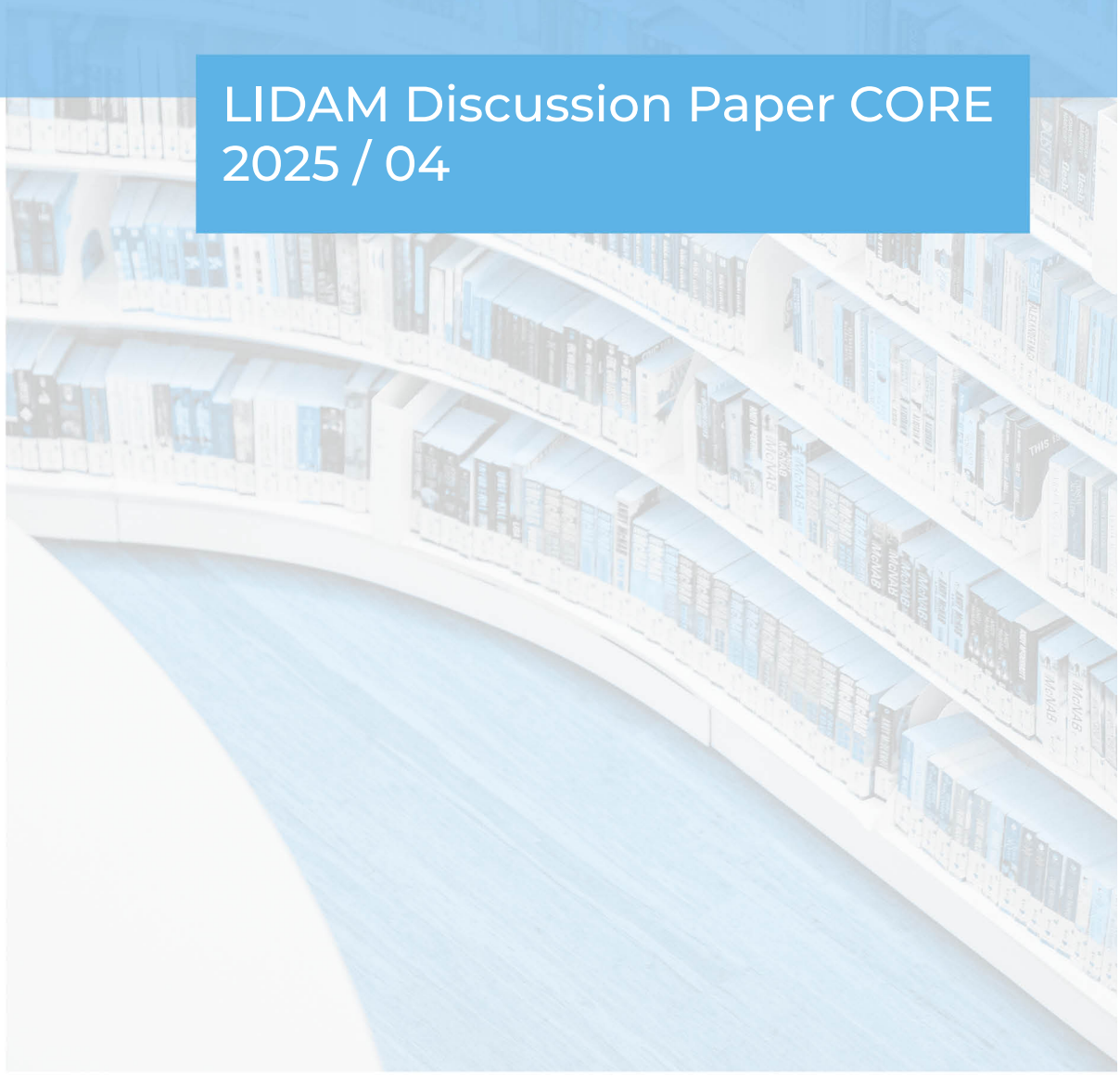
TRANSFER REINFORCEMENT LEARNING FOR PRICING, DRIVER REPOSITIONING AND CUSTOMER ADMISSION IN RIDE-HAILING NETWORKS

Thomas De Munck, Jean-Sébastien Tancrez, Philippe Chevalier

LIDAM Discussion Paper CORE
2025 / 04



UCLouvain

CORE | LIDAM
Louvain Institute of Data Analysis and Modeling in economics and statistics

CORE

Voie du Roman Pays 34, L1.03.01

B-1348 Louvain-la-Neuve

Tel (32 10) 47 43 04

Email: immaq-library@uclouvain.be

https://uclouvain.be/en/research-institutes/lidam/core/core-discussion-papers

Transfer Reinforcement Learning for Pricing, Driver Repositioning and Customer Admission in Ride-Hailing Networks

Thomas De Munck*[1], Jean-Sébastien Tancrez[1], and Philippe Chevalier[1]

[1]Center for Operations Research and Econometrics (CORE), Université catholique de Louvain, Voie du Roman Pays, 34, 1348, Louvain-la-Neuve, Belgium.

February 2025

Abstract

We consider the problem of a ride-hailing platform (e.g., Uber, Lyft) that connects supply with demand over a network of locations. To this aim, the platform makes pricing, driver repositioning, and customer admission decisions. Customers are impatient and have distinct willingness to pay. Drivers can be repositioned by the platform, or can choose to relocate to other locations by themselves. We formulate this problem as a discrete-time Markov decision process and propose a transfer learning approach to find an efficient policy. Our approach first derives a rolling-horizon strategy by repeatedly solving a deterministic optimization problem. Then, two neural networks are pretrained to replicate the strategy and learn the associated value function. Finally, the policy is further improved through deep reinforcement learning (DRL). Using data from New York City, we apply our approach to networks of up to 20 locations. The results show that our approach outperforms alternative DRL algorithms and rolling-horizon strategies while reducing computation time and stabilizing learning. We also explore the interplay between pricing, driver repositioning, and customer admission, providing insights into their respective roles.

Keywords: Transportation; Ride-hailing platforms; Pricing and repositioning decisions; Transfer learning; Deep reinforcement learning.

Declaration of interests: none.

1 Introduction

Over recent years, ride-hailing platforms (e.g., Uber, Lyft, or Bolt) have emerged as prominent components of urban mobility. These platforms typically operate in a city characterized by a network of locations in which drivers move around according to customer requests. They differ from ride-sharing (e.g., Blablacar) or car-sharing platforms (e.g., Cambio) in that customers do not drive the car but are served by drivers dedicated to the service.

*Corresponding author, thomas.demunck@uclouvain.be.

When it comes to managing ride-hailing networks, balancing driver supply with customer demand is particularly challenging due to four main reasons. A first challenge is related to the *heterogeneity of the demand*, both in time and space. That is, the expected demand changes from one location to another (e.g., business district vs. residential areas), and depends on the time of the day (e.g., morning vs. evening peak hours). Second, ride-hailing platforms must deal with *interdependent supply and demand*, as satisfied customer requests affect the current availability of drivers in origin locations and the future availability of drivers in destination locations. Third, the users of ride-hailing platforms tend to exhibit *strategic behaviors*. On the one hand, customers determine whether to request services depending on their willingness to pay. On the other hand, drivers may decide to self-relocate to different locations due to higher expected earnings and their own preferences. Lastly, these platforms offer *on-demand services* rather than scheduled services. Customer requests must be served in real time, or they will be lost. This implies that the platforms need to rebalance their network within short periods.

Given these issues, ride-hailing platforms employ multiple decision levers to connect supply and demand. In this study, we consider three practical decisions: pricing, driver repositioning, and customer admission. These decisions occur at the tactical level (e.g., every five minutes), as they are mainly used to balance aggregate flows of customers and drivers in the network. We distinguish them from operational decisions, which are made at a higher spatio-temporal granularity (e.g., driver dispatching, every thirty seconds). The pricing decision influences demand in time and space by pushing customers with low willingness to pay out of the system. It also impacts supply, as drivers tend to self-relocate to locations with higher prices (e.g., being guided by the platform through heatmaps, Uber, 2023). The repositioning decision affects supply, ordering drivers to reposition to other locations. It is expected to become prevalent with the rise of autonomous vehicles, and is already experienced by some ride-hailing platforms (Jiao et al., 2021). Lastly, the customer admission decision, either applied directly (e.g., warning that no driver is available) or indirectly (e.g., announcing long waiting times), allows further control of demand. This decision occurs when driver supply is insufficient to serve all the requests, and determines which customers are served based on their destinations. It is observed in practice, especially in markets where dynamic pricing is negatively perceived (Kanoria & Qian, 2024).

The mentioned decisions are often optimized separately by the literature, without accounting for potential interactions (cf. Section 2). However, as underlined by Qin et al. (2022), this can lead to inefficiencies since locally optimal decisions can be suboptimal for the whole system. For example, prices can be adjusted to reduce demand in a location, even though drivers could have easily been repositioned at the location in the first place. The goal of this paper is therefore to model these three decisions and evaluate their respective effectiveness in balancing supply and demand. For this purpose, we develop a model formulated as a Markov decision process in which a ride-hailing platform determines the prices, the drivers to reposition, and the customers to admit in a network of locations over a finite time horizon. Besides the platform's repositioning decision, the model also allows all the drivers to relocate according to their own choices. Impatient customers arrive according to non-stationary Poisson processes, and decide whether to request service based on their willingness to pay. In this setting, the platform's objective is to find a dynamic policy that maximizes the revenues for serving customers at a specific price while minimizing the costs incurred by repositioning drivers and leaving some requests unsatisfied.

Due to the size and complexity of the problem, our model cannot be solved to optimality with conventional methods such as dynamic programming. Even advanced reinforcement learning algorithms take excessively long time to

reach policies that are still underperforming (Section 5.2). We thus deploy a transfer learning methodology that blends mathematical optimization with deep reinforcement learning (DRL). Our original approach finds an efficient policy in three steps. First, a rolling-horizon strategy is derived by repeatedly solving multiple optimization problems based on the expected values of stochastic parameters. Second, two neural networks are pretrained to reproduce the rolling-horizon strategy, and to approximate its value function. Third, the policy is further improved using DRL. We demonstrate that this approach computes an improved policy, using less computational effort in comparison with alternative methods such as rolling-horizon strategies and DRL algorithms used alone. The resulting policy is then analyzed to explore the relevance of pricing, driver repositioning, and customer admission.

The main findings of our research can be summarized as follows.

- First, we introduce a stochastic model that optimizes pricing, driver repositioning, and customer admission. To the best of our knowledge, our work is the first attempt to integrate these three levers of decision in a single mathematical model. We do so in a comprehensive setting that includes spatio-temporal demand imbalances, stochastic demand and transit times, and independent drivers who can elect to self-relocate.
- Second, we develop an efficient approach that combines mathematical optimization with DRL. In numerical experiments, we show that our approach increases profit by 16.7% and 4% on average, compared to alternative rolling-horizon strategies and DRL algorithms, while being 42%-65% faster than DRL algorithms. Our approach also benefits from stable learning, allowing us to handle instances with many zones and extended horizons.
- Third, we provide managerial insights on how pricing, driver repositioning, and customer admission help cope with supply-demand imbalances. In particular, we find that both pricing and driver repositioning are necessary to mitigate imbalances, while customer admission substitutes driver repositioning when driver supply becomes too scarce to be repositioned. Our results also suggest that driver repositioning by the platform is more effective than drivers who relocate by themselves.

The rest of the paper is structured as follows. Section 2 discusses related literature, highlighting our contributions. Section 3 defines the problem, and formulates it as a Markov decision process. Section 4 first provides some background, and then presents our transfer learning approach. Section 5 presents numerical experiments to show the performance of our approach and bring out managerial insights. Section 6 concludes and suggests future research avenues.

2 Literature Review

Our work is related to the recent literature on ride-hailing platforms and reinforcement learning. In this section, we first present these two literature streams, and then outline our contributions.

2.1 Ride-Hailing Platforms

In the ride-hailing literature (reviewed by H. Wang & Yang, 2019), we focus on research about pricing, driver repositioning, and customer admission, either studied separately or together.

A significant number of papers examine the implications of pricing on supply and demand using single-location models (see, e.g., Bai et al., 2019, Sun et al., 2019). Other studies investigate how pricing can mitigate supply-demand

imbalances in multiple locations. Bimpikis et al. (2019) develop a model in which a ride-hailing platform sets prices in multiple locations to influence the entry choice of customers and drivers, and the self-relocating behavior of drivers. Li et al. (2021) and Banerjee et al. (2022) extend this research by examining alternative network structures, objective functions, and system constraints in related settings. In these studies, a common goal is to search for a pricing policy that is optimal in steady state. However, since real-world applications are dynamic and stochastic, they can lead to overly simplified problems. Similar to us, Nourinejad and Ramezani (2020), and Meskar et al. (2023) address this shortcoming by considering dynamic pricing under non-stationary arrival rates. Nevertheless, the former focuses on temporal pricing while we also include spatial pricing; the latter consider deterministic demand while we capture stochastic customer arrivals.

Driver repositioning has also garnered extensive attention in the literature on ride-hailing platforms. On the one hand, many studies examine the repositioning decision as a *centralized* decision made by the platform. In this setting, early works derive static policies with analytical properties (e.g., Braverman et al., 2019), while recent works deduce dynamic policies that are more actionable in practice (e.g., Hosseini et al., 2024, Beojone et al., 2024). On the other hand, several studies consider driver repositioning as a *decentralized* decision made by drivers (called driver self-relocation in our work). Following this view, a few papers explore the use of subsidies (Zhu et al., 2021), or information sharing (Beojone & Geroliminis, 2023) to incentivize drivers to relocate. Some papers, like the one by Shou et al. (2020), also address the repositioning problem at the individual driver level. All the mentioned papers suppose that the repositioning decision is either achieved by the platform or by drivers, not both. We take a broader perspective by developing a model where both platform-repositioning and driver-repositioning can occur. By doing this, we can compare and evaluate the two repositioning mechanisms.

Only limited research has investigated customer admission despite its practical value. De Munck et al. (2023) study customer admission decisions for a ride-hailing platform serving multiple customer classes with differentiated services. G. Wang et al. (2024) use a spatial queueing model to approximate the customer pickup time, and to admit customers whose pickup time is inferior to some threshold. Kanoria and Qian (2024) propose dynamic admission policies that do not require any knowledge of the customer arrival rates. These works mainly cover customer admission in a single location, without considering its integration with pricing and driver repositioning.

A few papers have jointly analyzed multiple decision levers on ride-hailing platforms. Notably, the repositioning and dispatching (also called matching) decisions are sometimes optimized together, by solving an integer optimization model (Alonso-Mora et al., 2017, Tuncel et al., 2023). There also exist papers that integrate pricing with dispatching decisions in equilibrium models (e.g., Özkan, 2020). Closer to our work, Afèche et al. (2023) analyze driver repositioning and customer admission in a stylized queueing network. They find that customer admission can induce drivers to relocate to high-demand locations, and suggest that further research should examine the interactions with pricing under non-stationary arrival rates. Q. Chen et al. (2023) propose a policy that can adjust prices and reposition drivers dynamically in a ride-hailing network with non-homogeneous arrivals rates. Yet, their policy does not consider customer admission, drivers with self-relocating behavior, and stochastic transit times. Their methodology also differs from ours, as they obtain a policy using mathematical optimization while we combine optimization with reinforcement learning.

2.2 Reinforcement Learning

We next discuss relevant literature on reinforcement learning by discussing recent developments in deep reinforcement learning (DRL), transfer learning, and applications to ride-hailing platforms.

The literature on reinforcement learning is extensive, dating back to the seminal work of Barto et al. (1983), and others in the late '80s. Following recent advances in deep learning, reinforcement learning benefits from a renewed interest, which has led to multiple successful applications (e.g., Silver et al., 2016). Several works, like those by Haarnoja et al. (2018), and Schulman et al. (2017), develop stable algorithms that can deal with continuous action spaces. Our work relies on Proximal Policy Optimization (PPO), an algorithm proposed by Schulman et al. (2017).

In contrast to traditional DRL methods, we leverage mathematical optimization in a pretraining process to enhance the overall training process. Our approach can be related to transfer learning, an umbrella of methods (see Taylor & Stone, 2009 for a review) that exploit external knowledge to make the learning process faster and more effective. Although promising, these methods are still underexplored in DRL, especially when applied to operations management problems (Boute et al., 2022). To date, the few successful applications have addressed inventory optimization problems. De Moor et al. (2022) design a deep Q-network algorithm guided by a modified reward term (the shaped reward) to mimic a given heuristic policy. As a result, the training process becomes faster and more stable. In the same vein, Oroojlooyjadid et al. (2022) solve the Beer Game with a deep Q-network algorithm. In their approach, the neural network of an agent is used to initialize the neural network of another agent, saving computational effort. Like the method proposed by Oroojlooyjadid et al. (2022), our approach uses pretrained neural networks. Unlike their method, the parameters of our neural networks are initialized from an optimization problem rather than from another trained agent. Furthermore, while these works apply transfer learning to systems with discrete actions, our approach can be employed for systems with continuous actions.

Closest to our research are studies applying reinforcement learning to ride-hailing platforms (see Qin et al., 2022 for a review). However, many of them consider a single driver as the primary agent (e.g., Jiao et al., 2021, Qian et al., 2022). Only a few of them optimize ride-hailing systems in a centralized manner, considering the platform as the main agent. Among those, C. Chen et al. (2021) use the PPO algorithm to optimize continuous prices in a network of locations. Al-Kanj et al. (2020), Mao et al. (2020), and Kullman et al. (2022) investigate dispatching problems for autonomous vehicles, and employ value function approximation, policy-gradient, and attention-based DRL to deal with the explosion of the action space. Turan et al. (2020) apply DRL to manage a fleet of electric vehicles through pricing, dispatching, and charging decisions. Haliem et al. (2021) resort to the deep Q-network algorithm to address a joint routing and repositioning problem. Yan et al. (2023) propose a model-based SARSA algorithm to optimize dispatching and charging decisions for electric vehicles. In contrast to most of these works, we focus on ride-hailing platforms that manage independent drivers who choose to self-relocate rather than autonomous vehicles. Our model also includes stochastic transit times and arrival rates, which significantly complicate the derivation and interpretation of the resulting policy. More fundamentally, while the majority of these works focus on (dispatching) decisions at the operational level, our model considers decisions to balance supply and demand at the tactical level. In that respect, our work complements the mentioned studies by providing a tactical policy to guide operational decisions.

2.3 Contributions to the Literature

Our paper makes two principal contributions to the literature. First, our work provides a modeling contribution, as being the first to jointly optimize pricing, driver repositioning, and customer admission on ride-hailing platforms. Furthermore, in contrast to previous research, our framework offers extensive modeling of the system, including features such as (i) spatio-temporal supply-demand imbalances, (ii) stochastic demand and transit times, and (iii) idle drivers with self-relocating behavior. This original setting leads us to gather relevant insights on the relevance of pricing, driver repositioning, and customer admission. Second, our work provides a methodological contribution. While the application of transfer learning methods to operations management problems is still in its early stage, our approach stands out by its recourse to mathematical optimization as a way to fasten the training process of a reinforcement learning algorithm. In contrast to alternative DRL algorithms, our approach can derive a better policy in a reduced amount of time through a stable learning process.

3 Markovian Model

In this section, we introduce the tactical problem of a platform that manages its network through pricing, driver repositioning and customer admission, and formulate it as a Markov decision process (MDP). A summary of the notation used throughout this section can be found in Table 1.

3.1 Problem Setting

We consider a ride-hailing platform that operates in a region (e.g., a city) over a finite time horizon (e.g., a working day). The region is divided into a set $\mathcal{N} \equiv \{1, ..., N\}$ of disjoint locations with reasonable size (e.g., a neighborhood). The platform makes decisions over a set $\mathcal{T} \equiv \{0, 1, ..., T-1\}$ of discrete epochs, separated by small periods of time Δt (e.g., every five minutes). In the following, we describe in detail the sequence of decisions and events between two decision epochs, as depicted in Fig. 1.

At each decision epoch t, the platform first observes the number of drivers $d^{\text{fin}}_{ij,t}$ who finish their journey from location i to j. It can then deduce the number of available drivers in location i, denoted by $x_{i,t}$, and the number of drivers in transit from location i to j, denoted by $y_{ij,t}$. Similar to recent literature (e.g., Q. Chen et al., 2023; Yan et al., 2023), we simplify exposition by supposing that the total number of drivers in the network is fixed to a constant D (we relax this assumption in Section 5.4). In addition, we assume that drivers in transit finish their journey from i to j after exponentially distributed transit times with mean $1/\tau_{ij}$. We suppose randomly distributed transit times to reflect the fact that transit times can be highly uncertain and are not always validly approximated with deterministic values. The assumption that their distributions are exponential allows us to keep the model Markovian, improving its tractability.

Next, the platform maximizes its profit rate through pricing, driver repositioning, and customer admission decisions. *The pricing decision* consists of setting the price rate $p_{i,t}$ that is charged per unit of distance for a service starting from location i. Consistent with practice, the price rate depends only on the origin location of the customer and is independent of the destination location (Uber, 2022). *The repositioning decision* refers to the number of drivers $r_{ij,t}$ who are moved empty by the platform from location i to j. Drivers can never be repositioned to their current location

Notation	Definition
Indexes and sets	
$i,\ j \in \mathcal{N} \equiv \{1, ..., N\}$	Locations i and j over the network $\mathcal{N}$.
$t \in \mathcal{T} \equiv \{0, ..., T-1\}$	Decision epoch t over the finite horizon $\mathcal{T}$.
Platform's decision variables	
$\mathbf{p}_t \equiv (p_{1,t}, ..., p_{i,t}, ..., p_{N,t})$	Price rates in location i at epoch t.
$\mathbf{r}_t \equiv (r_{11,t}, ..., r_{ij,t}, ..., r_{NN,t})$	Number of repositioned drivers from location i to j at epoch t.
$\mathbf{q}_t \equiv (q_{11,t}, ..., q_{ij,t}, ..., q_{NN,t})$	Percentages of admitted customers from location i to j at epoch t.
Demand parameters	
$F(\cdot)$ with $[0, \overline{f}]$	Cumulative distribution and support of the customer willingness to pay.
C^{lost}	Cost per unsatisfied customer request.
$\mathbf{\Lambda}_t \equiv (\Lambda_{11,t}, ..., \Lambda_{ij,t}, ..., \Lambda_{NN,t})$	Arrival rates of potential customers from location i to j at epoch t.
$\boldsymbol{\lambda}_t \equiv (\lambda_{11,t}, ..., \lambda_{ij,t}, ..., \lambda_{NN,t})$	Arrival rates of customer requests from location i to j at epoch t.
Demand variables	
$\mathbf{c}_t^{\text{req}} \equiv (c_{11,t}^{\text{req}}, ..., c_{ij,t}^{\text{req}}, ..., c_{NN,t}^{\text{req}})$	Number of customer requests from location i to j at epoch t.
$\mathbf{c}_t^{\text{adm}} \equiv (c_{11,t}^{\text{adm}}, ..., c_{ij,t}^{\text{adm}}, ..., c_{NN,t}^{\text{adm}})$	Number of admitted customer requests from location i to j at epoch t.
$\mathbf{c}_t^{\text{rej}} \equiv (c_{11,t}^{\text{rej}}, ..., c_{ij,t}^{\text{rej}}, ..., c_{NN,t}^{\text{rej}})$	Number of rejected customer requests from location i to j at epoch t.
Supply parameters	
D	Total number of drivers in the network.
$1/\tau_{ij}$	Expected transit time from location i to j.
δ_{ij}	Distance from location i to j.
β^{cons}, β^{price}, β^{trans}	Preference coefficients of self-relocating drivers.
γ	Temperature parameter of the dispersion of the self-relocating choice.
C^{rep}	Cost rate per repositioned driver.
Supply variables	
$\mathbf{x}_t \equiv (x_{1,t}, ..., x_{i,t}, ..., x_{N,t})$	Number of drivers available in location i at epoch t.
$\mathbf{y}_t \equiv (y_{11,t}, ..., y_{ij,t}, ..., y_{NN,t})$	Number of drivers in transit from location i to j at epoch t.
$\mathbf{d}_t^{\text{self}} \equiv (d_{11,t}^{\text{self}}, ..., d_{ij,t}^{\text{self}}, ..., d_{NN,t}^{\text{dis}})$	Number of drivers who self-relocate from location i to j at epoch t.
$\mathbf{d}_t^{\text{dis}} \equiv (d_{11,t}^{\text{dis}}, ..., d_{ij,t}^{\text{dis}}, ..., d_{NN,t}^{\text{dis}})$	Number of drivers who are dispatched from location i to j at epoch t.
$\mathbf{d}_t^{\text{fin}} \equiv (d_{11,t}^{\text{fin}}, ..., d_{ij,t}^{\text{fin}}, ..., d_{NN,t}^{\text{fin}})$	Number of drivers who finish their journey from location i to j at epoch t.

Table 1: Summary of the notations.

(i.e., $r_{ii,t} = 0$ for all $i \in \mathcal{N}$). For each repositioned driver, the platform incurs a repositioning cost rate C^{rep} per unit of distance, which resembles industry practices (e.g., Lyft's bonus zones, 2021). *The admission decision* determines the probability $q_{ij,t}$ for a customer going from location i to j to be admitted to the system. By doing this, the platform can select which requests are satisfied when supply is too scarce to serve the whole demand. As such, it allows the platform to ration driver supply, so as to prioritize customers heading toward more convenient locations. Since future demand is unknown at the decision epoch (and do not accumulate as in dispatching problems), we express this decision as a probability rather than a number of customers. For every rejected customer, the platform incurs a fixed lost sales cost C^{lost}, reflecting a loss of brand image.

After the platform's decisions, any driver who has not been repositioned by the platform can choose to self-relocate to another location. The number of drivers who elect to self-relocate from a location i to another location j is denoted by $d_{ij,t}^{\text{self}}$. Drivers never elect to self-relocate to their current location (i.e., $d_{ii,t}^{\text{self}} = 0$ for all $i \in \mathcal{N}$). Throughout this paper, these drivers are called the "self-relocating drivers". We distinguish them from the "repositioned drivers" who follow the platform's repositioning decision, and the "dispatched drivers" who satisfy customer requests. Following other studies (e.g., Lu et al., 2018), we suppose that drivers make their self-relocating choices based on the expected earnings that they would obtain in the other locations, and the transit times that they would spend to reach these

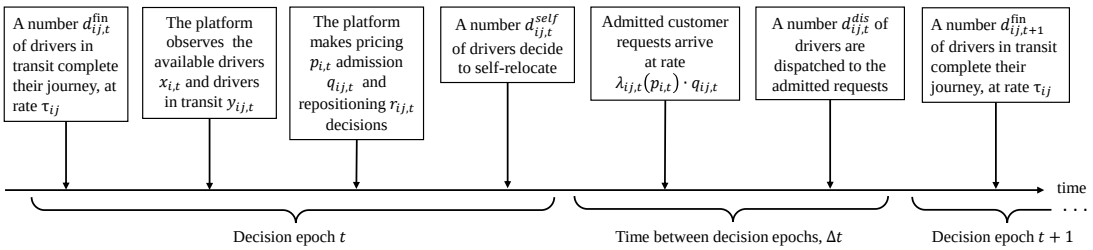


Figure 1: Timeline of decisions and events.

locations, relative to their current location. Although any discrete choice model can be used, we represent the driver self-relocating choice with a multinomial logit model. Let β^{cons} be the constant term, β^{price} be the expected driver preference toward locations with prices higher than in their location, and β^{trans} be the expected aversion toward locations with long transit times from their location. The expected utility of a self-relocating driver from location i to j is defined as $u_{ij,t} = \beta^{\text{cons}} + \beta^{\text{price}} \cdot (p_{j,t} - p_{i,t}) - \beta^{\text{trans}} \cdot 1/\tau_{ij}$. If the utility of the outside option (stay in the current location i) is normalized to 0, then the probability for a driver to self-relocate from location i to j is given by $\psi_{ij,t} = \exp(u_{ij,t}/\gamma)/\left[1 + \sum_{\substack{k=1 \\ k\neq i}}^{N} \exp(u_{ik,t}/\gamma)\right]$, with $\psi_{ii,t}$ being the probability that drivers stay idle in location i. The temperature parameter γ measures the dispersion of the driver choice. As $\gamma \to 0$, drivers choose with certainty the option with the highest expected utility. Conversely, as $\gamma \to \infty$, drivers choose each option with equal probability.

Between the decision epochs t and $t+1$ (see Fig. 1), potential customers who seek to go from location i to location j continuously arrive following a non-stationary Poisson process with rate $\Lambda_{ij,t}$. Upon arrival, a potential customer decides whether to request service based on her willingness to pay. The customer willingness to pay is heterogeneous, and characterized by the concave[1] cumulative distribution $F(\cdot)$ over the support $[0, \overline{f}]$. Therefore, given price rate $p_{i,t}$, the expected number of customer requests from location i to j can be written as $\lambda_{ij,t} = \Lambda_{ij,t} \cdot \left[1 - F(p_{i,t})\right]$. The realized number of customer requests is denoted by $c_{ij,t}^{\text{req}}$, and sampled from a Poisson distribution with rate $\lambda_{ij,t}$. Directly upon arrival, the platform applies the admission decision. According to this decision, a customer request from location i to j has a probability $q_{ij,t}$ to be admitted to the system. The number of admitted customer requests, noted $c_{ij,t}^{\text{adm}}$, and the number of rejected customer requests, noted $c_{ij,t}^{\text{rej}}$, thus follow Poisson distributions with rate $\lambda_{ij,t} \cdot q_{ij,t}$ and $\lambda_{ij,t} \cdot (1 - q_{ij,t})$ respectively, such that $c_{ij,t}^{\text{req}} = c_{ij,t}^{\text{adm}} + c_{ij,t}^{\text{rej}}$.

Following each admission, an available driver is directly dispatched to the admitted request if possible. We assume that drivers can only be dispatched to admitted requests in the same location, on a first-come, first-served (FCFS) basis. The first assumption reflects the practical idea that ride-hailing platforms match drivers and customers close to each other to avoid long pick-up times. The second assumption is common in theory (e.g., Afèche et al., 2023) and in practice (e.g., Lyft, 2024), as it reduces the problem complexity and is perceived fairer than other queueing regimes. Hence, the number of dispatched drivers from location i to j, noted $d_{ij,t}^{\text{dis}}$, depends on the number of admitted requests and the number of remaining drivers in location i. If the number of admitted requests is inferior or equal to the number of remaining drivers, then all the requests are served, so that $d_{ij,t}^{\text{dis}} = c_{ij,t}^{\text{adm}}$ for all $j \in \mathcal{N}$. If the number of admitted requests is superior to the number of remaining drivers, then only a fraction of the requests are served so that $d_{ij,t}^{\text{dis}} < c_{ij,t}^{\text{adm}}$ for some $j \in \mathcal{N}$ (depending on the order of arrival). In the latter case, unsatisfied customers are assumed

[1]The concavity assumption is in line with the revenue management literature (cf. Talluri et al., 2004), and fits many common distributions including the uniform, exponential, and logit distributions.

to be impatient and opt for other transport means (e.g., the subway).[2] Similar to rejected requests, each unsatisfied request generates a lost sales cost C^{lost}.

3.2 MDP Formulation

In this section, we formalize the problem introduced above as a discrete-time Markov decision process (MDP). To this end, we detail the states, decisions, transitions, rewards, and policies.

3.2.1 States and Decisions

The system state $\mathbf{s}_t$ at decision epoch t can be characterized by the number of drivers available $x_{i,t}$ in each location i, and the number of drivers in transit $y_{ij,t}$ from each location i to each location j. It is defined as the tuple $\mathbf{s}_t = (\mathbf{x}_t, \mathbf{y}_t)$, with vector $\mathbf{x}_t = (x_{1,t}, \ldots, x_{N,t})$ and matrix $\mathbf{y}_t = (y_{11,t}, \ldots, y_{NN,t})$. Therefore, the state space comprises $N + N^2$ dimensions. It only contains states such that the total number of drivers in the network equals D, the fixed number of drivers. The state space is described by the set

$$\mathcal{S} = \left\{(\mathbf{x}_t, \mathbf{y}_t) : t \in \mathcal{T}, \mathbf{x}_t \in \mathbb{Z}^N, \mathbf{y}_t \in \mathbb{Z}^{N\times N}, \sum_{i=1}^{N} x_{i,t} + \sum_{i=1}^{N}\sum_{j=1}^{N} y_{ij,t} = D\right\}. \tag{1}$$

Note that the system state does not include the number of customers waiting for service due to our assumption that customers are impatient.

At decision epoch t, after observing the system state $\mathbf{s}_t$, the platform makes the pricing $p_{i,t}$, repositioning $r_{ij,t}$, and admission $q_{ij,t}$ decisions, which can be represented by a tuple, denoted by $\mathbf{a}_t = (\mathbf{p}_t, \mathbf{r}_t, \mathbf{q}_t)$. The first component $\mathbf{p}_t = (p_{1,t}, \ldots, p_{N,t})$ is the vector comprising the price rates. Each price rate is continuous and bounded above by the maximum customer willingness to pay $\overline{f}$ as higher prices would result in no customer requests. The second component $\boldsymbol{r}_t = (r_{11,t}, \ldots, r_{N(N-1),t})$ is the matrix composed of the number of repositioned drivers. The number of repositioned drivers from a location i is always inferior or equal to the number of available drivers in the location, $x_{i,t}$. The third component $\boldsymbol{q}_t = (q_{11,t}, \ldots, q_{NN,t})$ is the matrix that contains the percentages of admission among customer requests. Consequently, the decision space has $2 \times N^2$ dimensions, and is given by

$$\begin{aligned}\mathcal{A}(\mathbf{s}_t) = \Big\{(\mathbf{p}_t, \mathbf{r}_t, \mathbf{q}_t) : \mathbf{p}_t \in [0, \overline{f}]^N, \mathbf{r}_t \in \mathbb{Z}^{N\times(N-1)}, \mathbf{q}_t \in [0, 1]^{N\times N}, \\ \sum_{j=1}^{N} r_{ij,t} \leq x_{i,t} \;\; \forall\, i \in \mathcal{N}\Big\}.\end{aligned} \tag{2}$$

3.2.2 State Transitions

Following the platform's decisions, the state transitions from $\mathbf{s}_t$ to $\mathbf{s}_{t+1}$. As shown in Fig. 1, the state $\mathbf{s}_t = (\mathbf{x}_t, \mathbf{y}_t)$ first evolves deterministically, due to the repositioning decision $\mathbf{r}_t$. Then, it changes stochastically, as a function of the self-relocating drivers $\mathbf{d}_t^{\text{self}} = (d_{11,t}^{\text{self}}, \ldots, d_{NN,t}^{\text{self}})$, the dispatched drivers $\mathbf{d}_t^{\text{dis}} = (d_{11,t}^{\text{dis}}, \ldots, d_{NN,t}^{\text{dis}})$, and the drivers in transit who finish their journey $\mathbf{d}_{t+1}^{\text{fin}} = (d_{11,t+1}^{\text{fin}}, \ldots, d_{NN,t+1}^{\text{fin}})$, respecting that order. The self-relocating drivers $\mathbf{d}_{ij,t}^{\text{self}}$

[2] Since we focus on tactical decisions (e.g. every 5 minutes), it is reasonable to suppose that customers leave if not served between epochs. Customer waiting can be included following slight model alterations.

are influenced by the pricing decision $\mathbf{p}_t$ while the dispatched drivers $d^{\text{dis}}_{ij\,t}$ indirectly depend on the pricing decision $\mathbf{p}_t$ and the admission decision $\mathbf{q}_t$. In mathematical terms, the number of drivers available in location i at the decision epoch $t+1$ is expressed as

$$x_{i,\,t+1} = x_{i,\,t} - \sum_{j=1}^{N} r_{ij,\,t} - \sum_{j=1}^{N} d^{\text{self}}_{ij,\,t} - \sum_{j=1}^{N} d^{\text{dis}}_{ij,\,t} + \sum_{j=1}^{N} d^{\text{fin}}_{ji,\,t+1} \qquad \forall\, i \in \mathcal{N},\, t \in \mathcal{T}. \tag{3}$$

On the right-hand side, the first term represents available drivers; the second (drivers repositioned by the platform), third (self-relocating drivers), and fourth (dispatched drivers) terms stand for drivers leaving location, while the last term refers to drivers entering the location.

Following the same logic as in Eq. (3), the number of drivers in transit from location i to j at the decision epoch $t+1$ is given by

$$y_{ij,\,t+1} = y_{ij,\,t} + r_{ij,\,t} + d^{\text{self}}_{ij,\,t} + d^{\text{dis}}_{ij,\,t} - d^{\text{fin}}_{ij,\,t+1} \qquad \forall\, i \in \mathcal{N},\, j \in \mathcal{N},\, t \in \mathcal{T}. \tag{4}$$

Note that we always have $r_{ii,\,t} = 0$ and $d^{\text{self}}_{ii,\,t} = 0$ since drivers are never repositioned, and never self-relocate to their current location.

In Eq. (3) and Eq. (4), the three stochastic components $\mathbf{d}^{\text{self}}_t$, $\mathbf{d}^{\text{dis}}_t$ and $\mathbf{d}^{\text{fin}}_t$ are derived as follows. First, the number of self-relocating drivers $\mathbf{d}^{\text{self}}_t$ is modeled according to a multinomial distribution specific to each location i. This is because drivers elect to self-relocate according to a multinomial logit model. For each distribution, the number of trials is given by the number of drivers remaining in location i after platform repositioning. The probability for drivers to self-relocate to another location j is encoded by the probability vector $(\psi_{i1,\,t},\, ...,\, \psi_{ij,\,t},\, ...,\, \psi_{iN,\,t})$, deduced from the multinomial logit model. Second, the number of dispatched drivers $\mathbf{d}^{\text{dis}}_t$ follows a hypergeometric distribution[3] that is also specific to each location i. This is because admitted requests arrive according to independent Poisson processes, and are served in an FCFS order. The distribution is characterized by the number of drivers in location i after driver repositioning and driver self-relocating, and the number of admitted requests going from location i to any location j, summarized by the vector $(c^{\text{adm}}_{i1,\,t},\, ...,\, c^{\text{adm}}_{ij,\,t},\, ...,\, c^{\text{adm}}_{iN,\,t})$. Lastly, the number of drivers finishing their journey $\mathbf{d}^{\text{fin}}_{t+1}$ follows a binomial distribution as transit times are exponentially distributed and thus memoryless. For each distribution, the number of trials corresponds to the drivers in transit between the decision epoch t and $t+1$. The probability for drivers to finish their journey from location i to j between two decision epochs can be expressed as $(1 - e^{-\tau_{ij} \cdot \Delta t})$.

3.2.3 Rewards

Service revenues, lost sales costs, and repositioning costs occur randomly at each decision epoch, along with the state transition. These are captured by the following reward function:

$$\begin{aligned} R_t(\mathbf{s}_t,\, \mathbf{a}_t) = &\sum_{i=1}^{N} \sum_{j=1}^{N} p_{i,\,t} \cdot d^{\text{dis}}_{ij,\,t} \cdot \delta_{ij} \\ &- C^{\text{lost}} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} (c^{\text{req}}_{ij,\,t} - d^{\text{dis}}_{ij,\,t}) - C^{\text{rep}} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} r_{ij,\,t} \cdot \delta_{ij}, \end{aligned} \tag{5}$$

[3]The multivariate hypergeometric distribution characterizes sampling without replacement from a finite population (the available drivers in location i) when each outcome falls into one of several categories (the service of a customer going to location j).

where the first term stands for the platform revenues, and the second, and third terms refer to the lost sales costs, and repositioning costs. According to Eq. (5), the platform collects some revenue for every driver dispatched from location i to j. This revenue is proportional to the price rate $p_{i,t}$ in the origin location i, and the distance δ_{ij} between the origin i and destination j. For every unsatisfied request, the platform incurs a fixed lost sales cost C^{lost}. Lastly, the cost for repositioning a driver depends on the distance δ_{ij} traveled by the driver from location i to j. In our model, the repositioning cost is entirely accounted for when a driver starts repositioning, with a cost rate C^{rep} per unit of distance.

3.2.4 Policies and Value Function

Over a finite horizon $\mathcal{T}$, a policy is a sequence $\pi = \{\pi_0, \pi_1, \pi_2, \dots, \pi_{T-1}\}$ where π_t is a function mapping the system states to decisions made in those states (at time t). Let Π be the set of feasible policies and $\mathbf{s}_0$ be the state at the beginning of the decision horizon. Our objective is to find an optimal policy π^* that maximizes the expected total reward:

$$\pi^* \equiv \arg\max_{\pi \in \Pi} \mathbb{E}_\pi \Big[\sum_{t=0}^{T-1} R_t\big(\mathbf{s}_t,\, \pi_t(\mathbf{s}_t)\big) \,\Big|\, \mathbf{s}_0 \Big]. \tag{6}$$

To find the optimal policy π^*, the optimal value function can be computed. The optimal value function $v^*(\mathbf{s}_t)$ $\big($with $v^*(\mathbf{s}_t) \equiv v_{\pi^*}(\mathbf{s}_t)\big)$ measures the total future reward that can be expected from a given state $\mathbf{s}_t$ following an optimal policy π^* (Powell, 2007). In our case, it stands for the total future profit expected until the end of the horizon. We can define it recursively as

$$v^*(\mathbf{s}_t) = \max_{\pi(\mathbf{s}_t) \in \mathcal{A}(\mathbf{s}_t)} \mathbb{E}_\pi \Big[R_t\big(\mathbf{s}_t,\, \pi_t(\mathbf{s}_t)\big) + v^*(\mathbf{s}_{t+1}) \,\Big|\, \mathbf{s}_t \Big]. \tag{7}$$

In theory, Eq. (7) can be solved using dynamic programming (Bellman, 1957). Nevertheless, for our specific model, the curse of dimensionality makes this method intractable, as the number of states and transitions grows exponentially with the number of drivers operating and the number of locations.[4] The decision space is also partially continuous (due to the pricing and admission decisions), while dynamic programming is more suited for discrete decisions. Deep reinforcement learning (DRL) can handle large state spaces, and continuous decisions. Yet, given the complexity of our model, even advanced DRL algorithms fail to learn efficient policies. For this reason, we devise a method to improve upon existing DRL in the next section.

4 Transfer Learning Approach

In this section, we develop a transfer learning approach that combines deep reinforcement learning (DRL) with mathematical optimization. At its core, the approach relies on a pretraining process that teaches actor-critic neural networks to reproduce decisions, followed by a rolling-horizon strategy. The approach proceeds in three steps. First, the rolling-horizon strategy is simulated in the MDP by repeatedly solving an optimization problem based on expected values of stochastic parameters. Second, during actor pretraining, the actor learns a policy using the state-decision

[4] For instance, if D drivers are operating in a network of N locations, then there are $\frac{\left(D+N+N^2-1\right)!}{D!\left(N+N^2-1\right)!}$ possible states.

pairs obtained by simulating the rolling-horizon strategy. Third, during critic pretraining, the critic neural network approximates the value function associated with the actor policy. After these steps, the DRL algorithm starts learning from an already efficient policy rather than from scratch, saving computational efforts while reaching improved performances. In the following, Section 4.1 introduces our actor-critic framework. Then, Section 4.2 details our pretraining process and its three components: the rolling-horizon strategy, actor pretraining, and critic pretraining.

4.1 Actor-Critic Framework

Our approach is grounded in actor-critic methods, which are made of two components (cf. Sutton & Barto, 2018 for a more detailed exposition). On the one hand, the *actor* looks for a near-optimal policy. To this end, it derives a policy function $\pi(\boldsymbol{\theta})$ that selects decisions according to the set of parameters $\boldsymbol{\theta}$ (which depends on the approximation method). On the other hand, the *critic* approximates the corresponding value function using the set of parameters $\boldsymbol{\eta}$ (which also depends on the approximation method). This approximation, noted $\widehat{v}_{\pi(\boldsymbol{\theta})}(\mathbf{s}_t, \boldsymbol{\eta})$, guides the search for a near-optimal policy by the actor.

Although any parametric function can approximate the value and policy functions, we choose the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ to be the weights of two distinct neural networks. Neural networks are flexible tools that can, in theory, approximate any function (LeCun et al., 2015). In our approach, the critic neural network returns the approximate value function $\widehat{v}_{\pi(\boldsymbol{\theta})}(\mathbf{s}_t, \boldsymbol{\eta})$ but the actor neural network does not directly predict decisions. Instead, the actor neural network learns a set of $N \cdot (N+1)$ Gaussian distributions. A vector $\mathbf{p}_t^{\text{net}} \in [0, 1]^N$, associated with the pricing decision, and a matrix $\mathbf{rq}_t^{\text{net}} \in [-1, 1]^{N \times N}$, associated with the repositioning and admission decisions, are then sampled from these distributions, ensuring the exploration of new decisions.

At each epoch t, the three decisions can be derived from $\mathbf{p}_t^{\text{net}} \in [0, 1]^N$ and $\mathbf{rq}_t^{\text{net}} \in [-1, 1]^{N \times N}$. The pricing decision $\mathbf{p}_t$ is obtained by scaling each component of $\mathbf{p}_t^{\text{net}}$ with the customer maximum willingness to pay $\overline{f}$ such that $p_{i,t} = p_{i,t}^{\text{net}} \cdot \overline{f}$. The repositioning decision $\mathbf{r}_t$ and the admission decision $\mathbf{q}_t$ are both determined from the matrix $\mathbf{rq}_t^{\text{net}}$, hence reducing the number of parameters in the actor neural network. This follows the intuition that the platform should never reposition drivers and reject customers simultaneously, for a pair of locations.

From location i to j, the percentage of admitted customers $q_{ij,t}$, and the number of repositioned drivers $r_{ij,t}$ are deduced from $\mathbf{rq}^{\text{net}}$ using the following transformations:

$$r_{ij,t} = \begin{cases} \left\lfloor x_{i,t} \cdot \frac{rq_{ij,t}^{\text{net}}}{\sum_{k=1}^{N} (rq_{ik,t}^{\text{net}})^+} \right\rfloor & \text{if } rq_{ij,t}^{\text{net}} > 0,\ i \neq j, \\ 0 & \text{otherwise,} \end{cases} \quad \text{and} \quad q_{ij,t} = \begin{cases} |rq_{ij,t}^{\text{net}}| & \text{if } rq_{ij,t}^{\text{net}} \leq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where $\lfloor \cdot \rfloor$ is the round down operator, and $(\cdot)^+ = \max(0, \cdot)$. If $rq_{ij,t}^{\text{net}} > 0$, then all the customer requests are admitted ($q_{ij,t} = 0$), and a fraction of the available drivers $x_{i,t}$ reposition from location i to j. In this case, the component $rq_{ii,t}^{\text{net}} > 0$ determines the share of drivers who are left available in location i.[5] Conversely, if $rq_{ij,t}^{\text{net}} \leq 0$, then only a percentage given by $|rq_{ij,t}^{\text{net}}|$ of the customer requests are admitted, and no driver is repositioned from location i to j ($r_{ij,t} = 0$).

[5]Note that our formulation does not allow customers to be rejected (i.e., $rq_{ii,t}^{\text{net}} < 0$) while drivers are left available (i.e., $rq_{ii,t}^{\text{net}} > 0$). This limitation does not cause notable profit loss, as shown in Section 5.2.

Various DRL algorithms could train the actor-critic neural networks. In this paper, we use the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017) for two reasons. First, PPO benefits from high stability and sample efficiency, reducing the amount of data needed to improve the policy. Second, PPO is known to require little hyperparameter tuning. PPO is thus convenient in our situation because the pretraining process in Section 4.2 involves additional hyperparameters. For a detailed presentation of the PPO algorithm, we refer to Schulman et al. (2017).

4.2 Pretraining Process

Next, we present the process used to pretrain the actor-critic neural networks. The process has three components: the rolling-horizon strategy, actor pretraining, and critic pretraining.

4.2.1 Rolling-Horizon Strategy

This section defines a rolling-horizon strategy, which periodically solves a deterministic optimization problem. Such a strategy serves two purposes. First, it generates state decision pairs that are used to pre-train the neural network of the actor (Section 4.2.2). Second, it provides a proper benchmark to compare our transfer learning approach (Section 5.2).

Before defining the rolling-horizon strategy, we first formalize its underlying optimization problem. When all uncertainty is neglected, a natural way to represent our problem is to formulate a multi-period optimization problem based on the expected values (Birge & Louveaux, 2011). Specifically, we propose an optimization problem, named the Expected Value Problem (EVP), that (i) replaces the stochastic number of requests, dispatched drivers, and service completions by their expected values, (ii) ignores self-relocating drivers,[6] and (iii) allows continuous rather than integer variables. The parameters of the problem can be assessed from available historical data. It takes an initial state as input and deduces prices, rejected customers, and repositioned drivers for future decision epochs in a deterministic setting.

Let us introduce $\mathbf{s}_0 = (\mathbf{x}_0, \mathbf{y}_0)$ the initial system state, and $\rho_{ij,t} = e^{-\tau_{ij}\cdot\Delta t} \cdot (1 - e^{-\tau_{ij}\cdot\Delta t})^t$ the probability that a driver takes t decision epochs ($t \geq 0$) to finish her journey from location i to j. Then, the EVP can be formulated as

$$\max \sum_{t=0}^{T-1}\sum_{i=1}^{N}\sum_{j=1}^{N} p_{i,t}\cdot d^{\text{dis}}_{ij,t}\cdot\delta_{ij} - C^{\text{lost}}\cdot\sum_{t=0}^{T-1}\sum_{i=1}^{N}\sum_{j=1}^{N} c^{\text{rej}}_{ij,t} - C^{\text{rep}}\cdot\sum_{t=0}^{T-1}\sum_{i=1}^{N}\sum_{j=1}^{N} r_{ij,t}\cdot\delta_{ij} \tag{9a}$$

$$\text{s.t.}\quad c^{\text{req}}_{ij,t} = \Lambda_{ij,t}\cdot\big(1 - F(p_{i,t})\big) \qquad \forall\, i,\, j,\, t, \tag{9b}$$

$$d^{\text{dis}}_{ij,t} = c^{\text{req}}_{ij,t} - c^{\text{rej}}_{ij,t} \qquad \forall\, i,\, j,\, t, \tag{9c}$$

$$x_{i,0} \geq \sum_{j=1}^{N}\big(d^{\text{dis}}_{ij,0} + r_{ij,0}\big) \qquad \forall\, i, \tag{9d}$$

$$x_{i,0} + \sum_{k=0}^{t-1}\sum_{s=0}^{t-k-1}\sum_{j=1}^{N}\rho_{ji,s}\cdot\big(d^{\text{dis}}_{ji,k} + r_{ji,k}\big) + \sum_{s=0}^{t-1}\sum_{j=1}^{N}\rho_{ji,s}\cdot y_{ji,0} \geq \sum_{k=0}^{t}\sum_{j=1}^{N}\big(d^{\text{dis}}_{ij,k} + r_{ij,k}\big) \qquad \forall\, i,\, t \geq 1, \tag{9e}$$

$$p_{i,t},\, c^{\text{req}}_{ij,t},\, c^{\text{rej}}_{ij,t},\, d^{\text{dis}}_{ij,t},\, r_{ij,t} \geq 0 \qquad \forall\, i,\, j,\, t. \tag{9f}$$

Similar to Eq.,(5), objective function (9a) includes revenues for serving customers and costs for rejecting customers

[6]Including self-relocating drivers would make the optimization model intractable, as the probability for drivers to self-relocate would be expressed through nonconvex constraints.

and relocating drivers. Constraints (9b) determine the number of requests between two locations based on the arrival rates of potential customers, prices, and the distribution $F(\cdot)$ of the customer willingness to pay. Constraints (9c) ensure that the number of dispatched drivers between two locations equals the number of admitted customers. Because of the absence of uncertainty, requests can never be admitted but not served. Constraints (9d) and constraints (9e), interpreted as "flow constraints," enforce that the number of dispatched and repositioned drivers is inferior to the number of available drivers in each location for each epoch. In constraints (9e), the first term is associated with the initial number of available drivers; the second term refers to the number of drivers initially in transit, who finished their journey; the third term represents the number of dispatched and repositioned drivers who finished their journey; the fourth term, on the right-hand side, corresponds to the number of dispatched and repositioned drivers. Finally, constraints (9f) are nonnegativity constraints. The EVP is nonlinear due to the first term in objective function (9a) and the concavity of constraints (9b). However, it can be made convex using McCormick envelopes (1976), and solved with a commercial solver.

Having introduced the deterministic optimization model, we can now present the rolling-horizon strategy. The strategy progresses through decision epochs by periodically re-optimizing the EVP every T^{RH} decision epochs and applying the decisions until the next re-optimization, over and over. At each re-optimization, the current state $\mathbf{s}_t^{\text{RH}}$ is taken as the initial system state in the EVP to account for dynamic variations that occurred between two optimizations. The EVP is re-optimized over the next Q^{RH} decision epochs (with $Q^{\text{RH}} \geq T^{\text{RH}}$) so as to capture future demand patterns. Because the EVP considers continuous variables and neglects uncertainty, its optimal solutions cannot be directly applied in the MDP. Therefore, the optimal solutions must be converted into feasible decisions $\mathbf{p}^{\text{RH}}$, $\mathbf{q}^{\text{RH}}$ and $\mathbf{r}^{\text{RH}}$. The pricing decision is determined as $p_{i,t}^{\text{RH}} = p_{i,t}$ in location i at epoch t. The admission decision is calculated as $q_{ij,t}^{\text{RH}} = 1 - c_{ij,t}^{\text{rej}} / c_{ij,t}^{\text{req}}$, and the repositioning decision as $r_{ij,t}^{\text{RH}} = \lfloor r_{ij,t} \rfloor$ from location i to j at epoch t.

As part of the rolling-horizon strategy, the optimization interval T^{RH} and optimization horizon Q^{RH} can balance solution quality with computational effort and memory use. On the one hand, the optimization interval T^{RH} finds a trade-off between static decisions involving few optimizations (with low T^{RH}), and dynamic decisions involving many optimizations (with high T^{RH}). On the other hand, the optimization horizon Q^{RH} balances myopic decisions requiring little computations and memory per optimization (with low Q^{RH}), with far-sighted decisions requiring more computations and memory per optimization (with high Q^{RH}). By varying these two parameters, the rolling-horizon strategy can be adapted to our purpose. High T^{RH} and low Q^{RH} are favored to quickly pretrain the actor neural network (Section 4.2.2). Low T^{RH} and high Q^{RH} are used to benchmark our approach against a strong baseline (Section 5.2).

4.2.2 Actor Pretraining

To pretrain our actor neural network, we simulate the rolling-horizon strategy in the MDP and collect information about the states visited and the decisions prescribed. That is, at each decision epoch t, the system state $\mathbf{s}_t^{\text{RH}}$ and the decision $\mathbf{a}_t^{\text{RH}}$ are stored in a state-decision set containing all the state-decision pairs observed so far, and defined as $\mathcal{D}^{\text{RH}} = \left\{(\mathbf{s}_k^{\text{RH}}, \mathbf{a}_k^{\text{RH}})\right\}_{k=0}^{t}$. By repeating the operation for many decision epochs (possibly through parallel computing), the state-decision set $\mathcal{D}^{\text{RH}}$ gradually covers more states that are likely to be visited, and records the associated decisions. In the end, the set includes various state-decision pairs, producing a partial policy (i.e., a mapping from a subset of the states to decisions). This partial policy offers an accurate representation of the rolling-horizon strategy.

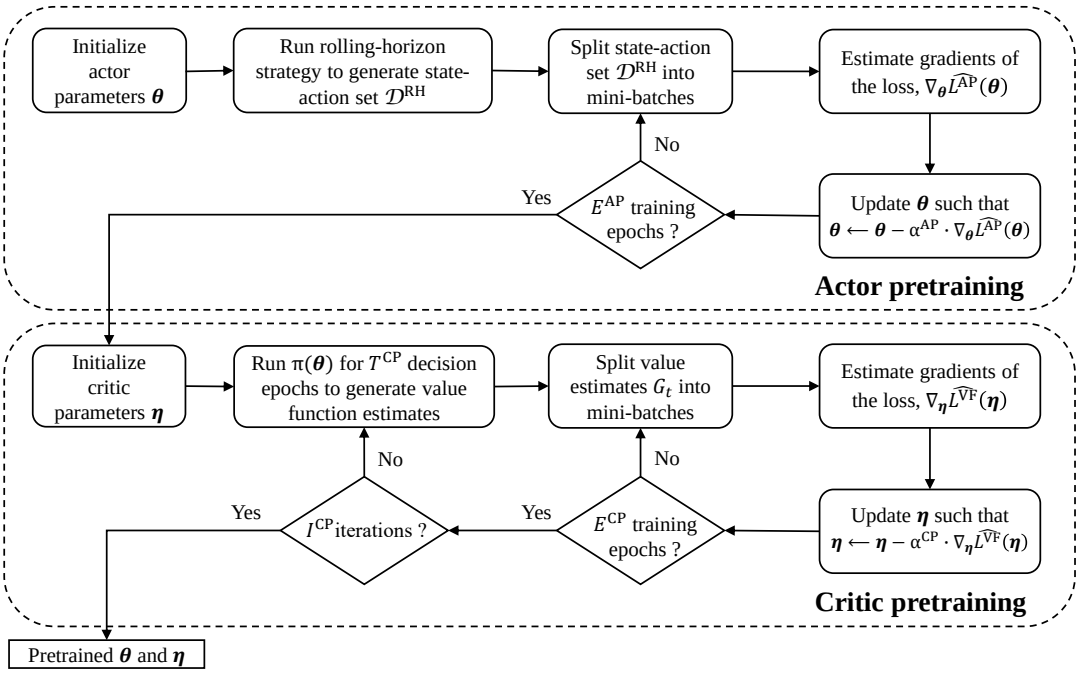


Figure 2: Flow diagram of the pretraining process.

The actor neural network then uses the state-decision set $\mathcal{D}^{\mathrm{RH}}$ to learn a policy that mimics the rolling-horizon strategy. Following standard machine learning terminology (LeCun et al., 2015), the state-decision set can be viewed as a set of training examples in which each state $\mathbf{s}_t^{\mathrm{RH}}$ is a feature vector, and each decision $\mathbf{a}_t^{\mathrm{RH}}$ is the corresponding label. Accordingly, we can use supervised learning methods, and train the policy function $\pi(\boldsymbol{\theta})$ to replicate the decisions contained in the set $\mathcal{D}^{\mathrm{RH}}$. Such a procedure is often called "imitation learning" in other fields such as robotics and telecommunications (Boute et al., 2022). In our case, it boils down to minimizing the following loss function:

$$L^{\mathrm{AP}}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{D}^{\mathrm{RH}}|} \sum_{(\mathbf{s}_t^{\mathrm{RH}},\, \mathbf{a}_t^{\mathrm{RH}}) \in \mathcal{D}^{\mathrm{RH}}} \left[\frac{1}{2} \left(\pi(\mathbf{s}_t^{\mathrm{RH}},\, \boldsymbol{\theta}) - \mathbf{a}_t^{\mathrm{RH}} \right)^2 \right], \tag{10}$$

where $|\mathcal{D}^{\mathrm{RH}}|$ is the cardinality of the state-decision set. Also named the "mean squared error", this loss function measures the squared gaps between the decisions predicted by the actor neural network, $\pi(\mathbf{s}_t^{\mathrm{RH}},\, \boldsymbol{\theta})$, and the decisions $\mathbf{a}_t^{\mathrm{RH}}$ taken for the same states according to $\mathcal{D}^{\mathrm{RH}}$.

The first box of Fig. 2 details the actor pretraining. After deriving the state-decision set $\mathcal{D}^{\mathrm{RH}}$, the function $L^{\mathrm{AP}}(\boldsymbol{\theta})$ is minimized by means of gradient descent. First, the state-decision set $\mathcal{D}^{\mathrm{RH}}$ is divided into mini-batches. Second, for each mini-batch, the approximate loss function $\widehat{L^{\mathrm{AP}}}(\boldsymbol{\theta})$ and its gradient $\nabla_{\theta} \widehat{L^{\mathrm{AP}}}(\boldsymbol{\theta})$ are computed with respect to $\boldsymbol{\theta}$. Third, the parameters $\boldsymbol{\theta}$ are updated by performing one gradient step per mini-batch such that $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha^{\mathrm{AP}} \nabla_{\boldsymbol{\theta}} \widehat{L^{\mathrm{AP}}}(\boldsymbol{\theta})$ where α^{AP} is the learning rate. One training epoch is completed after an update for each mini-batch. The procedure is repeated, making new mini-batches from $\mathcal{D}^{\mathrm{RH}}$ until a total of E^{AP} training epochs are completed. As more training epochs occur, $L^{\mathrm{AP}}(\boldsymbol{\theta})$ converges to a local minimum, and $\pi(\boldsymbol{\theta})$ approximates the rolling-horizon strategy described by $\mathcal{D}^{\mathrm{RH}}$.

4.2.3 Critic Pretraining

During critic pretraining, the critic neural network approximates the value function of the policy previously learned by the actor neural network, which is related to the general concept of policy evaluation (see, e.g., Chapter 4 in Sutton & Barto, 2018). To approximate the value function, we minimize the loss function given by

$$L^{\mathrm{CP}}(\boldsymbol{\eta}) = \sum_{t=0}^{T-1} \sum_{\mathbf{s}_t \in \mathcal{S}} \mathbb{P}(\mathbf{s}_t) \Big[\frac{1}{2} \big(\hat{v}_{\pi(\boldsymbol{\theta})}(\mathbf{s}_t, \boldsymbol{\eta}) - G_t \big)^2 \Big], \tag{11}$$

where G_t denotes the total realized profit from state $\mathbf{s}_t$ until the end of the horizon under policy $\pi(\boldsymbol{\theta})$, i.e., $G_t = \sum_{k=t}^{T-1} R_k\big(\mathbf{s}_k, \pi_k(\mathbf{s}_k, \boldsymbol{\theta})\big)$. Eq. (11) aims to quantify the difference between the approximate value function $\hat{v}_{\pi(\boldsymbol{\theta})}(\mathbf{s}_t, \boldsymbol{\eta})$ and G_t, an unbiased estimate of the true value function $v_{\pi(\boldsymbol{\theta})}(\mathbf{s}_t, \boldsymbol{\eta})$ for following policy $\pi(\boldsymbol{\theta})$ at epoch t.

An overview of the critic pretraining is displayed in the second box of Fig. 2. First, the pretrained policy function $\pi(\boldsymbol{\theta})$ is simulated in the MDP for T^{CP} decision epochs, and the states and rewards are collected. The collected rewards are then used to compute the value function estimates $(G_t, \ldots, G_{t+T^{\mathrm{CP}}-1})$ associated with each visited state. Second, the stochastic gradient descent algorithm is executed to minimize Eq. (11). That is, for E^{CP} training epochs, the visited states and associated value function estimates are shuffled into mini-batches. For each mini-batch, the gradient of the loss function $\nabla_{\boldsymbol{\eta}} \widehat{L^{\mathrm{VF}}}(\boldsymbol{\eta})$ is estimated, and the parameters $\boldsymbol{\eta}$ are updated with learning rate α^{CP}. The collected rewards are finally discarded when E^{CP} epochs are completed. The whole procedure is repeated until a total of I^{CP} iterations have taken place, with parameter I^{CP} determining the accuracy of the value function approximation.

5 Numerical Experiments

In this section, we numerically explore the performance of our approach, and collect managerial insights on the resulting policies. Section 5.1 presents the base experimental setting of our numerical study. Section 5.2 demonstrates the effectiveness of our approach in terms of final performance and computation time. Section 5.3 analyzes how the resulting policies employ each of the main decisions: pricing, driver repositioning, and customer admission. Finally, Section 5.4 examines the scalability of our approach.

5.1 Experimental Setting

Data and assumptions. Our numerical experiments are conducted on data made publicly available by the NYC Taxi and Limousine Commission (2023). The dataset consists of trips using Uber in Manhattan in February 2019, and divides the borough into 64 locations. For each trip, the dataset indicates the date on which the trip occurred, the pickup and drop-off times, the pickup and drop-off locations, and the trip distance. We curate the data by removing trips with incomplete or incoherent features, or trips that originate or end outside Manhattan. Furthermore, outliers are eliminated, i.e., trips whose time between pickup and drop-off is inferior to 30 seconds, or superior to 60 minutes.

The base experimental setting includes two simplifications, which allows interpretable results while still capturing the main features of a ride-hailing network. First, similar to Mao et al. (2020), we aggregate adjacent zones to form a network of 8 locations, as illustrated in Fig. 3. Second, we limit the study to morning peak hours (7 a.m. - 10 a.m.) on

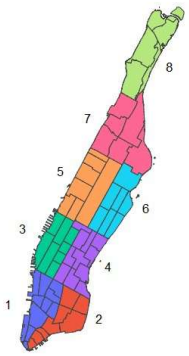


Figure 3: Partition of Manhattan into 8 locations.

Parameters	C^{lost}	C^{rep}	β^{cons}	β^{price}	β^{trans}	γ	D	$\overline{\Lambda}$
Values	0.4	0.2	0	6	1.5	1	900	1

Table 2: Default values of the model parameters.

Wednesday, Feb. 6. We divide this horizon into 36 intervals so that a decision epoch occurs every five minutes. The resulting dataset records 18,868 trips. In Section 5.4, we consider a larger network size (20 locations) and a longer time horizon (6 hours) to investigate the scalability of our approach.

Parameters. For the horizon considered, we extract the arrival rates of potential customers $\Lambda_{ij,t}$, expected transit times $1/\tau_{ij}$, and distances δ_{ij} from the dataset. To simplify exposition, we assume that the customer willingness to pay $F(\cdot)$ follows a uniform distribution within $[0, 1]$. Drivers in transit are supposed to be initially distributed according to past expected demand one hour before the start of the horizon, and available drivers are distributed according to future expected demand one hour after the start of the horizon. This follows the view that drivers can serve all customer requests before peak hours while anticipating future demand patterns.

Table 2 lists the default values of the remaining parameters. In our experiments, several instances are created by changing one of these values at a time. The lost sales costs, and repositioning cost rate are set to $C^{\text{lost}} = 0.4$, and $C^{\text{rep}} = 0.2$, representing about 20% of the maximum revenue for an average trip. The preference parameters are set to $\beta^{\text{cons}} = 0$, $\beta^{\text{price}} = 6$, and $\beta^{\text{trans}} = 1.5$, to reflect that drivers care more about high earnings than short transit times. The number of drivers in the network is roughly balanced with the demand using Little's law, such that $D = \frac{1}{2T}\sum_t \sum_i \sum_j (\Lambda_{ij,t}/\tau_{ij}) \approx 900$. The parameter $\overline{\Lambda}$ is a constant multiplying each arrival rate of potential customers (i.e., $\Lambda_{ij,t} \leftarrow \overline{\Lambda} \cdot \Lambda_{ij,t}$) to depict various demand conditions.

Experimental setup. Our approach is implemented with the programming language Python, and the open-source library Gym. The Expected Value Problem (EVP) is solved using the solver Gurobi, up to 1% optimality gap. Following limited manual tuning, our approach has been executed in all instances with the hyperparameters given in Appendix A. The implementation of the Proximal Policy Optimization (PPO) algorithm relies on the package Stable Baselines. During active learning, we evaluated the policy at regular intervals on 300 horizon runs. If the best average total profit had not changed more than 0.1% over five consecutive policy evaluations, then we considered the algorithm have converged. After deriving the policy, 300 horizon runs were again carried out to estimate all relevant metrics. All the computations were performed with NVIDIA A100 GPU, and 16 GB RAM. All data and codes can be accessed from `https://github.com/ThomasDeMunck12/TRL_RHNetworks`.

5.2 Approach Performance

This section aims to demonstrate the efficiency of our approach in terms of computation time and final performance. To this end, we compare our approach with alternative methods, including (i) two applications of the Proximal Policy Optimization (PPO) algorithm without pretraining,[7] and (ii) three rolling-horizon (RH) strategies.

The comparison between our approach and these methods is carried out through two metrics. The first metric, named the computation time difference, is expressed as $\%\text{Time(AM)} = \frac{CT^{\text{TL}} - CT^{\text{AM}}}{CT^{\text{AM}}} \times 100\%$, where CT^{TL} and CT^{AM} are the total computation times resulting from our transfer learning approach, and from one of the alternative methods. The second metric, named the profit difference, is defined as $\%\text{Profit(AM)} = \frac{TP^{\text{TL}} - TP^{\text{AM}}}{TP^{\text{AM}}} \times 100\%$, where TP^{TL} and TP^{AM} represent the average total profits found by our approach, and by one of the alternative methods.

Table 3 displays the computation time and profit differences between our approach, the applications of PPO without pretraining, and the RH strategies, for various instances. The instances are generated by varying one value at a time of the default parameters in Table 2. The computation time difference of our approach relative to RH strategies is not included, given that these strategies do not compute policies (i.e., mappings from all states to decisions) and thus entail negligible computation times compared with our approach (a few minutes vs. a few hours).

5.2.1 Comparison with PPO without Pretraining

The applications of *PPO without pretraining* use the PPO algorithm and do not pretrain the neural networks beforehand. The hyperparameter configuration is the same as in Appendix A, except for the number of iterations and the learning rate (denoted α^{PPO}) of the PPO algorithm. To ensure algorithm convergence, the maximum number of iterations is increased to 2×10^4 (instead of 6×10^3). Two learning rates α^{PPO} are considered: $\alpha^{\text{PPO}} = 3 \times 10^{-4}$ as recommended by Schulman et al. (2017), and $\alpha^{\text{PPO}} = 10^{-4}$, which shows to be better suited to our problem.

In Table 3, the benefits of pretraining appear clearly as our transfer learning approach improves profit by at least 16.7% on average. When $\alpha^{\text{PPO}} = 10^{-4}$, PPO without pretraining obtains a lower profit in around double the computation time than PPO with pretraining (see the computation time difference of −65%). When $\alpha^{\text{PPO}} = 3 \times 10^{-4}$, PPO without pretraining becomes faster but still slower than with pretraining (the average computation time difference going from −65% and −42%), while the final performance deteriorates (the average profit difference going from 16.7% to 23.4%). The fact that our approach outperforms PPO without pretraining can be further visualized in the learning curves in Fig. 4. In Fig. 4(a), we note that the PPO algorithm employed by our approach consistently outperforms PPO algorithms without pretraining. In addition, Fig. 4(b) shows that the PPO algorithm gradually improves upon the pretrained policy provided by our approach (i.e., the RH strategy with $T^{\text{RH}} = 4$ and $Q^{\text{RH}} = 12$).

Concerning the computation time difference, its variations between instances (usually ranging from −80% to −20%) reflect that the total training times are more stable with our approach than with PPO without pretraining. To further quantify these results, we measured the standard deviation of the computation times between instances. We observed a standard deviation that is 58% ($\alpha^{\text{PPO}} = 3 \times 10^{-4}$) and 156% ($\alpha^{\text{PPO}} = 10^{-4}$) higher for PPO without pretraining than for our approach. One possible reason for this result is that the pretraining process trains the neural networks to learn a unique policy and value function. In contrast, PPO updates the policy and value function many times by trial and

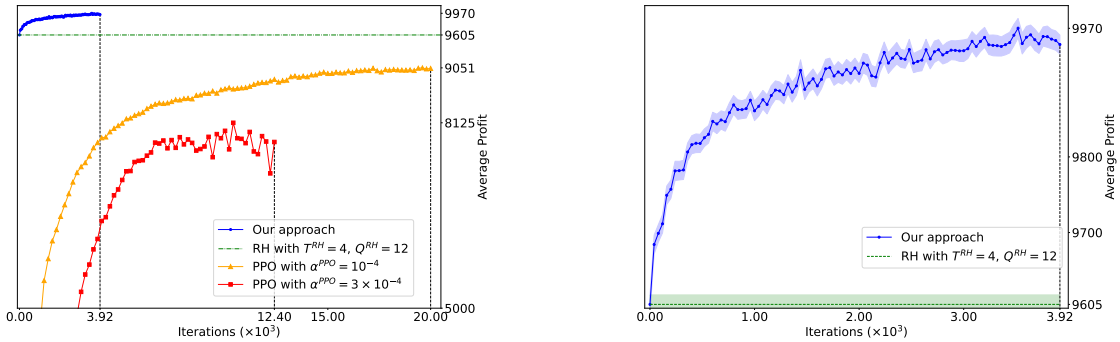
[7] Our approach was also compared to other DRL algorithms such as DDPG, TRPO, and TD3. The comparisons are not included in the experiments, given their notably worse performances.

	%Time - PPO		%Profit - PPO		%Profit - RH Strategies		
	α^{PPO}	α^{PPO}	α^{PPO}	α^{PPO}	$T^{\text{RH}} = 36$	$T^{\text{RH}} = 4$	$T^{\text{RH}} = 1$
Instances	$= 3 \times 10^{-4}$	$= 10^{-4}$	$= 3 \times 10^{-4}$	$= 10^{-4}$	$Q^{\text{RH}} = 36$	$Q^{\text{RH}} = 12$	$Q^{\text{RH}} = 12$
$\overline{\Lambda} = 0.5$	−35%	−66%	36.6%	55.8%	5.9%	0.1%	0.2%
$\overline{\Lambda} = 0.75$	−63%	−78%	25.7%	82.6%	6.1%	1.4%	1.2%
$\boldsymbol{\overline{\Lambda} = 1.0}$	**−42%**	**−72%**	**22.7%**	**10.2%**	**15.2%**	**3.9%**	**2.8%**
$\overline{\Lambda} = 1.25$	−67%	−56%	19.7%	*	19.2%	3.7%	2.6%
$\overline{\Lambda} = 1.5$	−34%	−73%	17.9%	7.3%	29.7%	3.1%	1.8%
$D = 450$	−69%	−59%	52.1%	8.3%	23.9%	2.6%	1.5%
$D = 675$	−59%	−61%	21%	10.2%	23.8%	4.1%	2.8%
$\boldsymbol{D = 900}$	**−42%**	**−72%**	**22.7%**	**10.2%**	**15.2%**	**3.9%**	**2.8%**
$D = 1125$	−43%	−79%	26.1%	8.5%	6.4%	1.6%	1.2%
$D = 1350$	−43%	−75%	21.6%	12.9%	4.7%	0%	−0.1%
$\gamma = 0.25$	−45%	−70%	20.8%	9.5%	13.4%	1.2%	0.4%
$\gamma = 0.5$	−18%	−68%	29%	36.7%	13.8%	1.4%	0.8%
$\boldsymbol{\gamma = 1}$	**−42%**	**−72%**	**22.7%**	**10.2%**	**15.2%**	**3.9%**	**2.8%**
$\gamma = 2$	−52%	−58%	2.6%	−1.2%	19.5%	16.6%	13.3%
$\gamma = 4$	110%	9%	−34%	*	51.7%	45.6%	33.9%
$\beta^{\text{trans}} = -1$	−47%	−80%	20.3%	10.9%	14.6%	1.9%	1.1%
$\beta^{\text{trans}} = -2$	−37%	−60%	17%	5.2%	37.8%	10.7%	8.6%
$\boldsymbol{\beta^{\text{trans}} = -1.5}$	**−42%**	**−72%**	**22.7%**	**10.2%**	**15.2%**	**3.9%**	**2.8%**
$\beta^{\text{cons}} = 3$, $\beta^{\text{price}} = 0$	−38%	−72%	30.6%	9.7%	18.9%	4.5%	3.4%
$\beta^{\text{cons}} = -5$, $\beta^{\text{trans}} = 0$	−59%	−71%	34.3%	9%	13.1%	4.1%	2.8%
$C^{\text{lost}} = 0$	−45%	−72%	18.6%	8%	7.3%	0.7%	0.2%
$C^{\text{lost}} = 0.2$	−49%	−59%	20%	6.3%	10.9%	1.9%	1.1%
$\boldsymbol{C^{\text{lost}} = 0.4}$	**−42%**	**−72%**	**22.7%**	**10.2%**	**15.2%**	**3.9%**	**2.8%**
$C^{\text{lost}} = 0.8$	−63%	−55%	26.7%	*	50.8%	6.1%	4.4%
$C^{\text{lost}} = 1.2$	−75%	−68%	53.3%	20.34%	146.1%	7.9%	5.6%
$C^{\text{rep}} = 0$	−34%	−67%	13.1%	5.9%	14.7%	3.8%	2.9%
$C^{\text{rep}} = 0.1$	−40%	−70%	14.8%	4.5%	11.9%	1.2%	0.3%
$\boldsymbol{C^{\text{rep}} = 0.2}$	**−42%**	**−72%**	**22.7%**	**10.2%**	**15.2%**	**3.9%**	**2.8%**
$C^{\text{rep}} = 0.3$	−60%	−77%	48.8%	39.7%	15.7%	4.3%	3.1%
$C^{\text{rep}} = 0.4$	−40%	−76%	25.4%	9.4%	19.4%	4.2%	3%
Mean	**−42%**	**−65%**	**23.4%**	**16.7%**	**23.5%**	**5.5%**	**4.0%**
Median	−45%	−70%	21.6%	9.2%	15.2%	3.7%	2.6%

Table 3: Comparison of the computation time and profit difference (in percentage) between our approach, PPO without pretraining, and RH strategies, for various model instances. The default instance, and the average values for all the instances are written in bold. If PPO without pretraining does not yield positive profits, the profit difference is replaced by "*".

error, making the learning process more volatile.

Concerning the profit difference, Table 3 shows that PPO with pretraining leads to significantly more efficient policies than PPO without pretraining for 23 out of 25 instances (for both PPO parameter configurations). Moreover, our approach always generates positive profits, an outcome not always reached by PPO without pretraining (cf. the (*) symbols in Table 3). This result further supports the idea that our approach stabilizes the learning process. The only two cases where our approach performs worse than PPO without pretraining occur when the temperature parameter γ is high, i.e., when many drivers elect to self-relocate. As RH strategies ignore drivers who self-relocate, they prescribe inadequate decisions (cf. the corresponding performances of RH strategies in Table 3) so that pretraining becomes counterproductive and delays learning.



(a) (b)

Figure 4: Learning curves of pretrained PPO (a, b) and PPO without pretraining (a), compared to the RH strategy with $T^{\text{RH}} = 4$, $Q^{\text{RH}} = 12$ (a, b). The shaded areas (b) represent 95% confidence intervals.

5.2.2 Comparison with RH Strategies

As discussed in Section 4.2.1, *RH strategies* can solve the EVP with different optimization intervals T^{RH} and optimization horizons Q^{RH}. Therefore, we benchmark our approach against three RH strategies. The first RH strategy produces static decisions by solving the EVP only at the first decision epoch ($T^{\text{RH}} = 36$) for the entire horizon ($Q^{\text{RH}} = 36$); the second RH strategy, which is used for actor pretraining, generates more dynamic decisions by solving the EVP every 4 epochs ($T^{\text{RH}} = 4$) over a horizon of 12 epochs ($Q^{\text{RH}} = 12$); the third RH strategy yields fully dynamic decisions by solving the EVP every epoch ($T^{\text{RH}} = 1$) over a horizon of 12 epochs ($Q^{\text{RH}} = 12$). Computational efforts grow from the first to the third RH strategy, due to the decrease of the optimization interval T^{RH}.

Table 3 notably indicates that our approach results in profits that are 4% higher on average than those of the best performing RH strategy ($T^{\text{RH}} = 1$, and $Q^{\text{RH}} = 12$). This RH strategy is the most dynamic one as it adapts its decisions at each epoch as a function of the state. It appears as a near-optimal baseline that is not easy to outperform. However, contrary to our approach, it solely relies on expected values to make its decisions. The 4% profit improvement hence reveals the benefits of taking stochastic variations into account, and the ability of our approach to seize them. As the importance of the uncertainty becomes higher, our approach yields increasingly better performances than the RH strategy (e.g., when the lost sales cost C^{lost} gets high).

Comparing our approach to the two other RH strategies, we observe that the profit difference grows even more on average (23.5% when $T^{\text{RH}} = 1$, $Q^{\text{RH}} = 36$; 5.5% when $T^{\text{RH}} = 4$, $Q^{\text{RH}} = 12$). This result shows the value of our approach in generating a dynamic policy, in opposition to more static RH strategies. It also confirms that our approach effectively uses PPO to improve upon the pretrained policy (a replication of the RH strategy with $T^{\text{RH}} = 4$, $Q^{\text{RH}} = 12$), although the pretrained policy already performs better than PPO alone (cf. Fig. 4(b)).

As a final comment, we remind that our approach generates a policy mapping all states to decisions over all decision epochs. As such, it leads to an output that can readily be executed once the training is over, so that no computation occurs during the actual operations. In contrast, RH strategies only seek decisions for a few states over some decision epochs. They need to periodically solve an optimization problem during operations, with limited computation time to make decisions. For this reason, we expect the performance of RH strategies to deteriorate in more complex settings. Our approach extend to larger applications, as shown in Section 5.4.

5.3 Policy Analysis

We now analyze the policies found by our approach to comprehend the use of pricing, driver repositioning, and customer admission. For each decision, we mainly look at the effect on the flow of customers and drivers entering (inflow) or leaving (outflow) each location. The net flow, which is the difference between total inflow and total outflow over the horizon, offers a simple measure of the imbalances in a location. A positive net flow indicates that more customers or drivers are entering the location than leaving it, whereas a negative net flow means the opposite. The absolute sum of the net flows in all the locations characterizes the total network imbalances.

5.3.1 Pricing Decision

For the default instance, Table 4 shows the average net flows of customer requests for each location under a policy that sets a single price $p = 0.5$, and under the policy resulting from our approach. The table also displays the average price prescribed by our policy in each location. In Table 4, we observe that the net flows of customer requests decrease substantially due to the pricing decision. In the whole network, the pricing decision reduces imbalances, with the absolute sum of net flows falling by roughly 40%. This observation illustrates the *spatial role* of the pricing decision in balancing demand between network locations. As shown in Table 4, the average prices are higher in locations with negative net flows and lower in locations with positive net flows. Consequently, more requests start from locations with positive net flows and end in locations with negative net flows, thereby reducing demand imbalances.

Besides its spatial role, the pricing decision also has a *temporal role*. Fig. 5 illustrates this role by presenting the evolution over time of the arrival rate of potential customers (a), the average price (b), and the arrival rate of customer requests (c) in the fourth location (the location with the largest demand imbalances), for various magnitudes of demand. In Fig. 5(a), we note that higher level of demands leads to more pronounced demand peaks. In response, prices are adjusted more dynamically over time so as to follow the demand peaks (cf. Fig. 5(b)). As a result, stable arrival rates of customer requests are obtained throughout the horizon (cf. Fig. 5(c)), regardless of the demand magnitude. This allows the platform to avoid temporary shortages of drivers, meaning that actual customer requests have higher likelihood to be served at any time.

Looking closely at Fig. 5(b), it can be noticed that the 95% prediction price intervals (between brackets) grow with the arrival rates of potential customers (0.13 when $\overline{\Lambda} = 0.5$ vs. 0.29 when $\overline{\Lambda} = 1.5$, on average). This variation of prices between horizon runs suggests that the pricing decision may exert a *state-dependent role*. That is, prices are adapted as a function of the observed state (i.e., the number of available drivers) to fit demand with supply in real time. If the number of drivers in a location is lower than expected, then the price is raised to discourage customer requests, and conversely. As a result, the number of available drivers becomes more stable, protecting the platform against stochastic fluctuations. This effect gains in importance with high demand, as drivers get more scarce.

Customer requests	Loc. 1	Loc. 2	Loc. 3	Loc. 4	Loc. 5	Loc. 6	Loc. 7	Loc. 8	Abs. sum
With Single Pricing	-52	-355	218	1104	-462	-365	-79	-8	2643
With Pricing Decision	27	-187	136	631	-299	-225	-69	-15	1589
Average Price	0.62	0.65	0.57	0.52	0.64	0.62	0.59	0.56	

Table 4: Average net flows of customer requests under static pricing ($p = 0.5$), and under the pricing decision obtained by our approach, in each location, for the default instance. The last row indicates the average prices decided in each location.

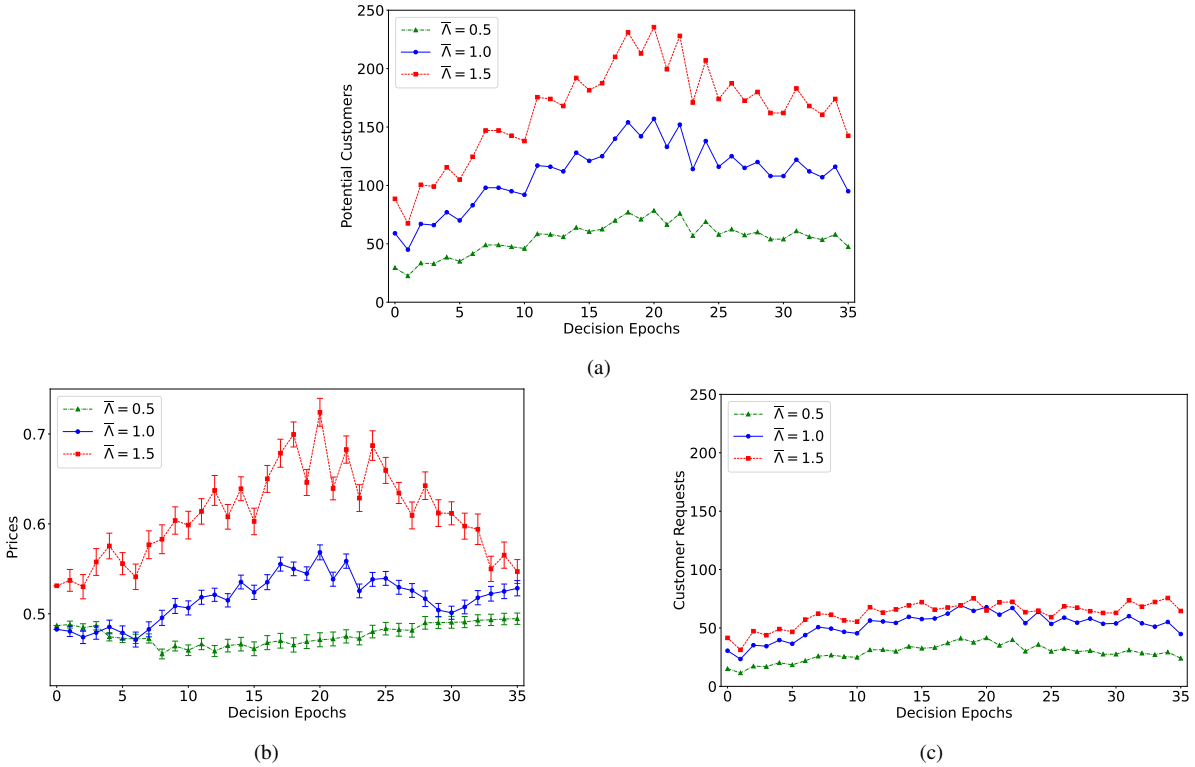


Figure 5: Evolution of the average prices (the brackets around the average prices represent 95% prediction intervals over 300 horizon runs) (a), the arrival rates of potential customers (b), and the arrival rates of customer requests (c), in the fourth location, for different arrival rates of potential customers.

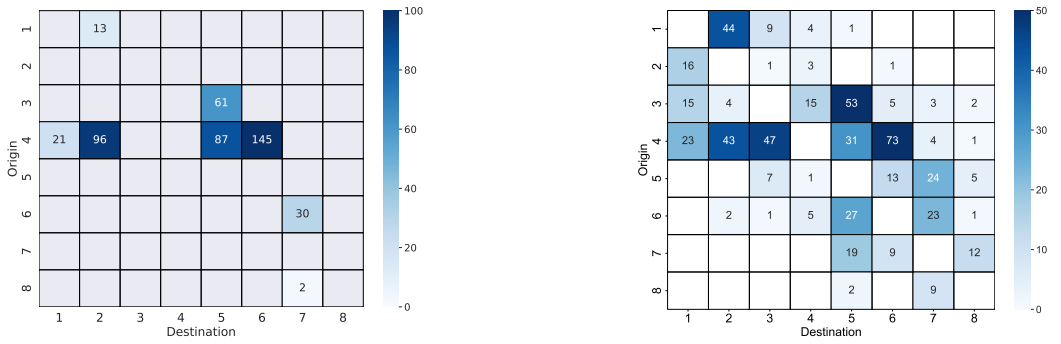
	Loc. 1	Loc. 2	Loc. 3	Loc. 4	Loc. 5	Loc. 6	Loc. 7	Loc. 8	Abs. Sum
Dispatched Drivers	23	-174	114	583	-272	-207	-61	-7	1441
Repositioned Drivers	8	109	-61	-349	148	115	32	-2	824
Self-Relocating Drivers	-3	72	-32	-194	83	42	23	8	457

Table 5: Average net flows of dispatched, repositioned, and self-relocating drivers, in each location for the default instance. The last column gives the absolute sum of the average net flows over the network.

5.3.2 Repositioning Decision

We investigate the repositioning decision by looking at Table 5, which shows, once again for the default instance, the average net flows of dispatched drivers, repositioned drivers and self-relocating drivers for each location. We note that the net flows of dispatched drivers present significant imbalances, as illustrated by the absolute sum of net flows being equal to 1441. These imbalances, comparable to those of customer requests (cf. Table 4), can cause excessive or scarce driver supply in some locations. Driver repositioning and driver self-relocating are crucial for rebalancing the network, sending drivers from locations with sufficient supply to locations with limited supply. For instance, in the fourth location, the number of dispatched drivers arriving exceeds the number of dispatched drivers departing by 583 (cf. Table 5). To offset this imbalance, our policy induces an average of 394 drivers to reposition and 194 drivers to self-relocate from this location. This permits the platform to satisfy 96% of the requests on average.

Interestingly, Table 5 reports that the net flows of repositioned drivers are often higher than those of self-relocating drivers. This result suggests that the repositioning decision is more effective to address network imbalances than driver self-relocating. We explain this result by the fact that the repositioning decision allows for a more immediate way



(a) (b)

Figure 6: Average number of repositioned (a) and self-relocating drivers (b) over the time horizon for each origin-destination pair, for the default instance. All the values are rounded down to the nearest integer. Values rounded to 0 are omitted for conciseness.

	Loc. 1	Loc. 2	Loc. 3	Loc. 4	Loc. 5	Loc. 6	Loc. 7	Loc. 8	Abs. sum
Customer Requests	27	-187	136	631	-299	-225	-69	-15	1589
Admitted Cust. Req.	24	-181	135	612	-296	-208	-71	-16	1543

Table 6: Average net flows of customer requests, and admitted customer requests, in each location for the default instance. The last column gives the absolute sum of the average net flows over the network.

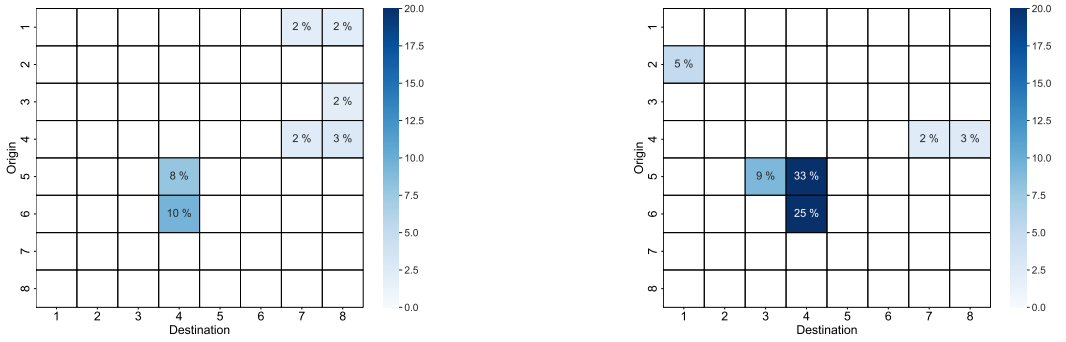
to rebalance driver supply than driver self-relocating. Despite being costly, driver repositioning is directly controlled by the platform. In contrast, driver self-relocating is indirectly controlled by the pricing decision (which also affects customer demand) and depends on factors beyond the platform control (e.g., the distance between locations).

Fig. 6 further elaborates the previous insight by displaying the average number of repositioned (a) and self-relocating drivers (b) for each pair of locations. In Fig. 6 (a), drivers are repositioned only on specific pairs of locations, from locations with excess of supply (e.g., the third and fourth locations) to locations with shortage of supply (e.g., the fifth and sixth locations). Conversely, in Fig. 6 (b), drivers self-relocate for much more origin-destination pairs, and even from locations with scarce supply to locations with excessive supply, making less appropriate choices. In total, more drivers (562) self-relocate than are repositioned (455). We thus conclude that the repositioning decision rebalances the network more accurately than driver self-relocating, while involving fewer drivers.

5.3.3 Admission Decision

To examine the admission decision, Table 6 displays the average net flows of customer requests and admitted customer requests in each location for the default instance. We observe that the admission decision reduces the absolute sum of net flows from 1589 to 1543 (a fall of about 3%). Overall, the admission decision has a limited impact on the customer net flows and demand imbalances, contrary to the pricing decision. This lack of customer rejection comes from the fact that it entails both a cost and a revenue loss. In contrast, higher prices prevent some customers from entering the system at no cost while increasing revenues from remaining customer requests.

When the number of requests is excessive, the admission decision can still be relevant since it allows for rejecting requests according to their origin and destination (contrary to prices that only depend on the customer origin). To illustrate this, Fig. 7 plots the average percentage of rejected customers for each pair of locations when customer demand is high ($\overline{\Lambda} = 1.5$) (a), or when driver supply is low ($D = 450$) (b). We see that the admission decision avoids sending drivers from locations with scarce supply (e.g., the fifth location) to locations with excessive supply (e.g., the fourth



(a) (b)

Figure 7: Average percentage of rejected customers for each origin-destination pair of locations when $\overline{\Lambda} = 1.5$ (a), and when $D = 450$ (b). For conciseness, values equal to 0% are omitted.

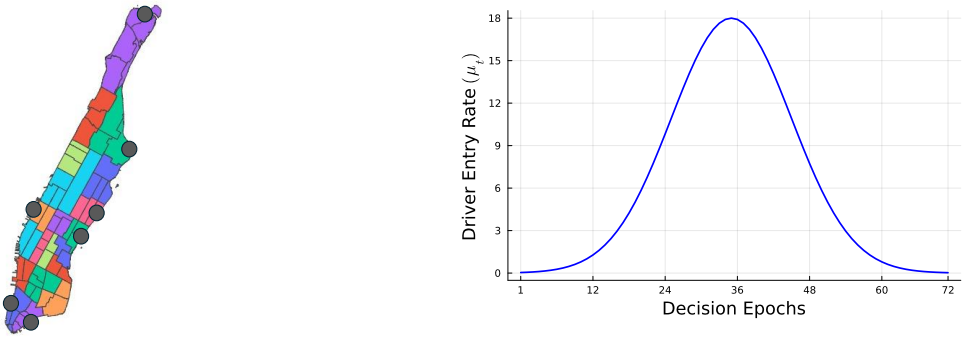
location), permitting finer control than prices. Because drivers are busy serving customers most of the time (e.g., the driver utilization rate is equal to 93% when $D = 450$), few drivers are available for repositioning. Thus, the admission decision acts as a substitute for driver repositioning, rejecting customers going to undesirable locations.

5.4 Extension to Large-Scale Applications

In this section, we examine the performance of our approach in a larger setting. We do so by making four modifications to the base experimental settings of Section 5.1. First, the ride-hailing network is extended to 20 locations, illustrated in Fig. 8(a). Second, the horizon is broadened to 72 decision epochs, occurring every five minutes from 6 a.m. to 12 p.m. Third, we relax the assumption that there is a constant number of drivers in the network. Instead, only half of the drivers are operating at the beginning of the horizon, and the rest of the drivers enter the network according to a non-stationary Poisson process with rate μ_t, as shown in Fig. 8(b). The random number of drivers in the network at epoch t is denoted by D_t. Assuming that drivers come mainly from other boroughs than Manhattan, drivers are only allowed to enter the network from seven entry points (cf. the gray dots in Fig. 8(a)), which correspond to bridges linking Manhattan to the remaining of the city. Fourth, since real-world applications are typically characterized by variable customer arrival rates, our approach first uses training data from 20 weekdays of February 2019 to learn a policy. Then, test data from 20 weekdays of March 2019 are used to evaluate the policy performance with new arrival rates that are unknown at time of training. For each test day, 50 horizon runs are simulated with random customer requests generated from the corresponding arrival rates to evaluate the performance of the learned policy.

The analysis consists of comparing our transfer learning approach with the best rolling-horizon (RH) strategy from Section 5.2 (with $T^{\text{RH}} = 1$, $Q^{\text{RH}} = 12$). Given the extensive size of the problem, the PPO algorithm does not reach sufficient performance without pretraining and is excluded from the comparison.[8] To quantify the benefits of our approach, we rely on three metrics. The first metric is the total profit of the platform over the whole horizon, $\text{TP} = \sum_t R_t(\mathbf{s_t}, \mathbf{a_t})$. The second metric is the driver utilization rate $\text{UR} = \frac{1}{T}\sum_t \sum_i \sum_j (d^{\text{dis}}_{ij,t}/D_t \cdot \tau_{ij})$, which evaluates the proportion of time that drivers are busy serving customers. The third metric is the request fulfillment rate $\text{FR} = \frac{1}{T}\sum_t (\sum_i \sum_j d^{\text{dis}}_{ij,t})/(\sum_i \sum_j c^{\text{req}}_{ij,t})$, which assesses the proportion of requests that are effectively satisfied.

[8]The PPO algorithm was implemented with the same set of hyperparameters as in Section 5.2 (and $\alpha^{\text{PPO}} = 10^{-4}$), leading to an average profit 53% lower after a training time more than twice longer, in comparison with our approach.



(a) (b)

Figure 8: Partition of Manhattan into 20 locations with driver entry points described by gray dots (a). Aggregate rate μ_t of drivers entering the network at each decision epoch (b).

Fig. 9 displays the total profit for our transfer learning approach and the RH strategy for each test day. The Whisker plots summarize the percentiles and average values obtained from the 50 horizon runs that evaluate the learned policy. We observe that our approach generates substantially higher profits than the RH strategy, increasing the profit by 6% on average. This result demonstrates the scalability of our approach, as the latter can be applied to larger settings with novel forms of system dynamics. These profit gains are generally more significant than those reported in Section 5.2, where the same parameters were used for training and testing. One probable reason for this improved performance lies in the ability of our approach to adapt to new situations. By leveraging DRL, our approach can learn a policy that generalizes well to new states not encountered in the training data (e.g., days with higher demand). In contrast, the RH strategy relies solely on expected values from historical data to make decisions. When test data significantly differ from data previously observed, this dependence causes the RH strategy to make inappropriate decisions, resulting in profit losses.

Fig. 10 complements the analysis by plotting the average driver utilization rate and request fulfillment rate for our approach and the RH strategy for each test day. Fig. 10 shows that our approach can also benefit to drivers and customers besides maximizing profit. On the one hand, driver utilization rates are consistently higher with our approach than with the RH strategy (5% on average). This result is desirable as it reveals that drivers spend less time idle, improving their earnings. It also suggests that more customers are served in total. On the other hand, the request fulfillment rate is raised, compared to the RH strategy (2% on average). That is, customer requests are more likely to be satisfied with our approach. The only cases where our approach has a lower fulfillment rate than the RH strategy (i.e., days 11, 12, and 16) occur during days of low demand. However, for these days, the fulfillment rate is already high, and the difference between our approach and the RH strategy remains marginal.

6 Conclusion

This paper considers a ride-hailing platform that balances driver supply with customer demand over multiple locations. Impatient customers arrive with distinct willingness to pay and decide whether to request services based on prices. Drivers can elect to self-relocate between locations according to their preferences regarding prices and transit times. In this broad setting, the platform makes pricing, customer admission, and driver repositioning decisions.

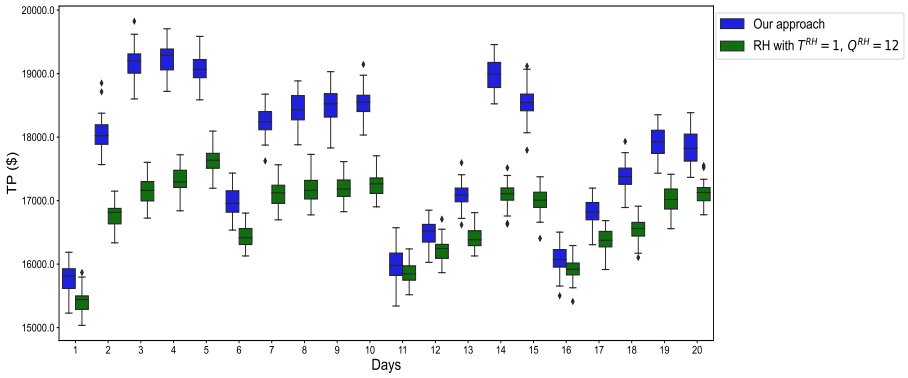


Figure 9: Comparison of the total profit (TP) between our approach and the RH strategy ($T^{\text{RH}} = 1,\ Q^{\text{RH}} = 12$) over 20 test days.

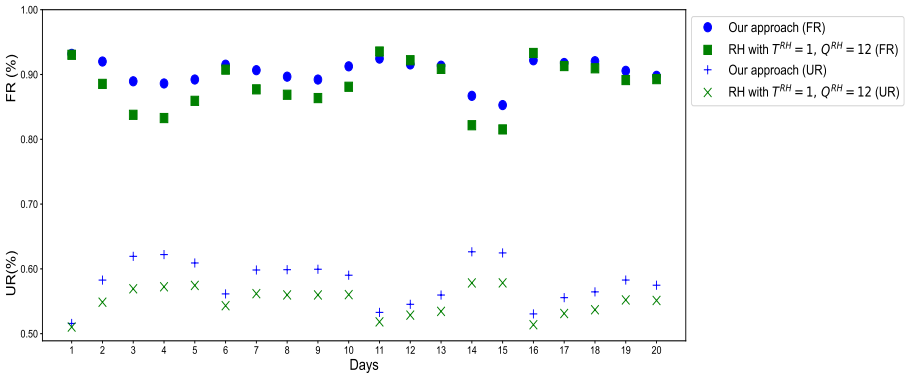


Figure 10: Comparison of the average driver utilization rate (UR), and request fulfillment rate (FR) between our approach and the RH strategy ($T^{\text{RH}} = 1,\ Q^{\text{RH}} = 12$) over 20 test days.

We formulate the problem as a Markov decision process that determines the price in each location, and the number of repositioned drivers and admitted customers between locations. Given the problem complexity, we develop a transfer learning approach that combines mathematical optimization with deep reinforcement learning. Our approach first designs a rolling-horizon (RH) strategy that makes decisions based on an expected-value formulation of the problem. Then, two neural networks are pretrained to reproduce the decisions prescribed by the RH strategy, and approximate the value function of the associated policy. Finally, the Proximal Policy Optimization (PPO) algorithm is applied to further improve the policy.

Using data from New York City, we demonstrate that our approach leads to a stable learning process characterized by improved performances. On average, our approach finds a policy that increases the platform's profit by 16.7% while reducing the total training time by 42% – 65% in comparison with the PPO algorithm without pretraining. Moreover, our approach outperforms RH strategies by 4% at least, while the latter are known to work well in practice. Our approach is also shown to generalize to settings with many locations, many decision epochs, and changing demand parameters. Overall, our work contributes to the literature by successfully applying a pretraining procedure to a complex problem. We hope to inspire more research that leverages mathematical optimization with DRL in the future.

By analyzing the resulting policy, we explore managerial insights on pricing, driver repositioning, and customer admission. Our main results are summarized as follows:

- The pricing decision plays three roles in balancing customer demand and driver supply. First, it adjusts prices

spatially to balance the flows of customer requests between locations. Second, it adjusts prices temporally to stabilize the number of customer requests in each location over time. Third, to a lesser extent, it adjusts prices as a function of the system state to match requests with drivers in real time.

- The repositioning decision can effectively mitigate supply-demand imbalances. Furthermore, it appears more effective than driver self-relocating because it allows the platform to control the driver rebalancing fully. In contrast, driver self-relocating is indirectly induced by the pricing decision and depends on the transit times between locations.
- In general, the use of the admission decision is limited. Nevertheless, when driver supply is scarce, customer admission becomes relevant. On the one hand, it balances the network flows more finely than the pricing decision. On the other hand, it serves as a way to indirectly reposition drivers when the latter are too busy serving requests and cannot be directly repositioned.

Our work leaves several avenues for further research. First, for the sake of tractability, we made several assumptions about the problem, such as drivers always complying with the platform's repositioning decision and transit times that follow exponential distributions. It would be interesting (yet challenging) to relax these assumptions and explore their related implications. Second, our model only makes decisions at the tactical level, leaving operational decisions to other optimization models. In practice, however, decisions from the two levels interact with each other. A challenge to undertake would be to develop a model integrating both types of decisions. Finally, this paper applies our approach to a ride-hailing network. Nevertheless, we believe the approach is sufficiently general to be used in other contexts where heuristics are readily available (e.g., inventory management and vehicle routing). It would be worth adapting our approach to other settings where DRL algorithms can be pretrained from available heuristics.

Funding

The ARC project "Platform Regulation and Operations in the Sharing Economy" (Project No. ARC 08-19-004) supported this research. Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2-5020-11 and by the Walloon Region.

A Hyperparameters

Hyperparameters	Values	Hyperparameters	Values
Neural networks		**Rolling-horizon strategy**	
Hidden layers	4	Optimization interval (T^{RH})	4
Nodes per hidden layer	128	Optimization horizon (Q^{RH})	12
Activation function	tanh	**Actor pretraining**	
Proximal Policy Optimization (PPO)		Cardinality of the state-decision set ($\lvert\mathcal{D}^{\mathrm{RH}}\rvert$)	10^5
Simulation horizon	2400	Mini-batch size	400
Mini-batch size	400	Learning rate (α^{AP})	10^{-4}
Learning rate (α^{PPO})	5×10^{-6}	Training epochs (E^{AP})	100
Clipping range (ε)	0.2	**Critic pretraining**	
Training epochs	10	Simulation horizon (T^{CP})	2400
Iterations between evaluations	50	Mini-batch size	400
Horizon runs during evaluation	500	Learning rate (α^{CP})	2×10^{-3}
Minimum number of iterations	2000	Training epochs (E^{CP})	10
Maximum number of iterations	6000	Number of iterations (I^{CP})	150

Table 7: Values of the hyperparameters of the transfer reinforcement learning approach.

References

Afèche, P., Liu, Z., & Maglaras, C. (2023). Ride-hailing networks with strategic drivers: The impact of platform control capabilities on performance. *Manufacturing & Service Operations Management*, *25*(5), 1890–1908.

Al-Kanj, L., Nascimento, J., & Powell, W. B. (2020). Approximate dynamic programming for planning a ride-hailing system using autonomous fleets of electric vehicles. *European Journal of Operational Research*, *284*(3), 1088–1106.

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., & Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, *114*(3), 462–467.

Bai, J., So, K. C., Tang, C. S., Chen, X., & Wang, H. (2019). Coordinating supply and demand on an on-demand service platform with impatient customers. *Manufacturing & Service Operations Management*, *21*(3), 556–570.

Banerjee, S., Freund, D., & Lykouris, T. (2022). Pricing and optimization in shared vehicle systems: An approximation framework. *Operations Research*, *70*(3), 1783–1805.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*(5), 834–846.

Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics*, 679–684.

Beojone, C. V., & Geroliminis, N. (2023). Relocation incentives for ride-sourcing drivers with path-oriented revenue forecasting based on a markov chain model. *Transportation Research Part C: Emerging Technologies*, *157*, 104375.

Beojone, C. V., Zhu, P., Sirmatel, I. I., & Geroliminis, N. (2024). A hierarchical control framework for vehicle repositioning in ride-hailing systems. *Transportation Research Part C: Emerging Technologies*, 104717.

Bimpikis, K., Candogan, O., & Saban, D. (2019). Spatial pricing in ride-sharing networks. *Operations Research*, *67*(3), 744–769.

Birge, J. R., & Louveaux, F. (2011). *Introduction to Stochastic Programming*. Springer Science & Business Media.

Boute, R. N., Gijsbrechts, J., Van Jaarsveld, W., & Vanvuchelen, N. (2022). Deep reinforcement learning for inventory control: A roadmap. *European Journal of Operational Research*, *298*(2), 401–412.

Braverman, A., Dai, J. G., Liu, X., & Ying, L. (2019). Empty-car routing in ridesharing systems. *Operations Research*, *67*(5), 1437–1452.

Chen, C., Yao, F., Mo, D., Zhu, J., & Chen, X. M. (2021). Spatial-temporal pricing for ride-sourcing platform with reinforcement learning. *Transportation Research Part C: Emerging Technologies*, *130*, 103272.

Chen, Q., Lei, Y., & Jasin, S. (2023). 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.

De Moor, B. J., Gijsbrechts, J., & Boute, R. N. (2022). Reward shaping to improve the performance of deep reinforcement learning in perishable inventory management. *European Journal of Operational Research*, *301*(2), 535–545.

De Munck, T., Chevalier, P., & Tancrez, J.-S. (2023). Managing priorities on on-demand service platforms with waiting time differentiation. *International Journal of Production Economics*, *266*, 109053.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning* (pp. 1861–1870).

Haliem, M., Mani, G., Aggarwal, V., & Bhargava, B. (2021). A distributed model-free ride-sharing approach for joint matching, pricing, and dispatching using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, *22*(12), 7931–7942.

Hosseini, M., Milner, J., & Romero, G. (2024). Dynamic relocations in car-sharing networks. *Operations Research, forthcoming*.

Jiao, Y., Tang, X., Qin, Z. T., Li, S., Zhang, F., Zhu, H., & Ye, J. (2021). Real-world ride-hailing vehicle repositioning using deep reinforcement learning. *Transportation Research Part C: Emerging Technologies*, *130*, 103289.

Kanoria, Y., & Qian, P. (2024). Blind dynamic resource allocation in closed networks via mirror backpressure. *Management Science*, *70*(8), 5445–5462.

Kullman, N. D., Cousineau, M., Goodson, J. C., & Mendoza, J. E. (2022). Dynamic ride-hailing with electric vehicles. *Transportation Science*, *56*(3), 775–794.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Li, S., Yang, H., Poolla, K., & Varaiya, P. (2021). Spatial pricing in ride-sourcing markets under a congestion charge. *Transportation Research Part B: Methodological*, *152*, 18–45.

Lu, A., Frazier, P. I., & Kislev, O. (2018). Surge pricing moves Uber's driver. In *Proceedings of the 2018 ACM Conference on Economics and Computation* (pp. 3–3).

Lyft. (2021). *Bonus zones - The Lyft Driver Blog.* Retrieved from `https://www.lyft.com/hub/posts/bonus-zones/` (2023/02/13)

Lyft. (2024). *Receiving airport FIFO pickup requests.* Retrieved from `https://help.lyft.com/hc/en-us/all/articles/115012922787-Receiving-Airport-FIFO-pickup-requests` (2024/02/13)

Mao, C., Liu, Y., & Shen, Z.-J. M. (2020). Dispatch of autonomous vehicles for taxi services: A deep reinforcement learning approach. *Transportation Research Part C: Emerging Technologies*, *115*, 102626.

McCormick, G. P. (1976). Computability of global solutions to factorable nonconvex programs: Part I—Convex underestimating problems. *Mathematical programming*, *10*(1), 147–175.

Meskar, M., Aslani, S., & Modarres, M. (2023). Spatio-temporal pricing algorithm for ride-hailing platforms where drivers can decline ride requests. *Transportation Research Part C: Emerging Technologies*, *153*, 104200.

Nourinejad, M., & Ramezani, M. (2020). Ride-sourcing modeling and pricing in non-equilibrium two-sided markets. *Transportation Research Part B: Methodological*, *132*, 340–357.

NYC Taxi and Limousine Commission. (2023). *Tlc trip record data.* Retrieved from `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page` (2023/09/03)

Oroojlooyjadid, A., Nazari, M., Snyder, L. V., & Takáč, M. (2022). A deep Q-network for the Beer Game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management*, *24*(1), 285–304.

Özkan, E. (2020). Joint pricing and matching in ride-sharing systems. *European Journal of Operational Research*, *287*(3), 1149–1160.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the Curses of Dimensionality* (Vol. 703). John Wiley & Sons.

Qian, X., Guo, S., & Aggarwal, V. (2022). Drop: Deep relocating option policy for optimal ride-hailing vehicle repositioning. *Transportation Research Part C: Emerging Technologies*, *145*, 103923.

Qin, Z. T., Zhu, H., & Ye, J. (2022). Reinforcement learning for ridesharing: An extended survey. *Transportation Research Part C: Emerging Technologies*, *144*, 103852.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *Working Paper, University of California, Berkeley*.

Shou, Z., Di, X., Ye, J., Zhu, H., Zhang, H., & Hampshire, R. (2020). Optimal passenger-seeking policies on e-hailing platforms using markov decision process and imitation learning. *Transportation Research Part C: Emerging Technologies*, *111*, 91–113.

Silver, D., Huang, A., Maddison, C. J., Guez, A., & Sifre, L. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Sun, L., Teunter, R. H., Babai, M. Z., & Hua, G. (2019). Optimal pricing for ride-sourcing platforms. *European Journal of Operational Research*, *278*(3), 783–795.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.

Talluri, K. T., Van Ryzin, G., & Van Ryzin, G. (2004). *The Theory and Practice of Revenue Management* (Vol. 1). Springer.

Taylor, M., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, *10*(7).

Tuncel, K., Koutsopoulos, H. N., & Ma, Z. (2023). An integrated ride-matching and vehicle-rebalancing model for shared mobility on-demand services. *Computers & Operations Research*, *159*, 106317.

Turan, B., Pedarsani, R., & Alizadeh, M. (2020). Dynamic pricing and fleet management for electric autonomous mobility on demand systems. *Transportation Research Part C: Emerging Technologies*, *121*, 102829.

Uber. (2022). *What is surge pricing?* Retrieved from `https://help.uber.com/driving-and-delivering/article/what-is-surge?nodeId=e9375d5e-917b-4bc5-8142-23b89a440eec` (2024/02/13)

Uber. (2023). *Delivery heatmap: Finding busy areas.* Retrieved from `https://help.uber.com/en/driving-and-delivering/article/delivery-heatmap-finding-busy-areas?nodeId=88c4fc0e-278b-4d05-bd53-cc1ce7f0cc0b` (2024/02/13)

Wang, G., Zhang, H., & Zhang, J. (2024). On-demand ride-matching in a spatial model with abandonment and cancellation. *Operations Research*, *72*(3), 1278–1297.

Wang, H., & Yang, H. (2019). Ridesourcing systems: A framework and review. *Transportation Research Part B: Methodological*, *129*, 122–155.

Yan, P., Yu, K., Chao, X., & Chen, Z. (2023). An online reinforcement learning approach to charging and order-dispatching optimization for an e-hailing electric vehicle fleet. *European Journal of Operational Research*, *310*(3), 1218–1233.

Zhu, Z., Ke, J., & Wang, H. (2021). A mean-field Markov decision process model for spatial-temporal subsidies in ride-sourcing markets. *Transportation Research Part B: Methodological*, *150*, 540–565.