we make use of a ReLU NN to learn and predict the expected recourse cost given a vector of design variables, i.e., given a vector $\mathbf{r}$, we train the NN

to approximate $\mathbb{E}_\omega [\mathcal{Q}(\mathbf{r}, \omega)]$. Labeled training data in our setup consists of input-out pairs of the form $\{(\mathbf{r}^{(i)}, \mathbb{E}_\omega[\mathcal{Q}(\mathbf{r}^{(i)}, \omega)])\}_{i=1}^N$ where the inputs are a choice of the vehicle vectors $\mathbf{r}$, and the output consists of an estimate of its expected recourse cost. Here, N represents the size of the training dataset. The aim is to replace the recourse cost computations entirely by the prediction of the NN. To generate the expected recourse cost label, $\mathbb{E}_\omega [\mathcal{Q}(\mathbf{r}, \omega)]$, one can use an exact value (if available) or can approximate this value (e.g., by solving the recourse subproblem over a *sample* of scenarios either exactly, or using a good heuristic). These recourse labels could also be obtained from historical data, if available.

The trained NN is then used to develop a surrogate MIP that replaces the term $\frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \sum_{(i,j) \in \mathcal{A}} \bar{f}_{ij} \bar{r}_{ij}^s$ in (2a) and also the second-stage subproblem constraints (2b)–(2e) in the deterministic equivalent. Since, we use a ReLU NN, it is possible to embed the trained NN to obtain a surrogate MIP model. We accomplish this using the formulation outlined in [54] which we briefly describe below for completeness.

To model a feed forward ReLU NN, we introduce binary variables of the form z_j^l and two sets of continuous variables $\hat{h}_j^l$ and $\check{h}_j^l$ for the j^{th} neuron in the l^{th} layer. Then, a general feed forward ReLU NN of the form given by equations (3)–(5) can then be modeled using the following sets of constraints:

$$\sum_{i=1}^{n_{l-1}} w_{ij}^{(l-1 \rightarrow l)} \hat{h}_i^{l-1} + b_j^{(l-1 \rightarrow l)} = \hat{h}_j^l - \check{h}_j^l, \quad \forall l \in \{1, \dots, L\}, j \in \{1, \dots, n_l\}, \quad (6)$$

$$z_j^l = 1 \Rightarrow \hat{h}_j^l \leq 0, \quad \forall l \in \{1, \dots, L-1\}, j \in \{1, \dots, n_l\}, \quad (7)$$

$$z_j^l = 0 \Rightarrow \check{h}_j^l \leq 0, \quad \forall l \in \{1, \dots, L-1\}, j \in \{1, \dots, n_l\}, \quad (8)$$

$$\hat{h}_j^l, \check{h}_j^l \geq 0, \quad \forall l \in \{1, \dots, L\}, j \in \{1, \dots, n_l\}, \quad (9)$$

$$z_j^l \in \{0, 1\}, \quad \forall l \in \{1, \dots, L-1\}, j \in \{1, \dots, n_l\}. \quad (10)$$

Constraints (6) ensure the correct propagation of outputs from each layer to the next, and obtain the output of the neural net. The ReLU activation function is achieved through the binary variables z_j^l , and the pair of variables $\hat{h}_j^l$ and $\check{h}_j^l$. Namely, the binary variable z_j^l allows at most one nonzero value in each pair of $\hat{h}_j^l$ and $\check{h}_j^l$. Since $\hat{h}_j^l$ represent the propagated variables, then according to the ReLU activation function, their values should be either positive (if affine combination of previous layer is positive) or zero (if negative). If the affine combination of the previous layer is positive, then $\hat{h}_j^l - \check{h}_j^l > 0$, and together with constraints (7) and (8), we have that $\hat{h}_j^l > 0$ and $\check{h}_j^l = 0$. Otherwise, if $\hat{h}_j^l - \check{h}_j^l < 0$, then we will have that $\hat{h}_j^l = 0$, and $\check{h}_j^l > 0$, so the propagated post-activation value will be zero, as desired. Equation (9) and (10) describe the domains of the variables used in the transformation. Note that for our application, $n_L = 1$ since our output is a scalar.

For completeness, the resulting *surrogate MIP* that we use to solve the SSND is as follows:

$$\min \sum_{(i,j) \in \mathcal{A}} f_{ij} r_{ij} + \hat{h}^L - \check{h}^L \quad (11a)$$

$$\text{s.t.} \quad (6) - (10), (2f), \quad (11b)$$

$$\hat{\mathbf{h}}^0 = \mathbf{r}. \quad (11c)$$

Constraint (11c) ensures that the $\mathbf{r}$ variable is passed as input to the neural net. Thus, the input layer consists of $\mathcal{A}$ neurons. We note that, in practice, constraints (2f) may need to be omitted when the neural network inputs corresponding to the vector of variables, $\mathbf{r}$, are standardized (e.g., using z-score normalization), such as in our implementation. This is because in such cases, $\mathbf{r}$ might take negative values as a result.

4.3 Data Generation

To generate labeled data for training the neural net, we propose the following three approaches. We note that, to the best of our knowledge, unlike previous similar surrogate solution approaches in the literature whose variables were binary and relied on random generation of binary vectors, our problem contains general integer variables in the first-stage, and thus the set of possibilities is unbounded in principal.

In each approach, we generate a set of integer vehicle design vectors, $\mathbf{r} = (r_{ij})_{(i,j) \in \mathcal{A}}$ where each vector provides a number of vehicles on each arc of the graph. These vectors are then labeled with an estimate of their expected recourse cost by solving a number of recourse subproblems exactly using a MIP solver. Although costly in terms of computational time, we opt for an exact computation (with a time limit) in this study to reduce the effect of approximation on the solution methodology. As discussed in Section 5, we also mitigate the effect of the estimation by choosing to evaluate the design vector on the entire “universe” of scenarios instead of sampling scenarios. In principle, however, the labeling of recourse costs can be either obtained from historical data, or via an approximation, e.g., from solving a sample problem or applying a heuristic.

To generate vehicle vectors with reasonable values, we impose a global constraint on the total number of vehicles: $\sum_{(i,j) \in \mathcal{A}} r_{ij} \leq R_{\max}$. The value $R_{\max}$ is not fixed a priori; instead, it is estimated empirically for each instance using a subproblem-based heuristic that solves a small number of randomly-selected time-limited *single-scenario models* using a MIP solver, and aggregates the resulting vehicle usage by taking $R_{\max}$ to be 125% of the maximum total number of vehicles observed across all single-scenario subproblems (this was calibrated through preliminary experiments). Although only a single scenario is taken into consideration in each subproblem, this is likely to be close to a representative estimate of the actual number of vehicles used while resulting in subproblems that are relatively easy to solve.

Whenever a generated solution violates this global limit, feasibility is restored by proportional scaling, i.e., the number of vehicles on an arc is updated according to $\left\lceil r_{ij} \cdot \frac{R_{\max}}{\sum_{(i,j) \in \mathcal{A}} r_{ij}} \right\rceil$, which preserves the relative structure of the generated solution while enforcing the global limit.

In the following, we describe the three approaches we use and later compare for data generation.

RANDOM. In this data generation approach, we generate vehicle vectors $\mathbf{r}$ randomly. That is, this approach is similar to approaches used in the literature in similar surrogate models. In particular, for each solution to be generated, integer vehicle counts are independently assigned to every network arc. The vehicle count r_{ij} is sampled uniformly from $r_{ij} \sim \text{Unif}\{0, \dots, U_{ij}\}$, where U_{ij} is a per-arc upper bound. We determine U_{ij} using a degree-based allocation strategy. Specifically, each arc (i, j) is assigned a weight equal to the sum of degrees of its two endpoints, $w_{ij} = \deg(i) + \deg(j)$. These weights are normalized and used to distribute $R_{\max}$ proportionally across arcs, yielding per-arc upper bounds $U_{ij} = \max\left(1, \left\lceil \frac{w_{ij}}{\sum_{(k,l) \in \mathcal{A}} w_{kl}} \cdot R_{\max} \right\rceil\right)$. The rationale behind this approach is that, while simple, it takes into consideration the relative importance of a node; e.g., arcs connecting more central consolidation hubs will generally have higher vehicle counts due to the higher demand flowing along these links.

After initial sampling, the total number of vehicles is computed. If the global vehicle limit exceed $R_{\max}$, feasibility is enforced via proportional scaling as described above.

Each generated vector/solution is then stored as a first-stage vehicle allocation, and the process is repeated until the desired number of randomly-generated vectors is obtained. To account for duplicates, we generate more vectors than needed, but only add unique vectors to the training dataset after obtaining their recourse cost. Generation stops once the target dataset size is reached.

PERTURBATION. In this approach, we generate vehicle vectors $\mathbf{r}$ by combining exact deterministic equivalent optimization with small local perturbations. This approach consists of two phases.

In the first phase, a collection of base solutions is first obtained by repeatedly solving τ *single-scenario* deterministic equivalent models. Although this approach can be used with a sampled set of scenarios instead of singletons, we choose singleton sets of scenarios for efficiency. Each solve is time-limited, and *all* integer-feasible first-stage solutions encountered during the solve are recorded to be part of the set of *base* solutions that will be used in the second phase. The idea behind this approach is to obtain a set of solutions that better resembles the set of solutions that the surrogate model might encounter in an optimization solver (solutions that may not be optimal but are of reasonable quality). Unique base solutions obtained from these runs are immediately added to the training dataset after obtaining their estimated recourse cost. Optionally, rounded LP relaxation solutions (for the single-scenario models) can also be added to the set of base solutions.

The remaining samples generated using this method are obtained by perturbing the base solutions in a second phase. Perturbations are distributed across base solutions such that multiple perturbed variants are generated per base solution. The goal of this step is to enrich the dataset with samples that lie in the local neighborhoods of high-quality solver-derived solutions. To generate these, a subset comprising $\rho^{\text{pert}}\%$ of the network arcs is selected uniformly at random. For each selected arc, the corresponding vehicle count is modified by an integer offset drawn uniformly from $\{-\kappa, \dots, 0, \dots, \kappa\}$, but ensuring that the number of vehicles on an arc does not fall below zero. After

applying these local modifications, the global vehicle limit, described above, is checked (and enforced, if necessary).

To account for duplicate configurations arising from perturbation and scaling, the procedure generates more candidates than required, adds unique vectors to the dataset, and stops the procedure once the target dataset size is reached.

HYBRID. In this approach, the data generation process combines the two approaches described above, i.e., a perturbation phase followed by a random phase. The motivation behind this approach is that the random approach is likely better at exploring a wide spectrum of solutions from those with low vehicle counts (where recourse costs are likely higher) to those with high vehicle counts (where recourse costs are likely lower). On the other hand, the perturbation approach is focused on relatively high-quality solutions (and their local perturbations) that are likely to be near-optimal for the stochastic program. Thus, hybridizing the two approaches is a natural idea to explore.

In this method, a certain percentage of required samples solutions, $\alpha \in (0, 1)$, is generated using the perturbation method, and the remainder of required training samples are obtained using the random approach. The combination of the two forms the complete the dataset. We study the effect of varying α in Section 5.4.

4.4 Sample Average Approximation

A common approach to solve two-stage stochastic programs is Sample Average Approximation (SAA) ([55]) which we will utilize as a benchmark for our approach in Section 5. In SAA, we sample N scenarios from Ω and use that to form a sample set $\bar{\mathcal{S}}$ where $|\bar{\mathcal{S}}| = N$. We then approximate the second-stage expected cost $\mathbb{E}_\omega [\mathcal{Q}(\mathbf{r}, \omega)]$ by this N -scenario sample average, $\frac{1}{N} \sum_{\omega \in \bar{\mathcal{S}}} \mathcal{Q}(\mathbf{r}, \omega)$. This resulting deterministic model is what we refer to here as the *sample problem*. For the SSND presented above, the resulting sample problem is a mixed-integer program (MIP) and is equivalent to model (2) but uses the sampled set of scenarios $\bar{\mathcal{S}}$.

The sample problem is then solved by a MIP solver to obtain a set of design variables, $\hat{\mathbf{r}}$. The recourse cost for $\hat{\mathbf{r}}$ is then estimated more accurately using a larger set of evaluation scenarios $N' \gg N$ as follows:

$$\mathbb{E}_\omega [\mathcal{Q}(\hat{\mathbf{r}}, \omega)] \approx \frac{1}{N'} \sum_{n=1}^{N'} \mathcal{Q}(\hat{\mathbf{r}}, \omega^n). \quad (12)$$

where the second-stage subproblem is independently solved for each individual subproblem.

In SAA, the above process is repeated M times, each with a newly-sampled set of N scenarios. Thus, this process yields up to M candidate design decisions. It then chooses the set of design variables that provide the best total estimated cost, which is the sum of the first-stage and second-stage evaluation cost.

For completeness, Algorithm 1 presents the full SAA procedure we adopt in this study.

Algorithm 1 Sample Average Approximation

Require: M, N, N' .

```
1: for  $m = 1, \dots, M$  do
2:   For sample problem  $m$ , generate  $\bar{\mathcal{S}} \subset \Omega$  with  $|\bar{\mathcal{S}}| = N$ .
3:   Generate  $N' \gg N$  demand scenarios for approximation of the second-
     stage cost.
4:   Obtain a first-stage design,  $\hat{\mathbf{r}}^m$ , by solving sample problem  $m$  given by
     (2) and scenario set  $\bar{\mathcal{S}}$ .
5:   for  $n = 1, \dots, N'$  do
6:     Solve second-stage subproblem (1c) with  $\hat{\mathbf{r}}^m$  and scenario  $n$  to
       obtain  $\mathcal{Q}(\hat{\mathbf{r}}^m, \omega^n)$ .
7:   Calculate the approximate expected total cost for design  $\hat{\mathbf{r}}^m$  given by
      $\hat{v}_{N'}(\hat{\mathbf{r}}^m) = \sum_{(i,j) \in \mathcal{A}} f_{ij} \hat{r}_{ij}^m + \frac{1}{N'} \sum_{n=1}^{N'} \mathcal{Q}(\hat{\mathbf{r}}^m, \omega^n)$ .
8: Select  $\hat{\mathbf{r}}^* \in \arg \min_{m \in \{1, \dots, M\}} \{\hat{v}_{N'}(\hat{\mathbf{r}}^m)\}$ .
9: return  $\hat{\mathbf{r}}^*$ .
```

5 Computational Experiments

In this section, we present our computational experiments. In Section 5.1, we provide details on instance generation and parameter settings. In Section 5.2, we compare the different data generation approaches described in Section 4.3, and also benchmark the learning-based approach against a Sample Average Approximation (SAA) algorithm. Finally, in Section 5.4, we investigate the effect of the α parameter in the perturbation data generation method.

5.1 Experimental Setup

5.1.1 Instances

Our experiments are conducted on randomly-generated instances of varying sizes. In particular, the method employed for generating instances for this study is similar to that of [19], and we describe it briefly for completeness.

Each instance consists of two types of nodes representing two classes of terminals: central hubs and satellite terminals. Node coordinates are drawn uniformly at random within a $[-30, 30]^2$ grid. To ensure hubs have more central locations, their coordinates are drawn uniformly at random within a smaller $[-18, 18]^2$ grid. On the other hand, satellite terminals can be located anywhere within the $[-30, 30]^2$ grid. For arcs, each hub is connected bidirectionally to its three nearest hubs (measured in terms of Euclidean distance), while each satellite terminal is connected bidirectionally to its nearest hub. A satellite terminal may additionally receive a second hub connection if two proximity conditions are simultaneously satisfied: (i) the direct distance to the alternate hub is less than 70% of the detour distance through the primary hub, and (ii) the direct distance to the alternate hub is less than 1.4 times the distance to the primary BB. These rules allow multi-loading at hubs that lie near a second hub without creating excessively dense networks. Calibration of these values was done in order

to keep the resulting second-stage (recourse) optimization problem computationally manageable. Minor manual adjustments to the locations of some terminals are also made in some instances to obtain reasonable network configurations. The first-stage fixed vehicle transportation cost, f_{ij} , for each pair of nodes (i, j) was set equal to the distance between their respective locations. Recourse vehicle costs were set to 150% of the corresponding first-stage fixed cost, i.e., $\bar{f}_{ij} = 1.5 \times f_{ij}$.

We define a set of commodities to represent shipments between ordered pairs of *satellite* terminals. If the set of satellite terminals is denoted by $\mathcal{V}^s$, then a random subset comprising a fraction, $\phi \in (0, 1)$, of all $|\mathcal{V}^s| \times (|\mathcal{V}^s| - 1)$ possible satellite pairs is selected to form the set of commodities in that instance. In particular, we select $\lceil \phi \cdot |\mathcal{V}^s| \times (|\mathcal{V}^s| - 1) \rceil$ commodities for each instance. Each commodity k is assigned a mean demand $\mu_k \sim \text{Uniform}[60, 80]$ and a standard deviation $\sigma_k \sim \text{Uniform}[10, 20]$, both drawn independently. Demand scenarios are generated by sampling from $\mathcal{N}(\mu_k, \sigma_k^2)$ independently for each commodity and scenario, but are restricted to only non-negative values (i.e., truncated). A universe of $|\Omega| = 100$ scenarios is generated per instance and fixed throughout all experiments, and all second-stage evaluations are done on all 100 scenarios. This is done for two reasons: (1) to ensure a manageable number of scenario subproblems (1c) to solve when evaluating the performance of a first-stage design exactly without the need to sample, and (2) to compare all approaches on the same set of scenarios and avoid the effect of sampling different scenarios for different approaches in our comparisons.

We consider six network configurations with $|\mathcal{V}^h|$ (the number of hubs) ranging from 4 to 6 terminals, $|\mathcal{V}^s| \in \{8, 12, 16\}$ terminals, and commodity fractions $\phi \in \{0.25, 0.35, 0.5\}$. Each network and commodity configuration is tested under three vehicle capacity profiles: $Q \in \{90, 140, 200\}$ units per vehicle, yielding tight, baseline, and loose capacity regimes, respectively. This creates a total of 18 instances. Instance sizes and parameters were chosen to ensure manageable second-stage subproblem solve times as they are solved to optimality when generating the dataset. The details of all 18 instances are summarized in Table 1. Note that the instances 4-8-0.35-X have a different network configuration compared to the other two 4-8-X-X instances.

5.1.2 Data Generation Parameters

In the following, we detail the parameter values used for the data generation methods described in Section 4.3. Table 2 summarizes the key settings.

For all approaches, $R_{\max}$ is estimated empirically per instance by solving, with the solver, 10 single-scenario subproblems (each with a 60-second time limit). The value of $R_{\max}$ is then set to be the maximum total vehicle count observed across the 10 scenarios multiplied by 1.25.

For the PERTURBATION data generation method, in the first phase, $\tau = 10$ single-scenario problems (in particular, 10% of $|\Omega|$) are solved with a 5-second time limit for each. We collect all unique integer-feasible $\mathbf{r}$ vectors encountered throughout these runs, and these form the set of base solutions for this method. The rounded LP relaxation solution of each single-scenario model is also added to the base solution set if it is not a duplicate. In the second phase, perturbations are applied to $\rho^{\text{pert}} = 25\%$ of arcs and apply an offset drawn from $\{-\kappa, \dots, \kappa\}$ with $\kappa = 3$.

Table 1: Complete set of instances.

Instance Identifier	$ \mathcal{V}^h $	$ \mathcal{V}^s $	ϕ	Q
4-8-0.50-200	4	8	0.50	200
4-8-0.50-140	4	8	0.50	140
4-8-0.50-90	4	8	0.50	90
4-8-0.35-200	4	8	0.35	200
4-8-0.35-140	4	8	0.35	140
4-8-0.35-90	4	8	0.35	90
4-8-0.25-200	4	8	0.25	200
4-8-0.25-140	4	8	0.25	140
4-8-0.25-90	4	8	0.25	90
5-12-0.50-200	5	12	0.50	200
5-12-0.50-140	5	12	0.50	140
5-12-0.50-90	5	12	0.50	90
5-12-0.25-200	5	12	0.25	200
5-12-0.25-140	5	12	0.25	140
5-12-0.25-90	5	12	0.25	90
6-16-0.35-200	6	16	0.35	200
6-16-0.35-140	6	16	0.35	140
6-16-0.35-90	6	16	0.35	90

For the HYBRID data generation method, the fraction of samples generated via the PERTURBATION METHOD is one of $\alpha \in \{0.25, 0.50, 0.75\}$; the remainder are generated via the RANDOM method. In Section 5.4, the effect of α is studied at a dataset size of 10,000 samples where α is varied between its three values. All other experiments use $\alpha = 0.75$ as this was found to work best. The perturbation parameters for HYBRID are the same as those used in PERTURBATION.

We evaluate all methods at dataset sizes: $\{250, 500, 1,000, 2,500, 5,000, 7,500, 10,000\}$. Note that these values represent the *total* number of samples in each one. Each dataset consists of a number of vehicle vectors $\mathbf{r}$ (i.e., the number of features is equal to $\mathcal{A}$), and are labeled with their second-stage/recourse cost evaluations obtained by solving (1c) on the set of scenarios in Ω and averaging them. All second-stage subproblems were solved to optimality and were parallelized.

Training for the neural network was done using the PyTorch library with an 80-20 training-testing random split of the datasets. Furthermore, we use 20% of the training set as a validation set which is used to select the best neural network hyperparameter configuration for each instance. In particular, hyperparameters are selected via a random search over 100 configurations drawn uniformly from the search space defined in Table 3. The configuration minimizing validation loss is selected and used to train the neural network. All networks use ReLU activations and are trained with MSE loss. Input features are standardized to zero mean and unit variance prior to training. After training the neural network using the combined training and validation datasets, its predictive performance is assessed using the independent test dataset (Section 5.2).

Table 2: Data generation parameter settings.

Parameter	Value	Method
Scenario universe size, $ \Omega $	100	All
Single-scenario subproblems for $R_{\max}$	10	All
$R_{\max}$ subproblem time limit (s)	60	All
$R_{\max}$ multiplier	1.25	All
Phase 1 solves, τ	10	PERTURBATION
Phase 1 time limit (s)	5	PERTURBATION
Arc perturbation percentage ρ^{pert}	25%	PERTURBATION and HYBRID
Max offset, κ	3	PERTURBATION and HYBRID
Hybrid perturbation fraction, α	$\{0.25, 0.50, 0.75\}$	HYBRID
Full dataset sizes	$\{250, 500, 1,000, 2,500, 5,000, 7,500, 10,000\}$	All

Finally, for Sample Average Approximation, we use the following parameters: $M = 5$, $N = 10$, and $N' = 100$ which is the full set of scenarios. Each mixed-integer linear program was time limited to 1200 seconds.

Table 3: Hyperparameter search space for neural network parameters.

Hyperparameter	Candidate values
Architecture [†]	[64], [128], [256], [512], [64, 64], [128, 128], [32, 32], [32, 32, 32], [64, 64, 64], [128, 64, 32], [512, 64, 32], [512, 128, 32]
Learning rate	10^{-5} , 10^{-4} , 10^{-3} , 10^{-2}
Batch size	32, 64, 128
Epochs	100, 150, 200, 250, 500, 750, 1000
Optimizer	Adam, Adagrad, RMSProp
L2 penalty	0, 0.01, 0.1

[†] Width per hidden layer; e.g., [64, 32] = two hidden layers of widths 64 and 32.

All computational experiments were carried out in Python on a conventional desktop workstation equipped with a 16-core AMD Ryzen 9 9950X3D processor and 128 GB of system memory. Neural network training was performed on an NVIDIA GeForce RTX 5090 graphics card with 32 GB of dedicated VRAM. All optimization problems were solved using Gurobi 13. The neural network was embedded into the surrogate problem (11a)–(11c) using the Gurobi Machine Learning library ¹, and a maximum runtime of 1,200 seconds was imposed when solving the surrogate model.

¹<https://gurobi-machinelearning.readthedocs.io/en/stable/>

5.2 Comparison of Data Generation Methods

In this section, we measure and compare how the trained neural networks using each of the three data generation approaches approximate the recourse cost of the problem. For this experiment, we will use the held-out test (20% of samples in the dataset).

First, let $y_i = \mathbb{E}_\omega[\mathcal{Q}(\mathbf{r}_i, \omega)]$ represent the true expected recourse cost for test vector $\mathbf{r}_i$ and let $\hat{y}_i = \mathbb{E}_\omega[\hat{\mathcal{Q}}(\mathbf{r}_i, \omega)]$ represent its corresponding prediction produced by the trained neural network. Then, we can compute the mean absolute error (MAE) as $\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$, the mean squared error (MSE) as $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$, and the mean absolute percentage error (MAPE) as $\frac{100}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i + \epsilon} \right|$ where $\epsilon = 10^{-8}$ and n is the number of testing dataset samples.

In Table 4, we compare these metrics for the three data generation approaches on all 18 instances using a total dataset size of 10,000 samples (of which 8,000 samples are used for training). In absolute terms, PERTURBATION is the most accurate on average, achieving the lowest mean absolute error (MAE = 9.11) and mean squared error (MSE = 161.95), ahead of RANDOM and HYBRID. In terms of MAPE, the ranking reverses: RANDOM yields by far the smallest mean absolute percentage error (2.02%), against 4.99% for PERTURBATION and 5.66% for HYBRID. A possible explanation for this is that while the approaches perform similarly in absolute error terms, upon investigation, the RANDOM approach yielded datasets which had higher average label values, y_i , since they generate many sparse designs which have relatively higher recourse costs. Dividing a similar absolute difference by these higher values results in a smaller MAPE value. For example, on the 5-12-0.50-90 instance, the average label values (recourse costs) were 1,395.52, 504.85, and 511.19, for RANDOM, PERTURBATION, and HYBRID, respectively. A similar pattern was observed for the other instances.

In addition to these prediction metrics, Table 5 compares the objective function value of the surrogate optimization model (11a)–(11c) with the corresponding *actual* cost objective obtained after evaluating its resulting first-stage solution across the full 100 scenarios. The former is reported under “Estimated Cost”, while the latter is reported under “Actual Cost”. To quantify the discrepancy between these two values, we report the absolute percentage error, denoted by $\Delta(\%)$, which is defined as

$$\Delta = 100 \times \left| \frac{\text{Act.Cost} - \text{Est.Cost}}{\text{Est.Cost}} \right|.$$

From Table 5, we observe that the PERTURBATION approach yields designs whose *total* costs have the closest agreement with the ground truth, with an average deviation of 1.33% and individual errors never exceeding 3.3%. The HYBRID approach follows closely at an average deviation of 2.12%. In contrast, the RANDOM approach produces designs with substantially larger discrepancies between estimated and actual costs, averaging 10.81% and reaching as high as 22.3%. Since PERTURBATION is designed to generate realistic vehicle vectors $\mathbf{r}$ from actual scenario subproblems, this could explain its superior performance. By learning from realistic vectors, it is more likely to generate designs that remain close to those observed in the training dataset, thereby resulting in more realistic solutions whose total costs are close to their actual values. On the other hand, since RANDOM sits at the opposite end of the spectrum, we see

Table 4: Prediction accuracy of the neural network generated by the three data generation approaches

Instance	RANDOM			PERTURBATION			HYBRID ($\alpha = 0.75$)		
	MAE	MSE	MAPE (%)	MAE	MSE	MAPE (%)	MAE	MSE	MAPE (%)
4-8-0.25-90	6.02	57.50	2.56	3.92	27.40	3.24	4.66	38.38	4.05
4-8-0.25-140	6.65	71.07	4.36	4.72	38.51	3.57	4.94	42.18	3.71
4-8-0.25-200	3.24	19.95	2.18	4.02	28.27	4.18	4.11	30.21	4.22
4-8-0.50-90	10.01	158.38	1.90	7.35	91.27	5.01	9.72	164.86	5.58
4-8-0.50-140	8.19	107.41	2.25	6.15	62.27	4.99	5.47	52.54	4.40
4-8-0.50-200	6.60	68.32	2.90	5.59	50.80	4.92	6.64	70.64	5.83
4-8-0.35-90	7.00	79.95	1.46	6.53	74.28	3.43	9.45	160.85	4.82
4-8-0.35-140	6.99	79.65	2.40	7.09	84.28	3.96	6.68	74.81	4.34
4-8-0.35-200	4.85	40.52	1.85	5.41	51.40	†	5.70	56.48	†
5-12-0.25-90	9.70	154.28	1.13	12.23	247.63	6.25	13.11	282.46	7.92
5-12-0.25-140	9.50	143.30	1.97	11.29	210.55	6.86	13.36	307.03	8.37
5-12-0.25-200	8.68	122.20	2.43	7.79	99.88	5.66	10.55	178.67	7.45
5-12-0.50-90	17.58	485.64	1.29	13.18	288.52	4.43	13.85	314.89	4.59
5-12-0.50-140	13.05	268.85	1.51	13.27	284.59	5.41	13.15	301.30	4.69
5-12-0.50-200	10.80	184.29	1.63	10.99	195.98	5.58	12.71	264.62	6.40
6-16-0.35-90	20.44	666.89	1.30	15.61	403.65	4.95	16.80	452.89	6.15
6-16-0.35-140	16.94	461.99	1.70	14.98	360.98	6.28	15.43	383.19	6.72
6-16-0.35-200	11.54	211.85	1.46	13.85	314.87	6.10	15.53	381.70	7.06
Average	9.88	187.89	2.02	9.11	161.95	4.99	10.10	197.65	5.66

† MAPE values were too large as they involved dividing by a zero y_i . These values are excluded when computing the averages.

that it is the worse-performing approach. HYBRID being a mixture of the two sits somewhere in between but is closer to the PERTURBATION approach; this is likely due to the choice of $\alpha = 0.75$. We also observe that, generally speaking, the Δ values grow with the growth in the instance size across all three methods.

Moreover, from Table 5, we can compare the actual costs of the designs obtained by all three approaches. We observe that PERTURBATION results in the lowest average actual cost, followed very closely by HYBRID, and then by RANDOM. Thus, both PERTURBATION and HYBRID data generation approaches yield datasets which, in turn, yield designs with better total actual costs compared to RANDOM thereby demonstrating the effectiveness of the proposed data generation approach. While the average performance is similar, HYBRID does manage to find better designs compared to PERTURBATION on a number of instances.

To study the effect of the training dataset size, we use instances 5-12-0.25-140 and 6-16-0.35-200 as examples. For these, we show in Figures 3–4 how the MAE, MSE, MAPE, and Δ values change as the training dataset size is increased. Our experiment involved *training* dataset sizes of 200, 400, 800, 2000, 4000, 6000, and 8000 samples. In particular, Figures 3a and 4a show MAE values, Figures 3b and 4b show MSE values, Figures 3c and 4c show MAPE values, and Figures 3d and 4d show Δ values. Unsurprisingly increasing the training dataset size generally decreases these error metrics until they mostly stabilize past 4,000 training samples for these instances. Furthermore, PERTURBATION and HYBRID exhibit mostly similar performance in all four figures with slightly more variations between them in the larger instance. All methods exhibit this downward trend on MAE, MSE, and MAPE. However, as seen

Table 5: Comparing surrogate solution estimated vs. actual total cost

Instance	RANDOM			PERTURBATION			HYBRID ($\alpha = 0.75$)		
	Estimated Cost	Actual Cost	Δ (%)	Estimated Cost	Actual Cost	Δ (%)	Estimated Cost	Actual Cost	Δ (%)
4-8-0.25-90	544.31	653.93	16.76	538.69	537.25	0.27	544.27	539.07	0.96
4-8-0.25-140	425.20	459.43	7.45	433.01	436.17	0.72	421.71	433.28	2.67
4-8-0.25-200	403.39	409.19	1.42	327.78	329.81	0.62	323.47	325.37	0.58
4-8-0.50-90	1041.45	1190.20	12.50	1047.03	1053.23	0.59	1034.15	1057.84	2.24
4-8-0.50-140	775.23	855.50	9.38	725.64	734.78	1.24	727.23	726.31	0.13
4-8-0.50-200	569.87	628.55	9.34	560.83	556.04	0.86	555.38	556.82	0.26
4-8-0.35-90	868.18	954.12	9.01	832.39	836.31	0.47	836.93	838.32	0.17
4-8-0.35-140	608.94	681.02	10.58	603.71	608.28	0.75	596.67	626.86	4.82
4-8-0.35-200	551.46	551.45	0.00	481.74	480.27	0.31	481.66	486.73	1.04
5-12-0.25-90	1457.72	1731.31	15.80	1468.51	1495.01	1.77	1466.66	1503.26	2.43
5-12-0.25-140	1031.13	1204.12	14.37	1023.51	1044.74	2.03	1025.91	1035.97	0.97
5-12-0.25-200	794.23	857.34	7.36	773.54	786.54	1.65	773.92	786.74	1.63
5-12-0.50-90	2817.85	3263.97	13.67	2737.99	2793.70	1.99	2717.47	2767.84	1.82
5-12-0.50-140	1856.43	2054.42	9.64	1808.89	1846.59	2.04	1833.26	1840.54	0.40
5-12-0.50-200	1375.89	1512.00	9.00	1355.22	1371.17	1.16	1343.77	1379.98	2.62
6-16-0.35-90	2913.13	3749.82	22.31	2895.87	2992.68	3.23	2823.93	3031.59	6.85
6-16-0.35-140	1941.84	2363.26	17.83	1902.49	1929.30	1.39	1805.28	1938.07	6.85
6-16-0.35-200	1538.40	1673.61	8.08	1452.43	1495.65	2.89	1443.08	1468.16	1.71
Average	1195.26	1377.40	10.81	1164.96	1184.86	1.33	1153.04	1185.71	2.12

in Figures 3d and 4d, it is worth highlighting that surrogate models trained using the HYBRID and PERTURBATION datasets initially yielded designs with worse Δ values compared to RANDOM on smaller datasets, but notably lower Δ values upon increasing the number of training samples from 800 to beyond 2,000 training samples.

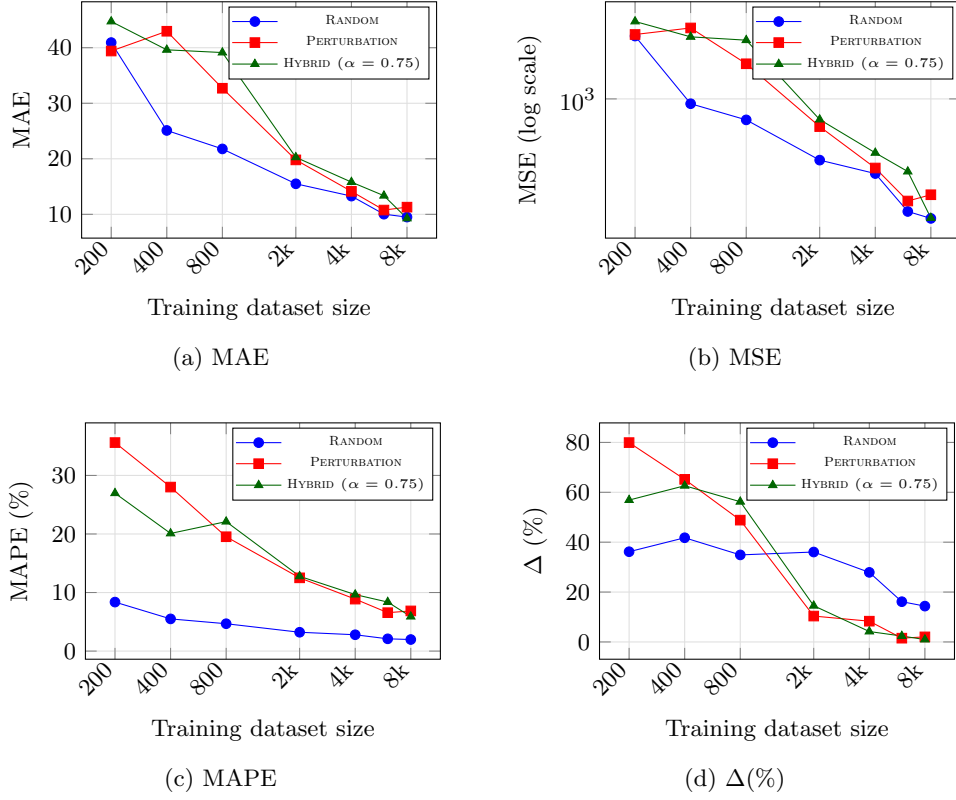


Fig. 3: Effect of dataset size on the three methods for instance 5-12-0.25-140.

In summary, the results above show that the PERTURBATION and HYBRID data generation approaches produce better results compared to RANDOM. They demonstrate the effectiveness of the perturbation-based mechanism used in generating design vectors, as it tends to generate design vectors that are more realistic and similar to the ones that are likely to be encountered during the solution process of the surrogate model. If data needs to be generated, then these two approaches are more effective compared to a random-based approach since the set of first-stage decision variables is unbounded and can be large. This also seems to influence the designs obtained by the surrogate model with PERTURBATION and HYBRID achieving comparable performance, and better designs on the instances tested compared to the RANDOM approach.

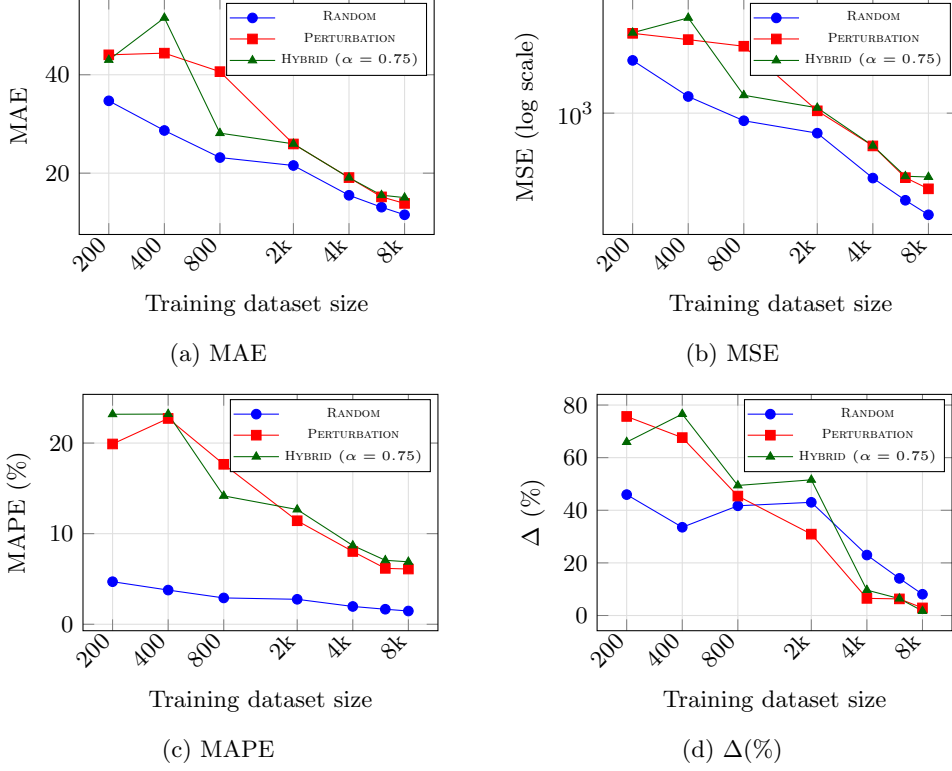


Fig. 4: Effect of dataset size on the three methods for instance 6-16-0.35-200.

5.3 Comparison with SAA

To benchmark the performance of our proposed learning-assisted approach for the two-stage SSND, we compare it (when using PERTURBATION and HYBRID ($\alpha = 0.75$) to generate the data) with the Sample Average Approximation approach presented in Section 4.4.

Table 6 shows the results of this comparison. In this table, we highlight the best value in bold, and show a “Gap to Best” column for each value which is defined as

$$\frac{\text{Cost of Method} - \text{Best Cost Across Methods}}{\text{Best Cost Across Methods}} \times 100\%$$

From Table 6, we first observe that on the first few instances which are relatively smaller in size, the three methods achieve comparable performance. On larger instances, however, SAA finds better designs consistently on the 5-12-X-X and the 6-16-X-X instances. However, PERTURBATION and HYBRID achieve close performance with an overall Gap to Best of 0.94% and 1.02%, respectively. Given that all three are considered heuristic approaches (SAA, in particular, is a heuristic due to the sampling

Table 6: Comparison of surrogate model-based approaches with SAA

Instance	PERTURBATION		HYBRID ($\alpha = 0.75$)		SAA	
	Actual Cost	Gap to Best (%)	Actual Cost	Gap to Best (%)	Actual Cost	Gap to Best (%)
4-8-0.25-90	537.25	0.00	539.07	0.34	537.25	0.00
4-8-0.25-140	436.17	0.67	433.28	0.00	434.97	0.39
4-8-0.25-200	329.81	1.36	325.37	0.00	325.37	0.00
4-8-0.50-90	1053.23	0.99	1057.84	1.43	1042.95	0.00
4-8-0.50-140	734.78	1.17	726.31	0.00	731.23	0.68
4-8-0.50-200	556.04	0.00	556.82	0.14	556.82	0.14
4-8-0.35-90	836.31	0.00	838.32	0.24	836.79	0.06
4-8-0.35-140	608.28	0.00	626.86	3.05	608.28	0.00
4-8-0.35-200	480.27	0.14	486.73	1.49	479.59	0.00
5-12-0.25-90	1495.01	0.98	1503.26	1.54	1480.53	0.00
5-12-0.25-140	1044.74	1.14	1035.97	0.29	1032.95	0.00
5-12-0.25-200	786.54	0.34	786.74	0.36	783.90	0.00
5-12-0.50-90	2793.70	1.46	2767.84	0.52	2753.49	0.00
5-12-0.50-140	1846.59	0.43	1840.54	0.10	1838.75	0.00
5-12-0.50-200	1371.17	0.46	1379.98	1.10	1364.94	0.00
6-16-0.35-90	2992.68	4.94	3031.59	6.30	2851.90	0.00
6-16-0.35-140	1929.30	0.69	1938.07	1.15	1916.07	0.00
6-16-0.35-200	1495.65	2.11	1468.16	0.23	1464.80	0.00
Average	1184.86	0.94	1185.71	1.02	1168.92	0.07

involved and the time limits imposed on the mixed-integer programs), this shows that the proposed learning-assisted approach is a viable heuristic for the problem.

When compared with the learning-assisted approaches, SAA finds better solutions, but its downside is the computational time required to solve M integer programs each with N scenarios. Depending on how these values (and their time limits) are set this can dictate the performance of the algorithm. On the other hand, the learning-assisted approaches can achieve similar performance by solving only one surrogate model. Its time limit can also be set according to preference which makes a direct time-based comparison with SAA somewhat challenging. The downside of these learning-based approaches is the training data required which took up the majority of computational time in our experiments. We note, however, that this can be slightly mitigated by relying on historical data and/or heuristic approaches to estimate the recourse costs and obtain the labels of the dataset, with our data generation approach used as a supplement, if not enough data is available. If copious amounts of labeled data are readily available, this makes the surrogate model approach more attractive since solving one model is less computationally intensive. We also note that the training of the neural network on the final dataset was relatively fast as there are only $|\mathcal{A}|$ input features, the size of which is typically not prohibitively large for most real-world networks.

As a direction for future research, training can be made more general, i.e., it might be possible to train one model that encompasses any instance of the problem as opposed to training on a per-instance basis as we do in this work. This would likely entail that the instance data become part of the features of the problem and this allows the model to adjust its estimate based on the network configuration and/or commodity demand data. Such an approach would enable better use of data, as a change in the network configuration will not require re-training the model. It may

also be possible to generalize data from smaller networks or subsets of larger networks to larger networks, thereby reducing the need for many data samples obtained for or from large instances of the problem.

5.4 Effect of α on the Hybrid Method

In this section, we study the effect of varying α on the HYBRID method. α controls the mix of PERTURBATION and RANDOM samples: $\alpha = 1$ corresponds to a pure PERTURBATION method, while $\alpha = 0$ corresponds to a pure RANDOM method. Thus, increasing α increases the number of samples that are obtained by PERTURBATION.

In Table 7, we compare HYBRID with different values of α and show both the accuracy on the testing set as well as the actual costs of the obtained designs. The best actual cost values are also highlighted in bold. We observe that increasing α from 0.25 to 0.75 monotonically reduces average MAE by 19.8% (from 11.66 to 9.35) and average MSE by 32.9%, with $\alpha = 0.50$ falling in between.

On the other hand, the average actual cost also falls monotonically from 1,211.40 at $\alpha = 0.25$ to 1,185.71 at $\alpha = 0.75$, a 2.1% reduction. $\alpha = 0.75$ attains the lowest (or tied-lowest) cost on 11 of the 18 instances, compared with 8 instances for $\alpha = 0.5$ and only 3 for $\alpha = 0.25$. More interestingly, on the smaller instances, the ranking is mixed and $\alpha = 0.5$ is often best. However, on the larger instances $\alpha = 0.75$ dominates, achieving the best on 8 of 9 instances on cost and cutting MAE by up to 46% relative to $\alpha = 0.25$. On the smaller instances, even when $\alpha = 0.75$ achieves a design that is worse than $\alpha = 0.5$, the actual cost value does not differ significantly. These observations are consistent with PERTURBATION being more valuable where the feasible/decision space is too large to cover by random sampling alone, and echoes the results seen in Section 5.2. They also justify the choice of $\alpha = 0.75$ from among the three values in previous experiments.

6 Conclusion and Future Work

In this paper, we proposed a neural network-augmented optimization framework for a two-stage stochastic service network design problem. The key idea is to replace the computationally expensive second-stage recourse cost with a trained feedforward neural network surrogate, which is then embedded directly into a mixed-integer programming formulation and then solved, as a surrogate model, using a commercial MIP solver.

This work also contains a comparison of three data generation strategies for training the surrogate model: a random approach similar to what was used in previous studies adopting a similar approach, a new perturbation-based approach, and a hybrid of the two. This was done because the problem at hand features general variables as opposed to binary variables. Our experiments across 18 instances show that the choice of data generation method meaningfully affects solution quality, particularly when comparing the random approach against the other two. The perturbation method, which seeds training data from solutions of single-scenario MIPs, produces the lowest absolute prediction error (MAE and MSE). Results also show that the difference in the resulting designs reflect the different regions of the solution space that each data

Table 7: Comparison of HYBRID approaches under different values of α

Instance	$\alpha = 0.25$				$\alpha = 0.5$				$\alpha = 0.75$			
	MAE	MSE	MAPE (%)	Actual Cost	MAE	MSE	MAPE (%)	Actual Cost	MAE	MSE	MAPE (%)	Actual Cost
4-8-0.25-90	6.70	73.31	4.31	537.25	4.07	29.03	3.55	537.25	4.66	38.38	4.05	539.07
4-8-0.25-140	6.68	75.73	4.81	444.97	4.62	35.56	3.60	433.28	4.94	42.18	3.71	433.28
4-8-0.25-200	5.25	45.57	4.37	327.51	4.36	33.01	3.78	325.37	4.11	30.21	4.22	325.37
4-8-0.50-90	10.87	203.95	3.64	1047.61	6.51	68.59	4.01	1049.71	9.72	164.86	5.58	1057.84
4-8-0.50-140	8.23	110.23	4.23	732.08	7.36	92.31	5.15	728.00	5.47	52.54	4.40	726.31
4-8-0.50-200	7.30	83.61	4.91	556.82	4.79	39.30	4.36	556.04	5.56	49.48	4.98	556.82
4-8-0.35-90	8.67	121.90	3.72	855.64	7.57	101.50	4.13	837.51	7.08	85.87	4.03	838.32
4-8-0.35-140	10.70	176.06	5.18	621.91	6.68	75.32	4.15	614.80	6.68	74.81	4.34	626.86
4-8-0.35-200	6.81	79.42	†	501.12	6.74	77.39	†	486.05	5.54	52.09	†	486.73
5-12-0.25-90	12.57	252.25	4.58	1530.71	11.50	217.98	5.92	1516.34	10.36	179.50	6.10	1503.26
5-12-0.25-140	12.51	250.45	6.36	1054.29	17.08	698.03	7.38	1037.70	9.26	144.42	5.89	1035.97
5-12-0.25-200	9.88	160.75	4.85	793.97	8.94	133.67	6.00	790.09	7.90	101.95	5.57	786.74
5-12-0.50-90	18.59	574.63	3.30	2884.61	14.37	370.61	4.38	2776.06	13.85	314.89	4.59	2767.84
5-12-0.50-140	14.76	351.18	3.49	1924.29	15.56	500.18	4.76	1851.11	13.15	301.30	4.69	1840.54
5-12-0.50-200	13.27	278.43	4.51	1401.11	12.09	236.45	5.65	1384.60	12.71	264.62	6.40	1379.98
6-16-0.35-90	23.25	856.57	4.02	2972.55	18.83	597.98	5.53	3311.59	16.80	452.89	6.15	3031.59
6-16-0.35-140	18.53	563.33	4.51	2017.11	17.13	480.49	6.75	2015.13	15.43	383.19	6.72	1938.07
6-16-0.35-200	15.27	376.47	4.63	1601.58	15.68	414.81	6.65	1487.99	15.03	377.12	6.89	1468.16
Average	11.66	257.43	0.04	1211.40	10.22	233.46	0.05	1207.70	9.35	172.79	0.05	1185.71

† MAPE values were too large as they involved dividing by a zero y_i . These values are excluded when computing the averages.

generation method explores. The hybrid approach offers a balance between coverage and solution quality. We also compare the perturbation-based and hybrid approaches against a Sample Average Approximation approach and show that they offer comparable results, and discuss the advantages and disadvantages of each approach. The data generation methods proposed in this study are useful on their own, when no data for training is available, or as a supplement to existing datasets.

This work can be extended in several directions. For instance, the surrogate model could be extended to be a single model that is able to solve any instance rather than the one-model-per-instance approach used in this study. This will allow data to be used more efficiently, without the need for re-training as networks grow or change over time. In cases where data is generated synthetically (as in this study), the number of samples that need to be generated would likely be reduced. In such an approach, it is likely that the instance characteristics (network size, commodity data, etc.) will be part of the features either directly or in aggregate form (e.g., basis functions). An alternative research approach is investigating direct learning-based approaches, i.e., learning-based or learning-assisted heuristics approaches that directly produce solutions/designs to the two-stage SND problem without the need for a surrogate model. Furthermore, the use of graph neural networks, which can naturally exploit the underlying network structure of the problem, either as part of a surrogate model (if viable) or as part of a direct learning-based heuristic for the problem can be investigated.

Acknowledgements. The author would like to acknowledge the support provided by the Deanship of Research (DR) at King Fahd University of Petroleum & Minerals (KFUPM) for funding this work through project No. EC26106.

Author Contributions. Ahmad Baubaid conceptualized the study, secured the funding, performed all work in this manuscript (coding, methodology, and experimental results and their analyses), and wrote the manuscript text.

Data Availability. All the datasets generated in this work and codes are available upon reasonable request to the corresponding author.

Funding. This work was supported by the Deanship of Research (DR) at King Fahd University of Petroleum & Minerals (KFUPM) through project No. EC26106.

Declarations

Conflict of interest. The author declares no competing interests.

References

- [1] Crainic, T.G.: Service network design in freight transportation. *European Journal of Operational Research* **122**(2), 272–288 (2000) [https://doi.org/10.1016/S0377-2217\(99\)00233-7](https://doi.org/10.1016/S0377-2217(99)00233-7)
- [2] Wieberneit, N.: Service network design for freight transportation: A review. *OR Spectrum* **30**(1), 77–112 (2008) <https://doi.org/10.1007/s00291-007-0079-2>

- [3] Ghamlouche, I., Crainic, T.G., Gendreau, M.: Cycle-based neighborhoods for fixed-charge capacitated multicommodity network design. *Operations Research* **51**(4), 655–667 (2003)
- [4] Baubaid, A.: Learning-assisted optimization for the stochastic service network design problem. *Transportation Research Procedia* **96**, 79–86 (2026) <https://doi.org/10.1016/j.trpro.2026.03.011>
- [5] Crainic, T.G., Hewitt, M.: Service network design. In: Crainic, T.G., Gendreau, M., Gendron, B. (eds.) *Network Design with Applications to Transportation And Logistics*, pp. 347–382. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-64018-7_12
- [6] Gendron, B., Crainic, T.G., Frangioni, A.: Multicommodity capacitated network design. In: Sansò, B., Soriano, P. (eds.) *Telecommunications Network Planning*, pp. 1–19 (1999)
- [7] Paraskevopoulos, D.C., Bektaş, T., Crainic, T.G., Potts, C.N.: A cycle-based evolutionary algorithm for the fixed-charge capacitated multi-commodity network design problem. *European Journal of Operational Research* **253**(2), 265–279 (2016) <https://doi.org/10.1016/j.ejor.2015.12.051>
- [8] Crainic, T.G., Hewitt, M., Toulouse, M., Vu, D.M.: Scheduled service network design with resource acquisition and management. *EURO Journal on Transportation and Logistics* **7**(3), 277–309 (2018) <https://doi.org/10.1007/s13676-017-0103-x>
- [9] Zetina, C.A., Contreras, I., Cordeau, J.-F.: Exact algorithms based on Benders decomposition for multicommodity uncapacitated fixed-charge network design. *Computers & Operations Research* **111**, 311–324 (2019) <https://doi.org/10.1016/j.cor.2019.07.007>
- [10] Crainic, T.G., Gendreau, M.: Heuristics and metaheuristics for fixed-charge network design. In: Crainic, T.G., Gendreau, M., Gendron, B. (eds.) *Network Design with Applications to Transportation And Logistics*, pp. 91–138. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-64018-7_4
- [11] Crainic, T.G., Gendron, B.: Exact methods for fixed-charge network design. In: Crainic, T.G., Gendreau, M., Gendron, B. (eds.) *Network Design with Applications to Transportation And Logistics*, pp. 29–89. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-64018-7_3
- [12] Hewitt, M., Rei, W., Wallace, S.W.: Stochastic network design. In: Crainic, T.G., Gendreau, M., Gendron, B. (eds.) *Network Design with Applications to Transportation and Logistics*, pp. 283–315. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-64018-7_10

- [13] Lium, A.-G., Crainic, T.G., Wallace, S.W.: A study of demand stochasticity in service network design. *Transportation Science* **43**(2), 144–157 (2009) <https://doi.org/10.1287/trsc.1090.0265>
- [14] Hoff, A., Lium, A.G., Løkketangen, A., Crainic, T.G.: A metaheuristic for stochastic service network design. *Journal of Heuristics* **16**(5), 653–679 (2010) <https://doi.org/10.1007/s10732-009-9112-8>
- [15] Crainic, T.G., Fu, X., Gendreau, M., Rei, W., Wallace, S.W.: Progressive hedging-based meta-heuristics for stochastic network design. *Networks* **58**(2), 114–124 (2011)
- [16] Crainic, T.G., Hewitt, M., Rei, W.: Scenario grouping in a progressive hedging-based meta-heuristic for stochastic network design. *Computers & Operations Research* **43**, 90–99 (2014) <https://doi.org/10.1016/j.cor.2013.08.020>
- [17] Rahmaniani, R., Crainic, T.G., Gendreau, M., Rei, W.: Accelerating the benders decomposition method: Application to stochastic network design problems. *SIAM Journal on Optimization* **28**(1), 875–903 (2018) <https://doi.org/10.1137/17M1128204>
- [18] Hewitt, M., Crainic, T.G., Nowak, M., Rei, W.: Scheduled service network design with resource acquisition and management under uncertainty. *Transportation Research Part B: Methodological* **128**, 324–343 (2019) <https://doi.org/10.1016/j.trb.2019.08.008>
- [19] Baubaid, A., Boland, N., Savelsbergh, M.: The value of limited flexibility in service network designs. *Transportation Science* **55**(1), 52–74 (2021) <https://doi.org/10.1287/trsc.2020.1009>
- [20] Crainic, T.G., Hewitt, M., Maggioni, F., Rei, W.: Partial benders decomposition: General methodology and application to stochastic network design. *Transportation Science* **55**(2), 414–435 (2021) <https://doi.org/10.1287/trsc.2020.1022>
- [21] Jiang, X., Bai, R., Wallace, S.W., Kendall, G., Landa-Silva, D.: Soft clustering-based scenario bundling for a progressive hedging heuristic in stochastic service network design. *Computers & Operations Research* **128**, 105182 (2021) <https://doi.org/10.1016/j.cor.2020.105182>
- [22] Jiang, X., Bai, R., Ren, J., Li, J., Kendall, G.: Lagrange dual bound computation for stochastic service network design. *European Journal of Operational Research* **302**(3), 1097–1112 (2022) <https://doi.org/10.1016/j.ejor.2022.01.044>
- [23] Sarayloo, F., Crainic, T.G., Rei, W.: A reduced cost-based restriction and refinement matheuristic for stochastic network design problem. *Journal of Heuristics* **27**(3), 325–351 (2021) <https://doi.org/10.1007/s10732-020-09460-y>

- [24] Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: A methodological tour d’horizon. *European Journal of Operational Research* **290**(2), 405–421 (2021) <https://doi.org/10.1016/j.ejor.2020.07.063>
- [25] Kotary, J., Fioretto, F., Hentenryck, P.V., Wilder, B.: End-to-End Constrained Optimization Learning: A Survey. *arXiv* (2021). <https://doi.org/10.48550/arXiv.2103.16378>
- [26] Schiffer, M., Hoppe, H., Su, Y., Bouvier, L., Parmentier, A.: Combinatorial Optimization Augmented Machine Learning. *arXiv* (2026). <https://doi.org/10.48550/arXiv.2601.10583>
- [27] La Rocca, C.R., Cordeau, J.-F., Frejinger, E.: Combining Supervised Learning and Local Search for the Multicommodity Capacitated Fixed-Charge Network Design Problem. *arXiv* (2024). <https://doi.org/10.48550/arXiv.2409.05720>
- [28] Crespo-Vazquez, J.L., Carrillo, C., Diaz-Dorado, E., Martinez-Lorenzo, J.A., Noor-E-Alam, Md.: A machine learning based stochastic optimization framework for a wind and storage power plant participating in energy pool market. *Applied Energy* **232**, 341–357 (2018) <https://doi.org/10.1016/j.apenergy.2018.09.195>
- [29] Bengio, Y., Frejinger, E., Lodi, A., Patel, R., Sankaranarayanan, S.: A Learning-Based Algorithm to Quickly Compute Good Primal Solutions for Stochastic Integer Programs. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58942-4_7
- [30] Wu, Y., Song, W., Cao, Z., Zhang, J.: Learning scenario representation for solving two-stage stochastic integer programs. In: *Proceedings of the 10th International Conference on Learning Representations, ICLR 2022, Virtual Online, Apr 25-29 (2022)*
- [31] Hu, S., Dong, Z.S., Dai, R.: A machine learning based sample average approximation for supplier selection with option contract in humanitarian relief. *Transportation Research Part E: Logistics and Transportation Review* **186**, 103531 (2024) <https://doi.org/10.1016/j.tre.2024.103531>
- [32] Wu, Y., Zhang, Y., Liang, Z., Cheng, J.: HGCN2SP: Hierarchical graph convolutional network for two-stage stochastic programming. In: *Forty-First International Conference on Machine Learning* (2024)
- [33] Defourny, B., Ernst, D., Wehenkel, L.: Scenario trees and policy selection for multistage stochastic programming using machine learning. *INFORMS Journal on Computing* (2012) <https://doi.org/10.1287/ijoc.1120.0516>
- [34] Murgia, G., Sbrilli, S.: Integrating multi-stage stochastic programming and machine learning for the evaluation of policies in the electricity portfolio problem. *IMA Journal of Management Mathematics* **28**(1), 109–130 (2017) <https://doi.org/10.1093/imaman/dax001>

[//doi.org/10.1093/imaman/dpv006](https://doi.org/10.1093/imaman/dpv006)

- [35] Fischetti, M., Fraccaro, M.: Machine learning meets mathematical optimization to predict the optimal production of offshore wind parks. *Computers & Operations Research* **106**, 289–297 (2019) <https://doi.org/10.1016/j.cor.2018.04.006>
- [36] Larsen, E., Frejinger, E., Gendron, B., Lodi, A.: Fast Continuous and Integer L-shaped Heuristics through Supervised Learning. *arXiv* (2022). <https://doi.org/10.48550/arXiv.2205.00897>
- [37] Morabit, M., Desaulniers, G., Lodi, A.: Learning to repeatedly solve routing problems. *Networks*, 22200 (2023) <https://doi.org/10.1002/net.22200>
- [38] Yilmaz, D., Büyüktaktın, İ.E.: A non-anticipative learning-optimization framework for solving multi-stage stochastic programs. *Annals of Operations Research* (2024) <https://doi.org/10.1007/s10479-024-06100-7>
- [39] Dumouchelle, J., Patel, R., Khalil, E.B., Bodur, M.: Neur2SP: Neural Two-Stage Stochastic Programming. *arXiv* (2022). <https://doi.org/10.48550/arXiv.2205.12006>
- [40] Alcántara, A., Ruiz, C., Tsay, C.: A Quantile Neural Network Framework for Two-Stage Stochastic Optimization. *arXiv* (2024). <https://doi.org/10.48550/arXiv.2403.11707>
- [41] Kronqvist, J., Li, B., Rolfes, J., Zhao, S.: Alternating mixed-integer programming and neural network training for approximating stochastic two-stage problems. In: Nicosia, G., Ojha, V., La Malfa, E., La Malfa, G., Pardalos, P.M., Umeton, R. (eds.) *Machine Learning, Optimization, and Data Science*, pp. 124–139. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-53966-4_10
- [42] Larsen, E., Lachapelle, S., Bengio, Y., Frejinger, E., Lacoste-Julien, S., Lodi, A.: Predicting tactical solutions to operational planning problems under imperfect information. *INFORMS Journal on Computing* **34**(1), 227–242 (2022) <https://doi.org/10.1287/ijoc.2021.1091> [arXiv:1807.11876](https://arxiv.org/abs/1807.11876) [cs, stat]
- [43] Chou, X., Messina, E., W. Wallace, S.: Solving two-stage stochastic programming problems via machine learning. In: Nicosia, G., Ojha, V., Giesselbach, S., Pardalos, M.P., Umeton, R. (eds.) *Machine Learning, Optimization, and Data Science* vol. 15508, pp. 1–12. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-82481-4_1
- [44] Su, B., Wang, F., Su, S., D’Ariano, A., Wang, Z., Tang, T.: Real-time metro train rescheduling under uncertainties: A hybrid machine learning and integer L-shaped approach. *Transportation Research Part E: Logistics and Transportation Review* **208**, 104704 (2026) <https://doi.org/10.1016/j.tre.2026.104704>

- [45] Nair, V., Dvijotham, K., Dunning, I., Vinyals, O.: Learning fast optimizers for contextual stochastic integer programs. In: UAI, pp. 591–600 (2018)
- [46] Abbasi, B., Babaei, T., Hosseini, Z., Smith-Miles, K., Dehghani, M.: Predicting solutions of large-scale optimization problems via machine learning: A case study in blood supply chain management. *Computers & Operations Research* **119**, 104941 (2020) <https://doi.org/10.1016/j.cor.2020.104941>
- [47] Zheng, J., Wang, L., Wang, S., Liang, Y., Pan, J.: Solving two-stage stochastic route-planning problem in milliseconds via end-to-end deep learning. *Complex & Intelligent Systems* **7**(3), 1207–1222 (2021) <https://doi.org/10.1007/s40747-021-00288-y>
- [48] Yilmaz, D., Büyüktaktakın, İ.E.: A deep reinforcement learning framework for solving two-stage stochastic programs. *Optimization Letters* (2023) <https://doi.org/10.1007/s11590-023-02009-5>
- [49] Angioni, D., Archetti, C., Speranza, M.G.: Neural combinatorial optimization: A tutorial. *Computers & Operations Research* **182**, 107102 (2025) <https://doi.org/10.1016/j.cor.2025.107102>
- [50] Kotary, J., Fioretto, F., Van Hentenryck, P.: Learning hard optimization problems: A data generation perspective. In: *Advances in Neural Information Processing Systems*, vol. 34, pp. 24981–24992. Curran Associates, Inc., Red Hook, NY, USA (2021)
- [51] Sarayloo, F., Crainic, T.G., Rei, W.: A learning-based matheuristic for stochastic multicommodity network design. *INFORMS Journal on Computing* **33**(2), 643–656 (2021) <https://doi.org/10.1287/ijoc.2020.0967>
- [52] Sarayloo, F., Crainic, T.G., Rei, W.: An integrated learning and progressive hedging matheuristic for stochastic network design problem. *Journal of Heuristics* **29**(4), 409–434 (2023) <https://doi.org/10.1007/s10732-023-09515-w>
- [53] Liu, C., Lin, S., Shen, Z.-J.M., Zhang, J.: Stochastic service network design: The value of fixed routes. *Transportation Research Part E: Logistics and Transportation Review* **174**, 103118 (2023) <https://doi.org/10.1016/j.tre.2023.103118>
- [54] Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. *Constraints* **23**(3), 296–309 (2018) <https://doi.org/10.1007/s10601-018-9285-6>
- [55] Kleywegt, A.J., Shapiro, A., Homem-de-Mello, T.: The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization* **12**(2), 479–502 (2002) <https://doi.org/10.1137/S1052623499363220>