Consolidation in Crowdshipping with Scheduled Transfer Lines: A Surrogate-Based Network Design Framework

Ramazan Çevik, Ali Şardağ, and Barış Yıldız
Department of Industrial Engineering, Koç University, Sarıyer, Istanbul, Türkiye
rcevik18@ku.edu.tr, asardag16@ku.edu.tr, byildiz@ku.edu.tr

October 7, 2025

Abstract

Crowdshipping has gained attention as an emerging delivery model thanks to advantages such as flexibility and an asset-light structure. Yet, it chronically suffers from a lack of mechanisms to create and exploit consolidation opportunities, limiting its efficiency and scalability. This work contributes to the literature in two ways: first, by introducing a novel consolidation concept that augments crowdshipping networks with dedicated scheduled delivery lines; and second, by developing a new methodology for solving complex network design problems, exemplified by the case studied in this paper, where design decisions needs to be made while taking complex operational dynamics into account, yet realistic representations of these dynamics are difficult to incorporate compactly into mathematical models. To address this challenge, we embed a neural network surrogate within a mathematical programming framework. The surrogate is based on a novel architecture that explicitly exploits the binary nature of design decisions, a feature common to many such problems. We establish theoretical guarantees regarding approximation accuracy and proximity to the true optimum. Numerical experiments with real-world data demonstrate both the practical benefits of the proposed hybrid delivery model—reducing costs, traffic, and emissions—and the computational advantages of the proposed solution framework.

Keywords: Crowdshipping, network design, surrogate models, neural networks

1 Introduction

The last few decades have enjoyed a substantial upward swing in both urbanization and digitization. In turn, both trends contribute to the explosive growth in the demand for e-commerce services. The ever-growing volume of e-commerce operations and the increasing service quality expectations of the modern customer in terms of short delivery times and delivery time flexibility put extraordinary strain on urban delivery systems, so much so that last-mile delivery makes up a majority of the package delivery cost (Capgemini 2023). In such a restrained yet competitive environment it is imperative to come up with delivery strategies that can reasonably keep up with the gargantuan demand, while keeping operations manageable and cost-effective.

One emerging idea is to turn to solutions rooted in sharing economy, by deferring the last-mile legs of delivery operations to *crowdshippers*. A crowdshipper is an “individual contractor” who agrees to carry a package along their existing pre-determined routes in exchange of a small, usually fiscal, compensation (Mohri et al. 2024). Crowdshipping-based delivery is an attractive alternative to traditional dedicated fleets for last-mile delivery because it requires relatively little infrastructural investment, and an abundance of potential carriers means a clever compensation strategy can ensure a financially favorable cost scheme for delivery operations.

However, crowdshipping systems are not the be-all-end-all solution to the urban delivery problem. Their financial viability critically depends on the effectiveness of the compensation mechanism, the design of which presents a fundamental challenge due to the need to manage a stochastic delivery capacity that can only be partially controlled through compensations (Chen et al. 2024). Besides these operational difficulties, a major roadblock in crowdshipping-based systems is that occasional carriers wish to integrate package deliveries into their existing trip plans only if this does not require excessive time or effort that could disrupt their primary activities. Combined with the fact that many rely on small personal vehicles such as bicycles or public transportation (Buldeo Rai et al. 2017), this makes it rather difficult to identify and exploit consolidation opportunities, thereby reducing the cost attractiveness of the system. This challenge is further reflected in the way crowdshipping has been modeled in the literature: classical single-mode crowdshipping operations are relatively straightforward to model mathematically and can be shown to be profitable in many cases (Gdowska et al. 2018); however, they inherently miss the possibility of consolidation.

To overcome this limitation, this study contributes to the literature by proposing a *multimodal* framework that combines crowdshipping with scheduled package transfers operated by company-run, high-capacity vehicles. When known in advance, these scheduled transfer lines (STL) can be incorporated into the planning of package routes, which may alleviate the need for excessive crowdshipper activity at certain points or create high-capacity, low-cost corridors to enhance system efficiency. Indeed, our case study with real-world data from Istanbul (presented in Section 5) demonstrates that such a strategy can reduce costs by more than 10% compared to single-mode crowdshipping, even when that benchmark already benefits from dynamic crowdshipper compensation and transshipment opportunities (Şardağ et al. 2024).

The promise of enhanced economic efficiency, however, comes with the added challenge of designing a system that involves multiple intertwined components. In particular, schedules for transfer lines must be determined by weighing the associated capital and operational costs against the potential savings in package routing (i.e., payments to crowdshippers and third-party service providers). Because these schedules drive tactical commitments—such as purchasing or allocating equipment (including vehicles), arranging staffing, and coordinating with external service providers—they must be fixed in advance of daily operations, thereby positioning them as tactical decisions with direct implications for downstream operational costs. Crucially, these decisions must be made without full information about daily package and crowdshipper arrivals, which only become known once operations unfold. Formalizing this complex design problem, we introduce the Crowdshipping Network Design Problem with Scheduled Transfer Lines (CND-STL) as determining the optimal set of scheduled transfer lines to operate so as to minimize the total cost over a specified planning period (e.g., upcoming week, month, or season), which comprises the expected payments to crowd-shippers (CSs) and couriers (on-demand delivery services as a back-up to ensure timely delivery), as well as the fixed and variable costs of executing the planned transfer lines.

For solving CND-STL, a key contribution of this study is the introduction of a novel solution framework for addressing challenging network design problems that involve multi-stage stochastic optimization with recourse actions, which are too intricate to be embedded directly into standard mathematical models. While broadly applicable across different domains, we demonstrate the framework in the context of crowdshipping. In our setting, the design problem is formulated as a Mixed Integer Quadratic Programming (MIQP) model that identifies the optimal spatio-temporal placement of company-operated transfer lines, while incorporating operational costs arising from real-time package routing and crowdshipper compensation decisions governed by a sequential decision-making policy aimed at minimizing expected costs. These complex operational costs are integrated into the MIQP through a surrogate neural network that predicts the operational costs associated with any given transfer line schedule under the adopted policy. Specifically, we employ a two-layer feed-forward neural network (FNN) with polynomial activation functions and show that, although such architectures do not provide universal approximation guarantees in general and are therefore not the first choice in general prediction tasks, they in fact possess such guarantees for discrete functions with binary domains and—most critically—can be embedded tractably into mathematical optimization models under mild assumptions. Importantly, while the framework relies on an estimator and may not always return the exact optimum, we establish probabilistic bounds that rigorously quantify the deviation from optimality in terms of estimator performance. This methodological advance provides a practical tool with solid theoretical foundations and is thus well-positioned to contribute to the broader agenda of the OR community in addressing challenging network design problems.

The rest of this paper is organized as follows: Section 2 presents an overview of the relevant literature. Section 3 exposes the problem setting, detailing our modeling assumptions and working principles of the envisioned system. Section 4 provides the technical details on the solution algorithms we develop for CND-STL. Finally, Section 5 presents computational results from our case study in Istanbul, showing the computational and operational results we obtain from our envisioned framework.

2 Literature review

From an application standpoint, our study contributes to the expanding body of literature on delivery network design within the gig economy, with a particular focus on crowdshipping systems for express deliveries in urban settings. Methodologically, our work aligns with the broader class of surrogate modeling, where complex systems are approximated using data-driven approaches to support decision-making. In the following subsections, we examine these two streams in detail.

2.1 Crowdshipping

Existing studies on crowdshipping can be classified into two main groups depending on whether the crowdshippers are *occasional carriers* drawn from the public of private travellers that transfer packages along their preplanned trips (Punel and Stathopoulos 2017) or they are *individual contractors* that dedicate some of their time to complete delivery tasks assigned to them (Yıldız and Savelsbergh 2019b). Although both models share common challenges, such as the lack of direct control over delivery capacity, their operational dynamics differ substantially. These differences are particularly evident in the level and structure of compensation offered, the mechanisms used for task assignment and incentive alignment, and the extent to which delivery performance can be monitored and enforced. For instance, while occasional carriers typically require minimal incentives but are constrained by their own travel plans (Arslan et al. 2019), individual contractors operate more flexibly, often under platform-mediated assignment rules with higher monetary compensations typically tied to performance metrics such as delivery time, distance, or total time spent in the system (Yıldız and Savelsbergh 2019a). Although the multi-modal delivery structure we propose and the methodological approach we develop for the resulting network design problem are applicable to both models, in this study, we focus on the former. We believe that our approach offers greater potential to address critical shortcomings regarding the lack of consolidation and mismatch of CS trajectories and package delivery demand often observed in systems relying on occasional carriers, while also enabling a more focused scope that facilitates a clearer exposition of our methodological contributions.

Earlier studies on the topic primarily focus on the division of work between company-owned delivery resources and occasional carriers, typically assuming deterministic settings for package and courier arrivals (Archetti et al. 2016, Macrina et al. 2017). More recent work, such as Arslan et al. (2019) and Dayarian and Savelsbergh (2020), incorporates stochasticity in both package and courier arrivals, reflecting a more realistic and operationally relevant dynamic. However, a critical challenge remains largely unaddressed in these studies: the spatial and temporal misalignment between the delivery demand and the planned trajectories of the crowd. Raviv and Tenzer (2018) and Yıldız (2021) are among the first to address this limitation by introducing the use of transshipment points and enabling the chaining of multiple crowd trips to complete a delivery—an approach that we also adopt in our work. Additional operational enhancements have been explored in the literature, including the integration of intermediate depots (Macrina et al. 2020), public transit as a delivery mode (Kızıl and Yıldız 2023), and dynamic courier compensation schemes (Şardağ et al. 2024). Distinct from these studies, our work proposes to improve system efficiency by introducing a new mode: company-operated transfer lines to create and exploit consolidation opportunities. We focus on tactical-level decisions related to the selection and operation of these transfer routes and propose a novel solution approach that uses ML-based surrogate modelling.

2.2 Surrogate Modeling

Serving as proxies for expensive simulations or complex structural relationships, surrogate models have found widespread use in operations research and related fields for various optimization problems. Applications span diverse areas such as production planning (Sun and Zhu 2025), energy systems (Bornatico et al. 2013), and healthcare (Anderer et al. 2022). For a comprehensive overview of surrogate modeling approaches, we refer the reader to the tutorial by Hong and Zhang (2021), and reviews by Alizadeh et al. (2020) and Bengio et al. (2021). Here, we focus specifically on machine learning–based surrogate models, which are more closely aligned with the methodological direction taken in our study.

Mišić (2020) is a pioneering study exhibiting optimization over a statistical surrogate model, where the surrogate is a tree-based estimator (or a collection thereof) and the resulting optimization problem can be modeled with tight binary constraints, leveraging the binary structure of the underlying estimator and eliminating the need for contrived convexification procedures. Notably, the paper lays the theoretical foundation of using binary tree classifiers as surrogate optimizers, but in practice, the accuracy of the embedded tree classifier is dependent on how efficiently the predicting features are chosen; the need for meticulous feature engineering also induces a potentially high number of binary variables to work with as each tree in the ensemble grows. Feed-forward neural networks have been used for this type of black-box function modeling, despite being significantly more challenging to embed into convex optimization models. The trade-off is that the convex formulation of a pre-trained neural-network-based model, and the tightness of the said formulation, is highly dependent on the choice of activation function. Despite the significant potential of creating accurate approximations with tabbing into the tremendous advancements in the field, computational challenges have been a major barrier for their effective use in solving bi-level or multi-echelon optimization problems (Aftabi et al. 2025). Fischetti and Jo (2018) is among the first studies that focus on the computational challenges arising from MILP modeling of ReLU networks, which works with Rectified Linear Unit (ReLU) functions that are embedded mathematical models with big-M type constraints. Grimstad and Andersson (2019) are centered on the surrogate modeling aspect of ReLU networks and provide a bound tightening technique to improve the solution time. On top of prevalent and essential big-M formulations of ReLU networks, Yang et al. (2022) present a mathematical program with complementarity constraints, which lacks global optimality guarantees, but generally outperforms the earlier approaches. Several studies also particularly focus on a priori design choices of neural network architectures or midstream hyperparameter adaptations to overcome challenges in the embedded model optimization, rather than embedding improvements for the pre-trained neural network. e.g., sparsity of the neural network (Rosenhahn 2023), loss function (Tsay 2021), activation function (Schweidtmann and Mitsos 2019). Our study belongs to this line of work with a tailored neural network architecture. Uniquely, we exploit the binary structure of the decision variables in our surrogate model and employ a quadratic activation function. This design yields a Mixed-Integer Quadratic Program (MIQP), which allows us to solve large-scale problem instances efficiently. Results of our numerical experiments, discussed in detail in

Section 5, show that the proposed NN architecture not only achieves a slightly better estimation performance than ReLU networks but also delivers substantial computational advantages, particularly as the network size grows to improve approximation quality.

3 Problem Definition and Notation

The operational setting considered in this study is mainly adopted from Şardağ et al. (2024), where the system dynamically determines the package routes and CS compensations along these routes to meet express delivery demand (i.e., two three-hour delivery windows) in an urban area. Although this setting and its related assumptions are not essential to our modeling approach, we retain them to ensure consistency with recent literature and to use Şardağ et al. (2024) as the state-of-the-art efficiency benchmark for evaluating the potential benefits of introducing scheduled transfer lines (STLs).

We consider a service area consisting of a set of service points, denoted by V, where packages can be picked up and dropped off by senders, receivers, and crowdshippers. The scope of the proposed system is limited to small package transfers (shoebox-sized parcels under 2–3 kg), which represent the majority of e-commerce deliveries and can be easily transferred by crowdshippers without needing any special equipment. A package p, has an origin $o_p \in V$, destination $d_p \in V$ and an arrival time t_p. The system operates over a discrete time horizon $T = \{t_0, \ldots, t_n\}$, during which packages arrive according to a stochastic process. The expected number of packages arriving at service point $i \in V$ to be delivered to service point $j \in V$ at time $t \in T$ is denoted by ε^t_{ij}. We consider express deliveries (e.g., same-day or hourly), where τ denotes the promised delivery lead time. That is, a package p must be dropped off at its destination d_p no later than time $t_p + \tau$.

The system uses three different transport modes for packages.

- The first is crowdshipping, where an individual crowdshipper (hereafter, CS) delivers a package from one service point to another, in exchange for a small compensation. Crowdshippers carry packages on their predetermined trips, i.e., do not make additional trips, long detours, or intermediate stops solely for the purpose of package delivery. To support this operational principle, service points are strategically located at high-traffic areas—such as shopping malls, public transit hubs, and educational or cultural centers—ensuring they are within short walking distance of common origins, destinations, or transfer points of everyday crowd trips. Since crowdshippers are expected to reach service points mainly on foot or through non-car modes of transport (e.g., public transit, bicycles, motorbikes, or e-scooters), we assume that each crowdshipper handles only a single package at a time. For origin-destination pairs, compensation offers are posted on a mobile app accessible to subscribed CSs. At each time point $t \in T$, the probability that a crowdshipper arrives before time $t+1$ to accept a package transfer task from service point i to j, given a compensation offer α, is denoted by $F^t_{ij}(\alpha)$. The flow of crowdshippers between service points follows the spatiotemporal distribution of personal trips, with service points near major transit hubs typically experiencing higher crowdshipper arrival rates and shorter expected waiting times. The time it takes for a CS to transfer a package from a station i to j is denoted by δ_{ij} The delivery company dynamically adjusts compensation offers based on the locations and the remaining delivery times for packages in the system.
- The second mode is an on-demand third-party courier service (referred to as couriers). In contrast to crowdshippers, couriers are available instantly upon request and can fulfill any portion of the delivery demand as needed. Consistent with current practice, the on-demand delivery service charges a fixed price α_{ij} for an origin–destination pair (i, j). The time it takes a courier to complete the delivery is denoted by $\bar{\delta}_{ij}$, which is assumed to be no more than δ_{ij}.
- The third mode—constituting the main novelty of this study—is the scheduled transfer

lines (STL), operated by company-owned vehicles that transport packages between designated origin-destination pairs at specific times, according to a fixed schedule. The fixed cost of operating an STL vehicle is denoted by $\bar{\alpha}$ and the per km operating cost by $\ddot{\alpha}$. We assume that the capacities of STL vehicles are not a limiting factor, as we consider large cargo vehicles and focus on the scheduling aspect of operations, where capacity constraints are unlikely to be binding under typical demand levels for the express delivery setting considered in this study. The STL lines selected for operation do not need to be concatenated end-to-end to form continuous vehicle tours, as we allow vehicles to make repositioning trips between transfer lines when necessary. The ordered set of candidate STLs is denoted by L, where a line $\ell \in L$ has an origin $\ell_o \in V$, destination $\ell_d \in V$, starting time $\ell_t \in T$, and arrival time $\ell_a \in T$. An STL schedule $\boldsymbol{\sigma}$ is a vector in $\{0,1\}^{|L|}$ where the ith entry takes the value of one if that line is to be operated in the schedule and zero otherwise.

Under these working principles, the system operates as follows. As a tactical-level decision, the platform determines the STLs to operate, e.g., for the coming month or season during which the delivery demand is expected to be similar. Then, in the operational level, when a package arrives at a service point, the next transfer point and an initial crowdshipper compensation are determined and dynamically updated (considering the STLs that will be operated) via an optimization engine such as the one proposed in Şardağ et al. (2024).

Let $\mathcal{C}(\varepsilon, \tau, \boldsymbol{\sigma})$ be the expected payments to crowdshippers and couriers (hereafter referred to as the C&C cost) for expected package arrivals ε, a service time guarantee τ, and an STL schedule $\boldsymbol{\sigma}$, evaluated under the optimal dynamic policy for package routing and courier compensation that minimizes total payments while ensuring deliveries within the promised service time. For the sake of brevity, when the context is clear, we will drop the parameters ε and τ and simply write $\mathcal{C}(\boldsymbol{\sigma})$ to denote this cost associated with the decision $\boldsymbol{\sigma}$.

We define $\mathcal{R}(\boldsymbol{\sigma})$ as the minimum cost of executing the STL schedule $\boldsymbol{\sigma}$ with respect to the fixed cost $\bar{\alpha}$ for operating a vehicle and the per km cost $\ddot{\alpha}$, for the movement of vehicles to cover all the STL lines indicated in $\boldsymbol{\sigma}$.

The *Crowdshipping Network Design Problem with Scheduled Transfer Lines* (CND-STL) is then defined as finding the optimal STL schedule $\boldsymbol{\sigma} \in \{0,1\}^{|L|}$ that minimizes the total delivery cost $\mathcal{C}(\varepsilon, \tau, \boldsymbol{\sigma}) + \mathcal{R}(\boldsymbol{\sigma})$.

Before concluding this section, we want to remark that our analysis adopts a tactical perspective on STL scheduling, assuming that the set of transfer lines to be operated is determined in advance—for example, for an upcoming month or season—based on expected demand patterns. This assumption reflects practical considerations such as operational simplicity, procurement arrangements, and the lead time required to organize and allocate fleet resources. Importantly, however, it is not essential for the proposed solution method: the methodology can readily be adapted to more dynamic STL routing decisions—for instance, through a rolling-horizon algorithm that re-optimizes STL routes at specific decision epochs—if such flexibility is warranted by the operational context.

4 Solution Methodology

The problem we address requires making two key interrelated decisions: (1) determining which transfer lines to operate, and (2) scheduling company-owned vehicles to serve the selected lines. Expanding the set of transfer lines reduces reliance on crowdshippers and couriers and can lower the C&C costs. However, operating more lines also increases the number of vehicles required and leads to higher fixed and operational costs for the fleet. A major modeling challenge arises from the nonlinear and opaque relationship between the selected set of transfer lines and the resulting C&C cost, which cannot be expressed in a closed-form expression due to the complex spatial and demand-based dependencies between STLs and crowdshipper compensations, especially in a dynamic compensation setting.

To address this challenge, we propose a novel surrogate modelling approach. Specifically, we train a neural network to predict the expected C&C cost as a function of the selected set of operated lines. Leveraging the fact that line selection decisions are binary (i.e., a line is either operated or not), we reformulate the neural network using a convex representation based on quadratic inequalities. This reformulation allows the neural network to be efficiently embedded into the main route optimization model to determine the STL schedule.

To make it easier to follow, we structure the technical exposition of our proposed framework as follows. We first describe the neural network architecture used to model this cost function and explain how it is reformulated with convex inequalities. Then, we present the integrated model that simultaneously determines the optimal set of operated lines and the routing decisions for company-owned vehicles.

The expected C&C costs for a given STL schedule naturally depend on the real-time policy used to determine package routes and CS compensations. To this end, we adopt the policy proposed by Şardağ et al. (2024), implementing it through a dynamic programming methodology described in Appendix A.

4.1 Machine Learning Surrogate

For the selection and routing of the STLs, we use a supervised machine learning model to estimate $\mathcal{C}(.)$, which will be the surrogate for the selection part of the objective. Before presenting the machine learning model specifications and the joint problem formulation, we first describe the data generation process used to train the machine learning model, which we refer to as *data farming*.

4.1.1 Generating the Training Data

To fit the machine learning model, we progressively generate a dataset of $(\boldsymbol{\sigma}, \mathcal{C}(\boldsymbol{\sigma}))$ pairs as regression inputs and targets for the machine learning model. Inspired by active learning applications, we use an adaptive approach to increase the information content of the final dataset as much as possible.

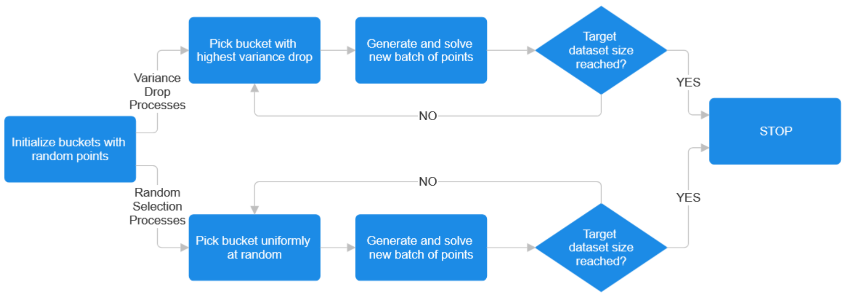

Initialize buckets with random points
Variance Drop Processes
Pick bucket with highest variance drop
Generate and solve new batch of points
Target dataset size reached?
YES
NO
STOP
Random Selection Processes
Pick bucket uniformly at random
Generate and solve new batch of points
Target dataset size reached?
NO
YES

Figure 1: Data generation flowchart

Figure 1 shows a visual summarization of the data generation procedure we follow. Instead of first generating the configurations to be used as regression inputs and then finding the associated C&C cost for each of them to be regression targets in one go, we generate the dataset using a bunch of generate-solve-repeat cycles running in parallel, generating configurations and associated cost values. We use this setup to build the training points and the associated labels one at a time, and build the dataset gradually. The intent behind this procedure is to first generate an initial batch of sample $(\boldsymbol{\sigma}, \mathcal{C}(\boldsymbol{\sigma}))$ points, and then conduct the rest of the data generation process in a way that encourages exploring the search space in an efficient way and direct the data point generation effort to regions of the search space that need more attention, i.e., have more variance in the solution values for different configurations.

To facilitate the data generation process as described, we first divide the search space into *buckets*, considering the "length" (i.e., number of operational STL lines) of points contained in each bucket. That is, bucket-n contains data points that contain $(5(n-1), 5n]$ operational STLs (bucket-$n = \{\boldsymbol{\sigma} : 5(n-1) < \boldsymbol{\sigma}^T\boldsymbol{\sigma} \leq 5n\}$). Each bucket tracks the variance of the labels it contains, as well as the "variance drop" (i.e., the percent change in variance) resulting from the latest batch of sample points added to the bucket. This constitutes a measure of how efficiently the subspaces induced by each bucket have been explored, as well as the informative value of the last batch of points that were sampled from each bucket. When a generate-solve-repeat loop terminates an iteration, it conducts a check of all buckets to see which bucket-n had the highest variance drop in the latest batch of points. A high change in percent variance indicates the last batch of points had significant contributions in this bucket; we use this measure as a proxy of how informative the next batch of points will be. When the procedure identifies this bucket, it randomly generates points that would fit in this bucket (i.e., have the relevant number of STLs) and restarts the loop. This way, the data generation process is guided by incentivizing smart exploration. In addition to this procedure, several generate-solve-repeat loops are also operating by choosing their buckets uniformly at random; to avoid cases where some buckets remain underpopulated, for example, due to sticking to "local maximum" variance drops. In our preliminary experiments, we have observed that when the training data is generated with the procedure described above, our PACFNN model yielded favorable prediction performance. Prompted by these promising early results, we utilized the above method for the data generation phases of all computational experiments.

4.1.2 Deep Learning Model

We propose a polynomial activation constrained feed-forward neural network model (PACFNN) with activation function x^2 to estimate $\mathcal{C}(.)$. To achieve a convex optimization problem for embedding, as will be described later on, we limit the number of hidden layers to two, and the weights of the last layer are forced to be nonnegative. Let the width of the hidden layers be H_1 and H_2, respectively. We formulate the forward pass of the network for a configuration $\boldsymbol{\sigma}$:

$$a_l^{(1)} = \left(b_l^{(1)} + \sum_{j=1}^{|L|} \boldsymbol{\sigma}_j W_{jl}^{(1)}\right)^2 \qquad \forall l \in \{1, \ldots, H_1\} \tag{1}$$

$$a_k^{(2)} = \left(b_k^{(2)} + \sum_{l=1}^{H_1} W_{kl}^{(2)} a_l^{(1)}\right)^2 \qquad \forall k \in \{1, \ldots, H_2\} \tag{2}$$

$$\hat{\mathcal{C}}(\boldsymbol{\sigma}) = a^{(3)} = b^{(3)} + \sum_{k=1}^{H_2} W_k^{(3)} a_k^{(2)} \tag{3}$$

$$W_k^{(3)} \geq 0 \qquad \forall k \in \{1, \ldots, H_2\} \tag{4}$$

Polynomial functions are not typically the first choice for neural network activation, as their use precludes the universal approximation guarantees enjoyed by other common activations (Leshno et al. 1993). In our setting, however, this limitation does not apply: as we explain shortly, power functions can also provide universal approximation guarantees, since the target function is defined on a finite, discrete domain. Also, when neural networks are employed as surrogates within mathematical optimization models, prediction accuracy is only part of the story: the structural complexity they introduce into the embedded problem is at least as important. In this respect, we will show that polynomial activations are a viable option for binary-input regression consecutively embedded in an integer programming formulation. With the proposed construct, we can approximate the C&C costs as a function of the binary decisions regarding the inclusion of an STL line in the solution, and embed this approximation into the optimization model through linear and quadratic constraints.

Against this backdrop, it is worth noting that in the literature, ReLU networks are used as surrogates, building on their popularity in neural network architectures. However, their integration into mathematical programs is typically achieved through reformulations with big-M constraints for each hidden-layer node (Grimstad and Andersson 2019), which are well known to make the resulting optimization problem considerably harder to solve (Fischetti and Jo 2018). In our numerical experiments, we compare the proposed PACFNN model with ReLU networks in terms of both prediction performance and computational efficiency of the embedded optimization model. The results of our numerical experiments in Section 5 show that PACFNN achieves slightly better prediction accuracy and, at the same time, delivers a substantial advantage in solution times for the resulting mathematical program.

Following up on our remark about the universal approximation guarantees for the NN with power activation functions, note that the function we aim to approximate is defined on a discrete (more importantly, finite) domain. Then we can argue that a neural network that does not only approximate this function, but in fact computes it exactly, exists; simply by being wide enough to accommodate a neuron for every possible discrete input in its hidden layer - essentially mapping the input vector L to its value. To align this "neural network as a mapping" idea closer to our specific application, we should show that such a mapping neural network *with the power function as its activation function* exists as well. The following result provides the theoretical basis for our choice of polynomial activation functions together with nonnegativity constraints on the output layer.

Proposition 1. *Let $\pi : \{0,1\}^n \to [0,\infty)$ be a function defined on a discrete and finite domain. There exists a neural network with activation function x^2, of finite width and depth, with all nonnegative entries in the weights of the output layer, which can compute $\pi(x)$ exactly.*

Proof. We show the existence of this network by construction. To this end, we first express the function π as a sum of *monomials*. For a binary input $L \in \{0,1\}^n$, let L^+ denote the index set of entries of L which are equal to one. Then, for each $L \in \{0,1\}^n$, we define the corresponding monomial as follows:

$$m_L(x) = \prod_{i \in L^+} x_i \tag{5}$$

where $x \in \{0,1\}^n$, and define $m_{\mathbf{0}} = 1$. Notice for a nonzero L, since $|x| = |L|$, we must have $m_L(x) = 1 \iff x = L$. Thus, we can construct a function g to represent π through monomials as follows:

$$g(x) = \sum_{L \in \{0,1\}^n} m_L(x)\pi(x) \tag{6}$$

Assume the output layer is of width 2^n and encodes every value π takes over its domain $\{0,1\}^n$. That is, letting $B_n(x) = \sum_{i=1}^n 2^{(n-i)} x_i$ denote the decimal expansion of binary vector x, assume the weights of the output layer W are characterized by $W_{B_n(x)} = \pi(x)$. We note that by definition of π, the vector W is nonnegative. To show the existence of a neural network that achieves the computation in (6) it suffices to show that we can express monomials as linear combinations of $(w^T l + b)^2$ terms. Letting $|L^+|$ denote the number of ones in L, this can be shown through the following argument.

For $|L^+| = 1$, letting i denote the index of the only positive entry of L we have $(e_i^T l)^2 = (l_i)^2 = l_i$ where the final equality follows from the fact that L is binary, therefore $l_i^2 = l_i$, concluding first-order monomials can be expressed through such functions.

For $|L^+| = 2$, assuming $L^+ = \{i, j\}$, we can construct $l_i l_j$ simply by taking $w = e_i + e_j$ and noting that $(w^T l)^2 = l_i^2 + l_j^2 + l_i l_j = l_i + l_j + l_i l_j$. The desired monomial $l_l l_j$ appears in this expression, and the remaining are first-order monomials, which we have shown to be obtainable by linear combinations of hidden layer outputs earlier; concluding the case of $|L^+| = 2$ can be covered by such a network as well.

Using this construction shows that we can find linear combinations of a single output layer with polynomial activations to express monomials of order up to 2. For monomials of order greater than 2; we note that for two binary vectors $L, K \in \{0,1\}^n$, the expression $((m_L(x) + m_K(x))^2$ will contain a monomial of order $|L^+ \cup K^+|$ (this expression in general will not necessarily be of the form we present in (5), but recall that we can drop the exponents for binary variables). As such, using first and second order monomial outputs from one hidden layer, we can extend the network with another hidden layer that enables constructing third and fourth order monomials (the lower-order monomials in the resulting expression can be negated by using the outputs of previously constructed layers). Another layer will extend the span of these expressions to monomials of order eight, and so on. Generally, this construction enables computing monomials of order n with $\mathcal{O}(log_2 n)$ hidden layers, where the final output layer will be a simple linear combination for each input.

$\square$

We finally note that it might be the case that the target function π might not be nonnegative everywhere. However, the above framework can also be used to approximate such a function, despite the earlier restriction that the weights of the output layer should take their values from the image of π and should be nonnegative. Simply finding the minimum value of π on its domain, say Λ and approximating $\pi - \Lambda$ also suffices: since the two prediction tasks are equivalent we can conclude such neural networks can approximate any function defined on a finite binary domain under the very mild assumption that it is never identically $-\infty$ anywhere. It can be very difficult to compute Λ exactly, but any valid lower bound also guarantees nonnegativity.

The idea presented above only shows the existence of an NN of finite width and depth adhering to our restrictions, which can also compute π exactly. Since the network is of exponential size, this result is not very applicable in practice. However, the key insight is that an NN with our restrictions and arbitrarily good approximation performance *does* exist, which motivates the idea of using a similarly constructed, but practical sized network, in predicting the function π. This idea is also validated experimentally, the empirical results obtained with a practical-sized network can be seen in Section 5.3.

4.2 Integer Programming Formulation

We make use of a pre-trained neural network with the architecture described as in Section 4.1 to estimate the cost reduction provided by the line configuration $\boldsymbol{\sigma}$, and embed it a mathematical programming model to jointly decide both line configuration and routing of the STL dedicated vehicles. To this end, we introduce the STL Routiong (STL-R) problem in the following section.

We formulate STL-R on the time-extended network ensuring that every line segment implied by the line configuration $\boldsymbol{\sigma}$ is traversed by a vehicle. Instead of directly embedding the C&C costs $\mathcal{C}(\boldsymbol{\sigma})$ into the model, we instead embed the pre-trained surrogate model into STL-R to compute the *estimated* C&C costs, denoted $\hat{\mathcal{C}}(\boldsymbol{\sigma})$. As such, we can embed the predicted crowdshipping cost into a mathematical programming model by embedding the activation function into the constraints. To account for the routing: we designate *source* nodes at time 0 and *sink* nodes at time T, where vehicles can start and end their daily routes. We optimize routes along these nodes, while ensuring that edges chosen by the line configuration $\boldsymbol{\sigma}$ are visited by at least one vehicle.

We finally note that the chosen STL schedule $\boldsymbol{\sigma}$ is also a decision that should be output by STL-R. To reflect this behavior, we introduce binary variables ℓ indicating whether STL $\ell_n \in L$ is operational.

Table 1: Notation for STL-R

Sets	
$\mathcal{V}$	Set of nodes of the time extended network
$\mathcal{E}$	Set of edges of the time extended network
S	Set of source nodes ($S \subseteq \mathcal{V}$)
T	Set of sink nodes ($T \subseteq \mathcal{V}$)
L	set of candidate STL locations along the time extended network (represented as edges).
Parameters	
h	Fixed cost associated with dispatching a vehicle
p_{ij}	Traversal cost for one vehicle along an arc $(i, j) \in E$
$W^{(i)*}$	Weights of the i-th layer of the trained neural network.
$b^{(i)*}$	Biases of the i-th layer of the trained neural network.
H_i	Width of the hidden layer i of the network
Variables	
ℓ_n	Binary variable denoting whether a candidate STL $n \in D$ will be opened.
x_{ij}	Integer variable denoting the number of vehicles traveling along arc $(i, j) \in \mathcal{E}$
$a^{(i)}$	Auxiliary variable keeping track of the neural network output from i-th layer.

With the notation as defined in Table 1, we formulate STL-R as follows:

$$\min f(x, \ell) = a^{(3)} + \sum_{s \in S} \sum_{(i,j) \in \delta^+(s)} h x_{ij} + \sum_{(i,j) \in E} p_{ij} x_{ij} \tag{7}$$

$$\text{s.t. } a_l^{(1)} = \left(b_l^{(1)*} + \sum_{m=1}^{|L|} \ell_m W_{ml}^{(1)*} \right)^2 \qquad \forall l \in \{1, \ldots, H_1\} \tag{8}$$

$$a_k^{(2)} = \left(b_k^{(2)*} + \sum_{l=1}^{H_1} W_{kl}^{(2)*} a_l^{(1)} \right)^2 \qquad \forall k \in \{1, \ldots, H_2\} \tag{9}$$

$$a^{(3)} = b^{(3)*} + \sum_{k=1}^{H_2} W_k^{(3)*} a_k^{(2)} \tag{10}$$

$$\sum_{(i,j) \in \delta^+(i)} x_{ij} = \sum_{(j,i) \in \delta^-(i)} x_{ji} \qquad \forall i \in \mathcal{V} \setminus (S \cup T) \tag{11}$$

$$x_{ij} \geq \ell_n \qquad \forall (i, j) \in L, (i, j) = \ell_n \tag{12}$$

$$x_{ij} \in \mathbb{Z}^+, \qquad \forall (i, j) \in \mathcal{E} \tag{13}$$

$$\ell_n \in \{0, 1\} \qquad \forall \ell_n \in L \tag{14}$$

The objective is to minimize the total expected delivery cost arising from crowdshipping and courier operations (represented by the first two terms, notice the output of the third layer $a^{(3)}$ is equal to the prediction of the neural network for the STL schedule $\boldsymbol{\sigma}$ induced by L, $\hat{\mathcal{C}}(\boldsymbol{\sigma})$), and the cost of dispatching and routing vehicles (represented by the last two terms). Constraints (8)-(10) encode the output of the neural network. Constraints (11) are the flow balance constraints. Constraints (12) ensure that if an edge is selected in a dedicated line configuration, at least one vehicle traverses it. Finally, constraints (13) are variable domains.

The model represented in (7)-(13) is nonconvex, due to constraints (8) and (9) which denote nonlinear equalities. Constraint (8) involves a quadratic expression with binary variables ℓ. Expanding this expression, we get:

$$a_l^{(1)} = (b_l^{(1)*})^2 + 2b_l^{(1)*} \sum_{m=1}^{|L|} W_{ml}^{(1)*} \ell_m + \sum_{m=1}^{|L|} (W_{ml}^{(1)*})^2 \ell_m^2 + 2 \sum_{m=1}^{|L|} \sum_{n=1}^{m-1} W_{ml}^{(1)*} W_{nl}^{(1)*} \ell_m \ell_l \tag{15}$$

The above is still not linear in decision variables: it contains squared (ℓ_m^2) and multiplied ($\ell_m\ell_l$) decision variables. Both of these nonlinearities can be amended by using the fact that ℓ are binary variables, we note that $x^2 = x$ for any binary x and the multiplication of two binary variables can be linearized by introducing an auxiliary binary variable. Therefore, we introduce the binary variables $\ell_{ml} = \ell_m\ell_l$ and represent constraint (15) as follows:

$$a_l^{(1)} = (b_l^{(1)*})^2 + 2b_l^{(1)*}\sum_{m=1}^{|L|} W_{ml}^{(1)*}\ell_m + \sum_{m=1}^{|L|}(W_{ml}^{(1)*})^2\ell_m + 2\sum_{m=1}^{|L|}\sum_{n=1}^{m-1} W_{ml}^{(1)*}W_{nl}^{(1)*}\ell_{ml} \tag{16}$$

$$\ell_{ml} \le \ell_m \tag{17}$$

$$\ell_{ml} \le \ell_l \tag{18}$$

$$\ell_{ml} \ge \ell_m + \ell_l - 1 \tag{19}$$

Where constraint (16) is linear in decision variables, and the remaining constraints (17)-(19) represent the relationship $\ell_{ml} = \ell_m\ell_l$ with linear constraints. We can thus replace the constraint (8) with constraints (16)-(19) to obtain a linear representation for it.

Constraint (9) is trickier to work with, as it involves a quadratic expression again but the decision variables on the right hand side are not binary this time. Breaking the equality constraint into a less-than and a greater-than constraint does not work here; since the equality is nonlinear, the resulting constraints are generally not convex. To circumvent this issue, we make use of the fact that during the training phase of the neural network we constrained the weights $W^{(3)}$ to be nonnegative. This enables replacing constraint (9) with:

$$a_k^{(2)} \ge \left(b_k^{(2)*} + \sum_{l=1}^{H_1} W_{kl}^{(2)*}a_l^{(1)}\right)^2 \qquad \forall k \in \{1,\ldots,H_2\} \tag{20}$$

This is because since $W^{(3)}$ are nonnegative, minimizing the value of $a^{(3)}$ in constraint (10) forces the model to pick the minimum value for all $a_k^{(2)}$. As such, constraint (20) alone would be binding and the corresponding less-than constraint is not required. Since the right hand side of (20) is convex, this constraint is also convex.

Having done the necessary transformations as described above, we obtain a convex Mixed Integer Quadratic Programming model. The final model is as follows:

$$\begin{aligned}&\min f(x,d)\\ &\text{ s.t.}(10)-(14),(16)-(20)\end{aligned}$$

We finally note the following two properties of STL-R, which can be used to enhance the computational performance when solving an instance.

Remark 1. The flow balance constraints (11) are written for the entire arc set. Given the sparsity of candidate line locations across the time-extended network, one can obtain a reduced edge set as follows: Consider only the edge set induced by the set of candidate dedicated lines, and connect the ends of these edges to the source and sink nodes. Next, we connect the candidate edges as follows. Consider two candidate dedicated lines along the spatio-temporal points $(o_1, t_1), (d_1, \bar{t}_1)$ and $(o_2, t_2), (d_2, \bar{t}_2)$, and assume $t_2 > t_1$. We connect these lines with a new arc $(d_1, \bar{t}_1), (o_2, t_2)$ if the shortest path between d_1 and o_2 is time-feasible, *i.e.*, if the shortest path between d_1 and o_2 can be traversed in $t_2 - \bar{t}_1$ time units. The length of this new edge is also the length of the shortest path, as in an optimal solution where a vehicle visits multiple dedicated lines, it must do so through the shortest path since the contrary case would yield higher routing costs and therefore contradict optimality.

Remark 2. Fix any $\ell \in \{0,1\}^{|L|}$ and solve the model with this fixed value of d. We observe that for fixed ℓ, the model reduces to a minimum-cost flow problem with lower bounds. It is well known that for such problems a transformation exists that yields an equivalent formulation with a totally unimodular constraint matrix (Wolsey 1998), implying that the integrality constraints on the x variables can be relaxed. Since this property holds for any ℓ, the relaxed model will always produce integer values for the x variables at optimality. Critically, this structure makes decomposition approaches such as Benders' Decomposition applicable. In this study, however, we do not exploit such methods, as the problem instances we consider can be solved to optimality directly—highlighting the computational efficiency of the embedding approach we employ.

We finally note that, as the combined framework uses a surrogate estimator to compute the value of the objective function, the final STL-R formulation does not guarantee an optimal solution to the overall problem. The deviation from the optimal solution can be calculated depending on the performance of the surrogate estimator embedded in the model. A common statistical assumption, made not only for mathematical tractability but also as an indicator of good model fit, is that the regression residuals are zero-mean Gaussian random variables (Hwang and Ding 1997). In supervised learning contexts—including neural networks—when a model is well specified, residuals should exhibit a random scatter around zero, often approximating a normal distribution; any systematic pattern or bias may signal underfitting, overfitting, or omitted relationships (Verma 2025). In what follows, under this normality assumption, we provide an expectation on the optimality gap.

Proposition 2. *Let E be the random variable denoting the optimality gap between the optimal solution given by the combined model, and the true optimum. Assuming the prediction errors of the embedded estimator are i.i.d. Gaussian with mean 0 and standard deviation σ, we have $\mathbb{E}[E] < 1.13\sigma$.*

Proof. Let $\hat{f}(x, \ell)$ denote the optimal value output by the model. Suppose the true optimum occurs at some $(x^\star, \ell^\star)$ with the prediction value being $\hat{f}(x^\star, \ell^\star) = f(x^\star, \ell^\star) + \varepsilon(\ell^\star)$. Further assume the model determines some (x, ℓ) as the optimal solution with $\hat{f}(x, \ell) = f(x, \ell) + \varepsilon(\ell)$. Since the model determines (x, ℓ) as optimal, we have:

$$\hat{f}(x^\star, \ell^\star) \geq \hat{f}(x, \ell) \tag{21}$$

$$\implies f(x^\star, \ell^\star) + \varepsilon(\ell^\star) \geq f(x, \ell) + \varepsilon(\ell) \tag{22}$$

$$\implies \varepsilon(\ell^\star) - \varepsilon(\ell) \geq f(x, \ell) - f(x^\star, \ell^\star) \tag{23}$$

Thus, the optimality gap is upper bounded by some random variable with the distribution $\varepsilon(\ell^\star) - \varepsilon(\ell) \sim Gaussian(0, 2\sigma^2)$ since prediction errors are i.i.d. $Gaussian(0, \sigma^2)$. Further, since we know the true optimum is $(x^\star, \ell^\star)$, we have:

$$f(x^\star, \ell^\star) \leq f(x, \ell) \tag{24}$$

$$\implies \varepsilon(\ell^\star) \geq \varepsilon(\ell) \tag{25}$$

Therefore, we know that $E \sim Gaussian(0, 2\sigma^2)$ and $E > 0$. Then, the expectation is:

$$\mathbb{E}[E|E > 0] = \frac{1}{\mathbb{P}[E > 0]} \int_0^\infty \frac{x}{\sqrt{4\pi\sigma^2}} \exp(-\frac{1}{2}\frac{x^2}{4\sigma^2})dx = \frac{2}{\sqrt{\pi}}\sigma \simeq 1.128\sigma \tag{26}$$

□

The above proposition shows how the expected error behaves depending on the performance of the estimator. The crux of the proof above is the observation that the deviation from optimality can be represented with a truncated Gaussian random variable. While the above analysis exploits this fact to compute the expected error, one can also extend this analysis with standard techniques from stochastic analysis to obtain probabilistic bounds on the error, like Chebyshev-type tail inequalities.

5 Computational Studies

In this section, we outline the setup of our experimental case study in Istanbul and report our results. In the following subsections, we specify the set of parameter used throughout the computational studies (Section 5.1), outline the details of our experimental design procedure (Section 5.2), examine the prediction performance of the introduced surrogate model (Section 5.3), and discuss the managerial insights derived from the solution of STL-R (Section 5.4).

5.1 Problem data and parameters

In our experiments, we study the determination of the STL schedules for the upcoming month, where the package and crowdshipper flows through the month are computed based on real-world data provided from our industry partner Trendyol (trendyol.com) and other publicly available sources. In this section, we detail the data we use and how the problem parameters were derived from them.

Service network: The service region under consideration is İstanbul, with 110 locations selected across the city to serve as pickup and drop-off points for packages. These locations were originally determined in (Kızıl and Yıldız 2023) and subsequently employed in (Şardağ et al. 2024). As the present study extends the setting of the latter work, we retain the same service network.

Each day, packages enter the system starting at 09:00 AM, and the cutoff time for accepting packages for same-day delivery is 03:00 PM. Deliveries are executed between 09:00 AM and 06:00 PM, resulting in a nine-hour daily planning horizon. Note that delivery operations continue beyond the cutoff to complete the service of packages that arrive close to 03:00 PM. This operation interval is discretized into 10-minute periods in our time-extended formulation.

We determined 100 STL candidates. To determine the STL candidates, we first run discrete-time event simulations across the delivery network where no STLs are opened and only couriers and crowdshippers are operational. The arcs accommodating the highest volumes of package flow on average in this case constitute the initial candidate set, which is then refined to eliminate similar candidates (e.g., same physical location at close time periods). This indicates we build our mathematical models with $|L| = 100$. This parameter is also relevant when generating our training data as described in Section 4.1.1; in this step, we opt to partition the search space into 20 subspaces so we have 20 buckets in total.

Package transfer demand: To model delivery demand, we use real-world transfer data provided by our industry partner, Trendyol, consisting of more than 1.5 million records. Each record specifies the origin, destination, and order placement time of a transferred package. We assume that 10% of this package flow is suitable for an express delivery setting, which corresponds to roughly 5,300 packages per day to be shipped through the CS+STL network. In our analysis, we vary this percentage between 5% and 30% to examine the impact of demand size on the proposed delivery model. As a base case, we impose a three-hour delivery time guarantee, and we additionally test two-hour and four-hour guarantees to assess how sensitive system performance is to this critical service quality measure.

Courier costs: We assume that a courier has a fixed cost of 50 liras per package and a variable cost of five liras per km, following the current industry practices of companies providing similar services in the area (Lider Kurye 2023).

Routing costs: For STLs, a vehicle has a fixed cost of $119,000$ liras per month which is calculated based on current (as of time of writing) vehicle rental service prices, and a variable cost of 10.66 liras per km calculated based on current fuel prices and other additional maintenance costs (Basma et al. 2022). It is worth emphasizing that the courier per-km price is only incurred for the length of the courier trip, while the vehicle per km price is in effect for the entire routing process including the deadhead between service points. To solve the routing problem, we consider two depots on the European and Asian sides of Istanbul for STL vehicles to depart from and return to.

Crowdshipper Data: To model the crowdshipper flows, we generate hourly commuter rates along origin-destination pairs from the IMM Open Data Portal (2012) data.

All components of this study including data generation, the solution algorithm to compute the C&C costs, and the simulation procedures are coded in a Python 3.9 environment. For the solution of the MIQP model, the Python wrapper of the Gurobi solver was utilized. All scripts were run on a Linux workstation with dual Intel Xeon Gold 5418Y processors and 512GB of RAM.

5.2 Experimental design

With the problem data and parameters as described in the previous section, we construct our computational experiment set as follows. We begin by defining a *base case* to constitute a baseline for our experiments, with the following parameter values: A crowdshipper stock (number of available commuters that can volunteer as crowdshippers) of 150,000 people, corresponding to roughle 0.5% of daily commuters in Istanbul; and a delivery lead time τ of 180 minutes to accommodate for an express delivery setting.

We first use this base case setting to evaluate our proposed system from an operational perspective. To this end, we first solve an instance of STL-R with the base case parameters to optimality, and use the resulting STL configuration as an input to a discrete-time event simulation procedure to monitor the movement of packages across the delivery network in this setting. This serves as a feasibility check for the proposed system, and showcases how frequently and efficiently STL delivery is used and how the routing and pricing decisions change in response to STL line activations.

To evaluate the STL system from an economic standpoint, we again make use of a discrete time event simulation procedure. In addition to the base case, we also introduce a *no-line* case where no STLs are operational - this enables us to determine the marginal effect of implementing STLs on top of an operational CS network. For a more in-depth analysis of the economic benefits of the system, we also examine how the fiscal performance evolves with respect to resource availability. To this end, we design different sets of experiments, each designated to investigate the impact of a distinct parameter on the system performance. Table 2 summarizes the setups for different experimental scenarios for convenient reference. In the experiment set E1, we investigate how the number of available crowdshippers would impact the system performance; by investigating varying values of CS stock between $\{75, 150, 225\}$ thousand people. In the experiment set E2, the impact of delivery lead time is being assessed by perturbing the value of τ between $\{120, 180, 240\}$ minutes. Finally, in experiment set E3, we investigate how the delivery load impacts the system performance; by varying the number of packages in the system. We earlier asserted that in the base case we assumed 10% of all packages available in the data would be eligible for an express delivery setting. To obtain varying package loads across the delivery network, we perturb this eligibility rate.

5.3 Estimator Performance

As discussed in Section 4, a distinctive feature of our work is the introduction of PACFNN as a surrogate function in place of the ReLU networks that constitute the benchmark from the literature. In this section, we present our experimental results comparing the two architectures in terms of both prediction accuracy and the computational efficiency of the resulting optimization model.

Our dataset consists of 60,000 datapoints for the base case and between 40,000 and 60,000 points for the cases examined in Subsection 5.4. We divide each generated dataset into 90% training set, 5% validation set, and 5% test set.

Table 2: Parameter Settings for Experimental Cases

Experiment ID	τ (minutes)	CS stock (thousand people)	Daily Packages (thousand)
BC	180	150	5300
E1.1	180	75	
E1.2		225	5300
E2.1	120	150	
E2.2	240		5300
E3.1			2600
E3.2			7900
E3.3	180	150	10600
E3.4			13200
E3.5			15900

Before turning to a detailed analysis of the prediction power of the proposed NN architecture and its ability to identify high-quality solutions for the joint problem, we first examine its learning curve to assess data requirements. Figure 2 shows how the root mean square error (RMSE) changes with varying training set sizes. We see that the RMSE decreases sharply with the initial increase in data, then stabilizes once the dataset reaches about 20,000–25,000 samples. Beyond this point, additional data provides only marginal improvements, with the error converging to a low and stable level, indicating that the network achieves strong predictive performance with a moderate amount of training data.

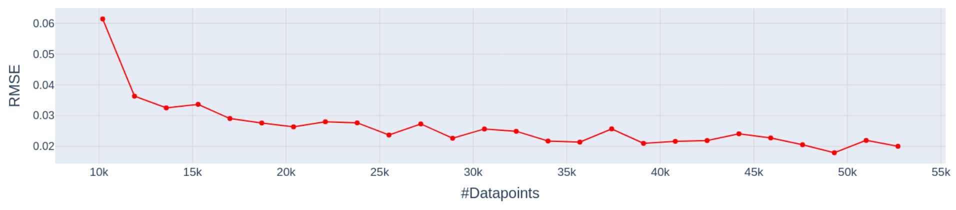


Figure 2: Attained test RMSE for varying training dataset sizes.

One of the critical hyperparameters in the NN models is the size of the hidden layers, H_1 and H_2. Increasing these sizes enhances the approximation capacity. However, larger hidden layers also increase the computational effort required to train the model and, more importantly, in our case of using an NN as a surrogate, enlarge the joint optimization model. Specifically, the number of equations and decision variables grows with the sizes of the hidden layers, making the optimization problem more computationally demanding. Therefore, the hidden layers should be large enough to ensure good predictive performance but not larger.

For simplicity, we set $H_1 = H_2$. Table 3 reports the test RMSEs of the PACFNN and ReLU models for different hidden layer sizes. Based on these results, we set the nodes to 256 ($H_1 + H_2 = 256$) for our computational studies, as it provides sufficiently accurate predictions, while further increases yield only marginal improvements.

The weights were initialized using the Xavier method. To enforce the non-negativity constraint in the output weights $W^{(3)}$, we replace (3) and (4) with $\hat{\mathcal{C}}(\boldsymbol{\sigma}) := b^{(3)} + \sum_{k=1}^{H_2} \left| W_k^{(3)} \right| a_k^{(2)}$ at the forward pass; the gradient is calculated accordingly at the backward pass.

5.3.1 Prediction accuracy

Table 4 reports the prediction performance of the proposed PACFNN surrogate in comparison with the ReLU network benchmark from the literature and a simple Ordinary Least Squares (OLS) regression. We evaluate the models using Root Mean Squared Error (RMSE) and Mean Absolute Percentage Error (MAPE).

Table 3: Test RMSEs for varying hidden layer sizes.

size (H_1+H_2)	PACFNN	ReLU
64	0.0219	0.0282
128	0.0169	0.0217
256	0.0142	0.0257
512	0.0153	0.0335

Table 4: Test metrics of selected models.

Metric	PACFNN	ReLU	OLS
RMSE	0.0142	0.0257	0.2488
MAPE (%)	0.122	0.214	2.231

The results indicate that PACFNN achieves the best prediction accuracy across both metrics. Specifically, PACFNN attains an RMSE of 0.014 and a MAPE of 0.12%, outperforming ReLU (0.026 and 0.21%, respectively) and substantially surpassing OLS (0.25 and 2.23%). These improvements, while modest in absolute terms when comparing PACFNN and ReLU, demonstrate that the use of polynomial activation functions does not compromise predictive accuracy and, in fact, yields slightly superior results. The large performance gap relative to OLS further underscores the necessity of nonlinear surrogates in this setting.

Overall, these findings establish that PACFNN provides prediction accuracy at least as strong as ReLU—our benchmark from the literature—while laying the foundation for the computational efficiency gains demonstrated in the following subsection.

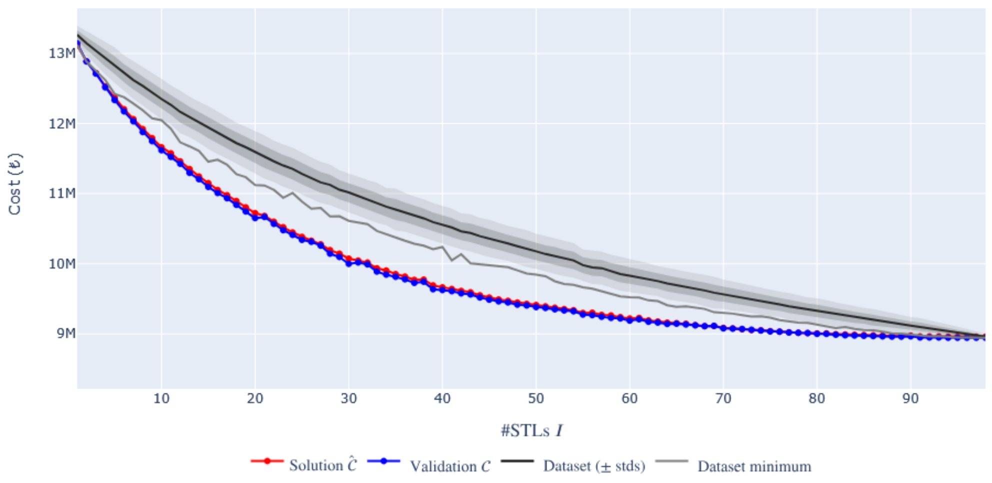


Figure 3: *Cardinality constrained coverage* solutions.

Having established that PACFNN delivers very high prediction performance, we now turn to the more fundamental question of whether this predictive power actually helps the optimizer identify high-quality STL configurations. To investigate this, we compare the solutions generated by the optimizer with those in the training dataset, which was constructed by randomly selecting STL lines to open. To simplify the comparison, we slightly modify the optimization model by removing the routing costs (which are directly addressed by the optimizer without using the surrogate) and including a limit on the number of lines to open, as follows.

$$\min \hat{\mathcal{C}}(\boldsymbol{\sigma}) = a^{(3)} \tag{27}$$

$$\text{s.t. } (10), (13)(14), (16) - (20)$$

$$\sum_{n=1}^{|L|} \boldsymbol{\sigma}_n = I \tag{28}$$

Figure 3 shows the results. The black curve represents the average cost in the training dataset (with shaded bands showing one standard deviation), while the red line reports the estimated costs $\hat{\mathcal{C}}$ from PACFNN and the blue line gives the validated costs $\mathcal{C}$ obtained by calculating the C&C costs for the given schedules as explained in the Appendix A. Across all cardinalities $I = 1, \ldots, 100$, the PACFNN-guided solutions consistently yield costs that are well below the dataset average but also significantly outperform the dataset minimum.

These results clearly demonstrate that PACFNN effectively translates its regression performance into optimization power by guiding the search toward STL configurations that no random sampling of lines could uncover, validating its role as a reliable surrogate in the joint optimization framework.

5.3.2 Computational efficiency

Figure 4 compares the computational efficiency of the optimization models obtained with PACFNN and ReLU surrogates across different hidden-layer widths, showing the optimality gap (%) for a run time limit of one hour.

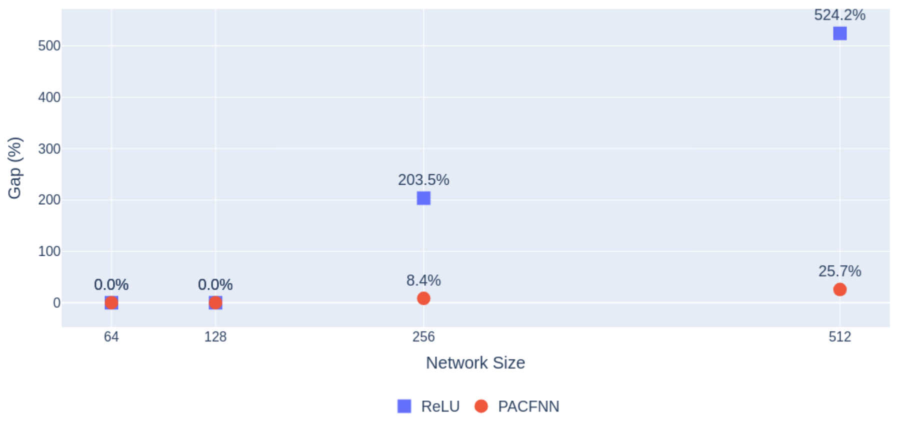


Figure 4: MIP gaps after one hour of solution time.

The results highlight a clear distinction between the two architectures. For small network sizes (64 and 128), both models yield solutions with negligible gaps, indicating that the optimization problem remains tractable for either surrogate. However, as the width increases, the differences become substantial. With 512 nodes, the ReLU-based model exhibits an average gap of 203.5%, compared to only 8.4% for PACFNN. This divergence grows more pronounced at larger widths: for 512 nodes, the gap grows to 524.2% with ReLU, versus 25.7% with PACFNN.

These findings underscore the merits of our approach: while PACFNN achieves slightly better predictive accuracy than ReLU, as discussed in the previous subsection, its computational advantage is drastic. The convex embedding enabled by polynomial activations yields optimization models that remain solvable, even as network complexity increases, whereas the big-M reformulations required for ReLU rapidly become intractable.

We conclude this section by examining the runtimes of STL-R for the problem instances studied, as reported in Table 5. All instances are solved to optimality within roughly two hours. In most cases, runtimes are comparable across experiments, with the exception of E1.1 and E2.2, which require relatively longer solution times. Interestingly, these are also the settings where the benefits of STL lines are most pronounced, suggesting that the added consolidation opportunities make the routing problem more challenging. In E1.1, reduced crowdshipper availability increases the reliance on STLs to cover demand, while in E2.2, the four-hour service guarantee allows more packages to accumulate for transfer, thereby amplifying the role of STLs in the solution.

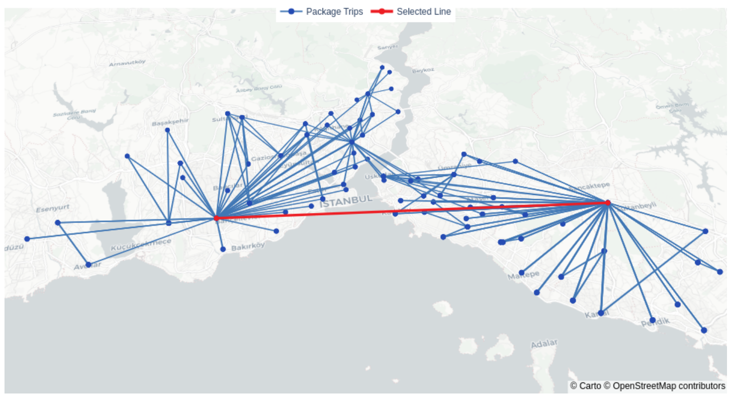
5.4 Managerial Insights

In this section, we analyze the solution of the joint routing model (STL-R), to derive key managerial insights. Recall that the central motivation of this study is to address the lack of consolidation opportunities in conventional crowdshipping systems, where deliveries are handled by individual couriers in a highly fragmented manner. Our contribution is to augment crowdshipping with STLs, which provide high-volume, low-cost transfer corridors. By enabling the bulk transfer of packages between strategically chosen locations at critical times, STL provides a systematic means to create and exploit consolidation opportunities within crowdshipping networks. The efficacy of STLs is dependent on how much package build-up occurs "naturally", as indicated by how many packages would travel along a given STL arc in a CS-only system anyway. In addition to this bulk of packages, when an STL is opened, it has the potential to attract packages from other nearby service points that might not necessarily be routed through this path in the absence of an STL. Motivated by this idea, we first begin by examining the utilization of STLs in order to assess how efficiently STL-based order consolidation can be carried out.

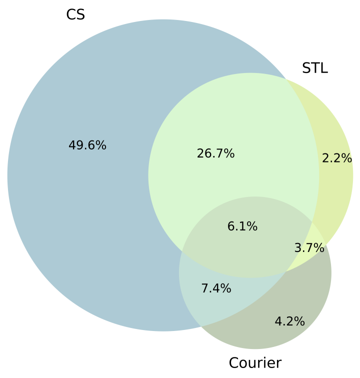
Table 5: MIP results of STL-R instances.

Instance	Solution Time (s)	#STLs	#Vehicles	Expected Saving
BC	4258	39	8	14.8%
E1.1	5622	37	8	16.2%
E1.2	2492	33	7	12.9%
E2.1	3919	24	5	3.2%
E2.2	6468	32	6	22.1%
E3.1	2558	21	4	7.4%
E3.2	4628	46	10	18.8%
E3.3	3397	52	12	21.4%
E3.4	3990	57	13	23.0%
E3.5	3219	61	14	24.1%

Table 5 shows the number of STLs operated, with respect to the expected number of packages per day. In the base case where 5,300 packages are transferred daily, 39 of 100 candidate STLs are opened and eight vehicles are operated in total to traverse these STLs. As open STLs have the potential to affect a significant portion of the operations of the corresponding hours, we carry on to analyze the package circulation across the network in the base case to measure how the STL system gets utilized. Figure 5a shows the "attraction area" of a sample STL line opened in the optimal solution for the base case, where the red line shows the STL itself and the rest of the lines in the figure show the routes of packages that use this specific STL as a part of their delivery trip. As shown in Figure 5a, STLs might connect two areas that collect packages from/send packages to a wide distribution of service points. This confirms the earlier assertion that operational STLs can alter the spatio-temporal dynamics of package delivery significantly, as evidenced by the breadth of the influence area of the sample STL.As for overall STL utilization, Figure 5b shows a breakdown of all the different delivery modes (CS, courier or STL) used. In the base case, 38.7% of all packages use at least one STL in their



(a) Package circulation of an STL.



(b) Modes of transfer of all packages.

Figure 5: Overview of package flows.

delivery process, and almost 90% of all packages make at least one trip with a crowdshipper. This indicates that both these alternative transfer modes are fairly attractive, and the addition of STLs captures a significant portion of the package flow, serving once again as a confirmation of the widespread usage and efficiency of the proposed STL solution. Compared to a typical crowdshipper-only delivery system where no STL lines are operated, the base case also provides a 33.16% reduction in total courier mileage, meaning employing the proposed delivery system would also be an environmentall friendly choice.

Seeing as STLs can generate significantly wide collection/distribution ranges at their endpoints and more than a third of all packages being delivered through the CS network make use of at least one STL during delivery processes, we can conclude that a CS system is indeed rife with opportunities for order consolidation and STLs seem to be providing a remarkable solution for this need.

We now turn to the question of economic efficiency: there is an opportunity for significant improvement in the system, but what would be the benefit of enacting such a system would be to a platform operator? Besides the C&C and routing costs, the system is also implicitly constrained by two additional resources: the crowdshipper stock (i.e. how much of the public is willing to act as potential crowdshippers) and the delivery lead time (which might provide or prohibit flexibility in routing decisions). For a more comprehensive evaluation, we assess the economic performance of our proposed system with respect to the availability of these two implicit resources. In what follows, cost results under cases in which the promised lead time τ and the total number of expected crowdshippers in the system varies will be examined comparatively.

Crowdshipper stock: The effect of the change in the number of crowdshippers is evident in cost terms. We evaluate the monetary savings compared to the *no-line* case as described in Section 5.2. A detailed breakdown of cost components can be seen in (Figure 6). We see that in the base case, the addition of STLs can reduce the operational costs by 15.09%, indicating significant cost savings can be achieved on top of the crowdsourced delivery system. When the number of crowdshippers increase, we also see a significant portion of the savings are generated from courier costs. A somewhat counterintuitive result is that we observe that STL savings tend to increase in relative terms when there are fewer crowdshippers available to the system: this is because a majority of the delivery operations fail in the absence of crowdshippers and courier services are used heavily. Conversely, when the number of crowdshippers increase, most deliveries do not need to resort to couriers; so the relative savings generated by STLs are not as prominent in this case. This pattern can also be seen in the absolute costs: while the case with 225k crowdshippers has the smallest relative savings, in absolute terms it still incurs the lowest costs because the abundance of crowdshippers makes deliveries easy and cheap.

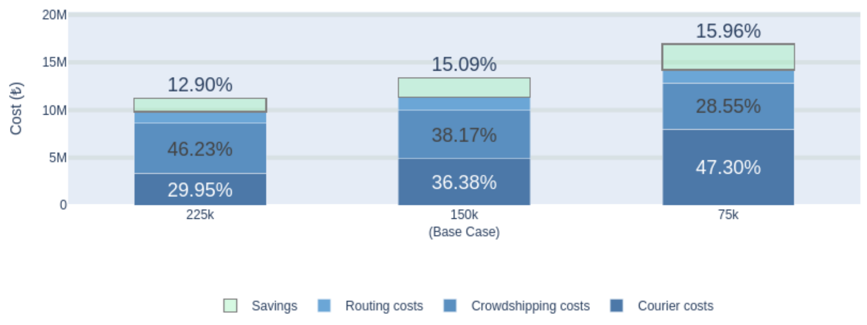


Figure 6: Cost components under varying number of crowdshippers in the system.

Delivery lead time: In terms of operational costs, increasing delivery times tend to reduce the operational costs but the diminishing returns pattern seen in the crowdshipper counts is also apparent here. This can be explained by very short lead times being punishingly restrictive; whereas reasonably long delivery times can accommodate a large portion of the delivery demand through CS network. In line with this observation, we also see that the majority of the cost composition is made of CS costs for longer delivery times. A detailed analysis of the cost components can be seen in Figure 7.

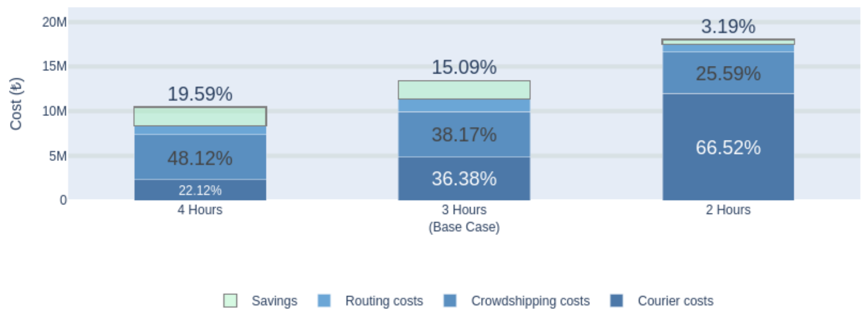


Figure 7: Cost components under varying promised delivery lead time.

In summary, as expected, the economic performance of the system is generally enhanced when more resources become available at its disposal; as an increase in crowdshippers and delivery lead times would provide with more flexibility both in terms of cost and routing. The extent to which cost savings can be provided is dependent on the resources available to the system; but at any rate, the proposed system in the base case can shave off more than 10% operational costs by introducing STLs to the delivery network. We would also like to reiterate that this final cost saving is calculated in comparison to a system with CS deliveries but no STLs. This has two implications: intuitively, the overall costs savings compared to a traditional delivery system would naturally be much higher; and more importantly, while 10% cost saving is already a significant achievement, it should also be kept in mind that this saving is attained on top of an already optimized system, underlining how crucial of a defect order consolidation really is for individual contractor based crowdshipper systems.

6 Conclusion and Future Work

The present study aims to enhance a crowdshipping delivery network by deterministically operating transfer lines placed across strategic spatiotemporal points in the delivery network, to facilitate easier and efficient order consolidation. We observe that significant additional cost savings can be achieved: depending on the resources (time and potential crowdshippers) available, additional cost savings of 3.19%-19.59% can be achieved. The network management problem that is needed to solve to find this outcome is very complex and likely intractable when modeled through classical means. To overcome this challenge, we devise a new solution framework that first generates data for resulting costs for alternative design structures and then uses an ML surrogate to obtain a functional form that takes design decisions as arguments and outputs the resulting operational costs. For the ML surrogate, we introduce a novel deep learning model architecture for solving discrete problems that can be efficiently embedded into convex optimization models. We observe that this new model can be solved efficiently, and scales much better than classical surrogates due to being modeled mostly with continuous variables. We also examine theoretical properties of our proposed framework, and present stochastic bounds on the deviation of the surrogate model from the true optimum with respect to the strength of the estimator.

This study makes use of a somewhat unconventional neural network architecture, which enables easier embedding into convex optimization models: an interesting research direction would be investigating other potential architectures that would enable convexification of deeper neural networks with more hidden layers. The present study also uses a surrogate to estimate the solution for a stochastic DP for which any configuration of design variables yield a feasible solution: yet another interesting research direction would be to investigate how deep learning models can be used as surrogates for constrained models to obtain a broader range of applicability across network design problems and other commonplace problems where binary design variables naturally arise. Finally, for the problem tackled in this study, obtaining training data for the surrogate model was efficient due to the guided data generation process we employ, but also due to the specific subproblem structure we were able to exploit: for generic problems where training label generation might not be as practical, design of data-efficient surrogates remain critical.

This study has been supported by the European Research Council (ERC) under grant number 101076231 as a part of the GoodMobility project (https://goodmobility.ku.edu.tr). Furthermore, this work would not have been possible without the help of the Trendyol Operations and Innovation Department. We thank them not only for providing us with real-life data but also for the many informative and insightful discussions.

References

N. Aftabi, N. Moradi, and F. Mahroo. Feed-forward neural networks as a mixed-integer program. *Engineering with Computers*, pages 1–19, 2025.

R. Alizadeh, J. K. Allen, and F. Mistree. Managing computational complexity using surrogate models: a critical review. *Research in Engineering Design*, 31(3):275–298, 2020.

A. Anderer, H. Bastani, and J. Silberholz. Adaptive clinical trial designs with surrogates: When should we bother? *Management science*, 68(3):1982–2002, 2022.

C. Archetti, M. Savelsbergh, and M. G. Speranza. The vehicle routing problem with occasional drivers. *Eur. J. Oper. Res.*, 254(2):472–480, Oct. 2016.

A. M. Arslan, N. Agatz, L. Kroon, and R. Zuidwijk. Crowdsourced delivery—a dynamic pickup and delivery problem with ad hoc drivers. *Transp. Sci.*, 53(1):222–235, Feb. 2019.

H. Basma, F. Rodríguez, J. Hildermeier, and A. Jahn. Electrifying last-mile delivery: A total cost of ownership comparison of battery-electric and diesel trucks in europe. 2022.

Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

L. Bertazzi, R. Mogre, and N. Trichakis. Dynamic project expediting: a stochastic shortest-path approach. *Management Science*, 70(6):3748–3768, 2024.

R. Bornatico, J. Hüssy, A. Witzig, and L. Guzzella. Surrogate modeling for the fast optimization of energy systems. *Energy*, 57:653–662, 2013.

H. Buldeo Rai, S. Verlinde, J. Merckx, and C. Macharis. Crowd logistics: an opportunity for more sustainable urban freight transport? *European Transport Research Review*, 9(3):39, 2017.

Capgemini. Navigating the complex web of last-mile deliveries, 2023. URL `https://www.capgemini.com/insights/expert-perspectives/navigating-the-complex-web-of-last-mile-c` Accessed: 2025-08-02.

Q. Chen, Y. Lei, and S. Jasin. Real-time spatial–intertemporal pricing and relocation in a ride-hailing network: Near-optimal policies and the value of dynamic pricing. *Operations Research*, 72(5): 2097–2118, 2024.

I. Dayarian and M. Savelsbergh. Crowdshipping and same-day delivery: Employing in-store customers to deliver online orders. *Production and Operations Management*, 29(9):2153–2174, 2020.

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

K. Gdowska, A. Viana, and J. P. Pedroso. Stochastic last-mile delivery with crowdshipping. *Transportation research procedia*, 30:90–100, 2018.

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

L. J. Hong and X. Zhang. Surrogate-based simulation optimization. In *Tutorials in Operations Research: Emerging Optimization Methods and Modeling Techniques with Applications*, pages 287–311. INFORMS, 2021.

J. G. Hwang and A. A. Ding. Prediction intervals for artificial neural networks. *Journal of the American Statistical Association*, 92(438):748–757, 1997.

IMM Open Data Portal. Istanbul transportation master plan household survey, 2012. URL `https://data.ibb.gov.tr/en/dataset/6cefaa5b-bd1b-4e98-a27e-94ddac1ecd2b/resource/1fa02d91-f794-4`

K. U. Kızıl and B. Yıldız. Public transport-based crowd-shipping with backup transfers. *Transportation Science*, 57(1):174–196, 2023.

M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Netw.*, 6(6):861–867, Jan. 1993.

Lider Kurye, 2023. URL `https://www.liderkurye.com.tr/`.

G. Macrina, L. Di Puglia Pugliese, F. Guerriero, and D. Laganà. The vehicle routing problem with occasional drivers and time windows. In *International conference on optimization and decision science*, pages 577–587. Springer, 2017.

G. Macrina, L. D. P. Pugliese, F. Guerriero, and G. Laporte. Crowd-shipping with time windows and transshipment nodes. *Computers & Operations Research*, 113:104806, 2020.

V. V. Mišić. Optimization of tree ensembles. *Operations Research*, 68(5):1605–1624, 2020.

S. S. Mohri, N. Nassir, R. G. Thompson, and P. S. Lavieri. Public transportation-based crowd-shipping initiatives: Are users willing to participate? why not? *Transportation Research Part A: Policy and Practice*, 182:104019, 2024.

A. Punel and A. Stathopoulos. Modeling the acceptability of crowdsourced goods deliveries: Role of context and experience effects. *Transportation Research Part E: Logistics and Transportation Review*, 105:18–38, 2017.

T. Raviv and E. Z. Tenzer. Crowd-shipping of small parcels in a physical internet. 2018.

B. Rosenhahn. Optimization of sparsity-constrained neural networks as a mixed integer linear program. *J. Optim. Theory Appl.*, 199(3):931–954, Dec. 2023.

A. Şardağ, K. U. Kızıl, and B. Yıldız. Crowdshipping problem with dynamic compensations and transshipments. *Transportation Research Part C: Emerging Technologies*, 166:104796, 2024.

A. M. Schweidtmann and A. Mitsos. Deterministic global optimization with artificial neural networks embedded. *Journal of Optimization Theory and Applications*, 180(3):925–948, 2019.

X. Sun and X. Zhu. Dynamic control of a make-to-order system under model uncertainty. *Management Science*, 2025.

C. Tsay. Sobolev trained neural network surrogate models for optimization. *Computers & Chemical Engineering*, 153:107419, 2021.

V. Verma. A comprehensive framework for residual analysis in regression and machine learning. *Journal of Information Systems Engineering & Management*, 10(31s):1–18, 2025. doi: 10.52783/jisem.v10i31s.4958. Open access.

L. A. Wolsey. *Integer programming.* John Wiley & Sons, 1998.

D. Yang, P. Balaprakash, and S. Leyffer. Modeling design and control problems involving neural network surrogates. *Computational Optimization and Applications*, 83(3):759–800, 2022.

B. Yıldız. Express package routing problem with occasional couriers. *Transportation Research Part C: Emerging Technologies*, 123:102994, 2021.

B. Yıldız and M. Savelsbergh. Provably high-quality solutions for the meal delivery routing problem. *Transportation Science*, 53(5):1372–1388, 2019a.

B. Yıldız and M. Savelsbergh. Service and capacity planning in crowd-sourced delivery. *Transportation Research Part C: Emerging Technologies*, 100:177–199, 2019b.

A Calculating the Crowdshipping and Courier Costs

As indicated in Section 3, each CS is assumed to carry a single package, with compensation offers made independently for each package in the system. Şardağ et al. (2024) shows that relaxing this assumption has only a minimal impact on solution quality for the problem instances we study in this paper, yet this simplification allows routing and compensation decisions to be made independently for each package. In our case, this simplification enables computing the expected C&C costs under a given STL schedule $\boldsymbol{\sigma}$ by first calculating the expected cost per package and then aggregating over the expected demand.

We use dynamic programming to find the expected cost of a package. To formalize the algorithmic steps, we make the following definitions.

- **State variable** s is a tuple $\langle i_s, j_s, t_s, \bar{t}_s \rangle$ with current location $i_s \in V$, destination $j_s \in V$, current time $t_s \in T$, and delivery deadline $\bar{t}_s$.
- **Actions** that can be taken at a state s are denoted by $A(s)$, which depends on current location i_s and time t_s. We consider four groups of actions in $A(s)$.
  - (i) For each station $j \in V \setminus \{i_s\}$, the set of *crowdshipper call* actions $A_{CS}(s)$ includes an action $a_{i_s k} > 0$, which denotes the compensation offered to a crowdshipper for transferring the package to location k. This action is available only if the transfer by a crowdshipper is feasible, that is, if the condition $t + \delta_{i_s k} + \bar{\delta}_{k j_s} \leq \bar{t}$ holds.
  - (ii) The *courier call* action $\hat{a}_s$ represents handing over the package to a courier for delivery to the destination s_t.
  - (iii) For each STL ℓ in the given schedule $\boldsymbol{\sigma}$, with origin $\ell_o = i_s$ and starting time $\ell_t = t_s$, there exists an *STL action* a_ℓ available at state s. The set of all STL actions available at state s is denoted by $A_\sigma(s)$.
  - (iv) The *waiting action* represents keeping the package at its current location for one time period. This action is available only if waiting is feasible—that is, the package would still have enough time to reach its destination in the worst case via direct transfer. Mathematically, this condition is given by $(t_s + 1) + \bar{\delta}_{i_s j_s} \leq \bar{t}_s$.
- **Transition function** $\mathcal{S}(.)$ takes a state and action pair (s, a) and produce the next state $\hat{s}$. Depending on the type of the action a, the following transitions occur.
  - (i) If a is a crowdshipper call, i.e., $a \in A_{CS}(s)$, with origin i_s and destination k:
    - With probability $F^{t_s}_{i_s,k}(a)$, a crowdshipper arrives to pick up the package, and the next state becomes:
$$\mathcal{S}(s, a) = \langle k, j_s, (t_s + \delta_{i_s k}), \bar{t}_s \rangle.$$
    - With probability $(1 - F^{t_s}_{i_s,k}(a))$, no crowdshippers arrive. The package remains at its current location and advances in time to the next period. The next state becomes:
$$\mathcal{S}(s, a) = \langle i_s, j_s, (t_s + 1), \bar{t}_s \rangle.$$
  - (ii) If a is a courier call, the package is taken to its destination by the paid courier and the next state becomes $\mathcal{S}(s, a) = \langle j_s, j_s, (t_s + \bar{\delta}_{i_s j_s}), \bar{t}_s \rangle$.
  - (iii) If a is an STL action associated with the line L, the package is transferred to its next stop L_d and the next state becomes $\mathcal{S}(s, a) = \langle \ell_d, j_s, \ell_a, \bar{t}_s \rangle$.
  - (iv) If a is a waiting action, the package stays at its current location and the next state becomes $\mathcal{S}(s, a) = \langle i_s, j_s, (t_s + 1), \bar{t}_s \rangle$.

- **Cost function** $C(.)$ takes a state and action pair (s, a) and returns the immediate expected cost as follows.

$$C(s,a) = \begin{cases} \alpha F^t_{i_s,k}(\alpha) & \text{if } \alpha \text{ is crowdshipper call form station } i_s \text{ to a station } k \\ \hat{a}_{i_s j_s} & \text{if } \alpha \text{ is a courier call} \\ 0 & \text{if } \alpha \text{ is STL} \\ 0 & \text{if } \alpha \text{ is a waiting action} \end{cases} \tag{29}$$

We define the value functions with the Bellman equations as follows.

$$V(s) = \min_{\alpha \in \mathcal{A}(s)} \{C(s, \alpha) + \mathbb{E}[V(\mathcal{S}(s, a))]\} \tag{30}$$

Considering that the system does not permit late deliveries—and will resort to an on-demand courier service if necessary to ensure timely completion—we define the following boundary conditions for the value functions.

$$V(\langle i_s, j_s, t, \bar{t}_s\rangle) = \infty \qquad \forall t \in \{\bar{t}_s + 1, ..., t_n\} \tag{31}$$

$$V(\langle j_s, j_s, t, \bar{t}_s\rangle) = 0 \qquad \forall t \in \{t_0, ..., \bar{t}_s\} \tag{32}$$

Minimization over action set in (30) induces both finding optimal compensations for the CS option for all current stop-next stop pairs and selection among discrete alternatives, including the courier, waiting, and STL options, if there are any. Finding the optimal compensation for an individual CS transfer (i, k) at time t is the following problem, which is solved in (30) implicitly:

$$\min_{\alpha \geq 0} \{F^t_{i_s k}(\alpha)(V(\langle k, j_s, (t + \delta_{i_s k}), \bar{t}_s\rangle) + \alpha) + (1 - F^t_{i_s k}(\alpha))V(i_s, j_s, (t + 1), \bar{t}_s\rangle)\} \tag{33}$$

Remark 3. Having a continuous and strictly positive pdf $f^t_{i_s k}(\alpha)$ of the cdf $F^t_{i_s k}(\alpha)$ for the valid domain α, the first-order condition suffices to find a^* for (33). The derivative $\frac{d}{d\alpha}(F^t_{i_s k}(\alpha)(c_1 + \alpha) + (1 - F^t_{i_s k}(\alpha))c_2)$ has at most one root if the hazard rate is monotonic, indicating the minimum occurs either at the root or one of the endpoints of the range of α.

Using this, we can reduce (30) from a problem with a continuous action space into a problem with a discrete action space, by fixing the compensation for the crowdshipper call. We also note that when the state space is defined as described above and action space is discretized, the problem of finding the minimizing action of (30) reduces to a *stochastic shortest path* problem. Moreover, since moving from a state to any possible "next state" also embeds information about the passage of time, it is possible to embed the state space into a directed acyclic graph (contrary case would suggest it is possible to "move backwards" in time). This specific structure enables solving for the expected cost of any state very efficiently; by starting from the "last" possible state with the maximal time where the expected value is known through the boundary conditions, and iterating over the possible previous states in time-reverse order. We refer the reader to Bertazzi et al. (2024) for the explicit description and implementation details of this algorithm. For an individual package, this dynamic programming approach has a complexity of $\mathcal{O}(\tau|V|)$. The complete solution approach we devise requires the *aggregate cost*, meaning this procedure should be run for every package coming into the system. Thus, we make this calculation for all possible values of the package-state $V(\langle i, d_p, t_p, t_p+\tau\rangle)$. After having computed the expected state values, we find the expected C&C cost for some given STL schedule $\boldsymbol{\sigma}$ as follows:

$$\mathcal{C}(\boldsymbol{\sigma}) = \sum_{\forall i \in V} \sum_{\forall j \in V} \sum_{t=t_0}^{t_n} \varepsilon^t_{ij} V(\langle i, j, t, t + \tau\rangle) \tag{34}$$

The problem instances we study consider a Poisson process to model the interarrival times (Şardağ et al. 2024). Let $\lambda^t_{i_s,k}$ be the rate of crowdshipper arrivals along arc (i,k) at time t, and the ratio of offer-accepting arrivals for until the next decision epoch could be determined by a continuous, nondecreasing attraction rate function $r(.)$. Then (33) takes the form:

$$\min_{\alpha\geq 0}\{(1-e^{-r(\alpha)\lambda^t_{i_s k}})(V(\langle k,j_s,(t+\delta_{i_s k}),\bar{t}_s\rangle)+\alpha)+e^{-r(\alpha)\lambda^t_{i_s k}}V(i_s,j_s,(t+1),\bar{t}_s\rangle) \tag{35}$$

Furthermore, again using the same parameters as in Şardağ et al. (2024) we use a piecewise linear attraction rate function $r(\alpha)$ with offers that have the values between 10 and 50 liras:

$$r(\alpha)=\frac{\alpha-10}{40} \tag{36}$$

Let c_1 and c_2 be the value functions $V(\langle k,j_s,(t+\delta_{i_s k}),\bar{t}_s\rangle)+\alpha)$ and $V(i_s,j_s,(t+1),\bar{t}_s\rangle)$ respectively, for brevity. Then the equation to solve takes the form:

$$\alpha^*=\underset{50\geq\alpha\geq 10}{\arg\min}\left\{(1-e^{(\frac{10-\alpha}{4}\lambda^t_{i_s k})})(c_1+\alpha)+e^{(\frac{10-\alpha}{4}\lambda^t_{i_s k})}c_2\right\} \tag{37}$$

The function (37) is either convex or concave, since the second derivative does not change sign anywhere. Thus, to find the minimizer, one can check three points: the point α^{**} where the first derivative of (37) is 0, (the convex minimizer), or the endpoints 10 and 50 (the possible concave minimizers). As such, the minimum of (37) occurs at one of $\{10,\alpha^{**},50\}$ and the minimizer can be found rapidly by evaluating the function at these three points. The explicit formula for α^{**} is given by:

$$\alpha^{**}=\frac{-4\omega(1-\frac{1}{4}(c_1-c_2)\lambda^t_{i_s k}-\frac{5}{2}\lambda^t_{i_s k})-(c_1-c_2)\lambda^t_{i_s k}+4}{\lambda^t_{i_s k}} \tag{38}$$

where $\omega(.)$ is the Wright omega function. In cases where the true minimizer α^{**} is outside the interval $[10,50]$, we also check endpoints to ensure the cost generated by the policy is within the operational bounds.