

UNIVERSITÀ
DEGLI STUDI
DI PADOVA



UNIVERSITY OF PADUA
Department of Information Engineering
Master's degree in "Control Systems Engineering"

A MARL Framework for Rebalancing Dockless Bike-Sharing Systems

MASTER CANDIDATE

Alessia Vitaliani

Student ID 2158288

SUPERVISOR

Professor Gian Antonio Susto

CO-SUPERVISORS

Dr. Matteo Cederle

Professor Federico Chiariotti

Professor Marco Fabris

Dr. Edoardo Scarpel

A chi ha imparato a ballare sotto la pioggia,
aspettando il momento di fiorire.

Abstract

The dockless Bike-Sharing Systems (BSSs) have grown in recent decades, serving as a means of urban mobility capable of reducing traffic congestion and environmental pollution. Compared to conventional docked systems, these systems simplify the process for users, allowing them to pick up and return bicycles anywhere within the service area. However, the lack of physical stations and the asymmetry of demand lead to spatiotemporal imbalances: the accumulation of bicycles at certain points and their complete absence at others severely compromises the system's effectiveness and reliability.

To address this issue, this thesis proposes a solution based on dynamic rebalancing that utilizes Multi-Agent Reinforcement Learning (MARL) techniques. The problem has been formulated as a Markov Decision Process (MDP), in which a fleet of intelligent agents controls the movements and logistical operations (pickup, delivery, recharging) of vehicles tasked with rebalancing demand. To represent the complex topology and operational dynamics of the road network, the simulation environment was modeled using a graph structure, dividing the area into cells. The system state is processed via a graph neural network architecture, which allows agents to extract spatial features and identify critical areas in real time.

As a case study, the model was tested and validated on a real-world road network in Manhattan (New York, USA). Numerous simulation tests were conducted to assess the agent's learning ability and improve its performance, demonstrating the effectiveness of the proposed approach through a reduction in service failures and an even distribution of vehicles across the area.

Contents

List of Acronyms	ix
1 Introduction	1
2 Theoretical Background	5
2.1 Basics of Reinforcement Learning	5
2.1.1 Tabular Q-learning	7
2.1.2 Neural Networks	8
2.2 Deep Reinforcement Learning	9
2.3 Proximal Policy Optimization	12
2.4 Multi-Agent Reinforcement Learning	14
2.4.1 Multi-Agent Actor-Critic Framework	16
2.4.2 Multi-Agent PPO	17
3 State of the Art	19
3.1 Rebalancing in Bike-Sharing Systems	19
3.2 Reinforcement Learning in Shared Mobility	21
3.3 Fleet Management through Multi-Agent RL	22
4 System Model	25
4.1 System Representation	25
4.2 Modeling User Demand	27
4.3 Battery Dynamics	29
4.4 The Multi-Area Setting	30
5 Proposed Framework	31
5.1 Environment and Data Pipeline	31
5.2 Problem Formulation	34
5.3 PPO with Graph Attention Networks	38
5.4 Scaling to Multiple Areas	40

6 Experimental Validation 41

6.1 Simulation Setup 41

6.2 Comparison between Static and Dynamic Rebalancing 45

6.3 Analysis of the Results 48

7 Conclusions 57

7.1 Future Work 58

A System Model Notation 59

Acknowledgments 61

References 69

List of Acronyms

Acronym	Definition
ANN	Artificial Neural Network
BSS	Bike-Sharing System
CTCE	Centralized Training Centralized Execution
CTDE	Centralized Training Decentralized Execution
CTMC	Continuous-Time Markov Chain
DDQN	Double Deep Q-Network
DP	Dynamic Programming
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DRP	Dynamic Repositioning Problem
DTDE	Distributed Training Decentralized Execution
DTMC	Discrete-Time Markov Chain
GAE	Generalized Advantage Estimation
GAT	Graph Attention Network
GATv2	Graph Attention Network v2
GNN	Graph Neural Network
GPS	Global Positioning System
IDW	Inverse Distance Weighting
KL	Kullback-Leibler
MAPPO	Multi-Agent Proximal Policy Optimization
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
MFRL	Mean Field Reinforcement Learning
MLP	Multi-Layer Perceptron
OR	Operations Research
PMF	Probability Mass Function

Acronym	Definition
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SR	Static Rebalancing
SRP	Static Repositioning Problem
TD	Temporal Difference
TRPO	Trust Region Policy Optimization
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem

1

Introduction

Bike-Sharing Systems (BSSs) are micro-mobility platforms that offer a flexible, sustainable, cost-effective, and on-demand transportation alternative for short-to-medium distance travel. The development of BSSs was aimed at lowering the usage of private transport vehicles, solving the problems related to traffic pollution and minimizing the emission of greenhouse gases in metropolitan cities. BSSs have been incorporated in the framework of urban transportation systems. Along with public transport such as subways, railway transport and buses, bike-sharing has become one of the most widely adopted modes of urban mobility [1]. The integration of these two means of transport promotes the creation of an intermodal transportation system, where individuals can take advantage of the large capacity of public transport for traveling great distances and the wide availability of bike-sharing for the beginning or end of their trip [2, 3].

The classification of BSSs is determined by their structure [4]:

- Station-based BSSs: These are the traditional and most common form, where bikes should be rented and returned only from specific physical stations. The organization of such systems provides high orderliness of city life; however, it ties the users to the existence of the corresponding infrastructure.
- Dockless BSSs: These have no fixed stations, users can rent and return bikes wherever they want, enabled by dedicated smartphone application, the Global Positioning System (GPS) on every bike, and internet to locate the bikes. While offering the greatest flexibility for the users, such systems cause serious problems in maintaining order (e.g., unauthorized parking).

Despite their benefits, BSSs face substantial socio-spatial and operational challenges. From a geographical perspective, they often suffer from severe spatial inequities, with fleets clustering in affluent central districts while neglecting suburban and disadvantaged communities. Additionally, daily commuting patterns induce highly asymmetric



(a) Station-based BSSs.



(b) Dockless BSSs.

Figure 1.1: Types of BSSs in the world.

traffic flows: a massive influx of bikes towards urban centers in the morning is typically followed by a prominent exodus in the afternoon. To counteract this fluctuating demand and ensure system efficiency, operators must continuously redistribute the vehicles in a complex maintenance task known as rebalancing [5].

This thesis presents a data-driven, dynamic rebalancing framework for dockless BSS. The goal is to design an intelligent, adaptive decision-making strategy that mitigates vehicle shortages, optimizes battery charging, and guarantees equitable service availability across complex urban road networks. The contributions of this work encompass the entire process of modeling and solving the fleet management problem. First, the urban road network and stochastic user demand are formalized as a Markov Decision Process (MDP) defined on a discrete-cell graph, which incorporates continuous-time demand dynamics, realistic travel speeds, and battery consumption models. Building on this formulation, it develops a Multi-Agent Reinforcement Learning (MARL) approach that combines Proximal Policy Optimization (PPO) with Graph Attention Network (GAT), enabling autonomous rebalancing agents to process the structural characteristics of the road network and identify critical demand areas in real time. Then, it establishes a generalized multi-area framework that leverages parameter sharing to train a single, area-independent policy across geographically distinct regions, achieving scalability without increasing network complexity. Finally, the proposed architecture is empirically validated using historical Citi Bike data from Manhattan, New York, to assess its effectiveness. The experimental results confirm that the learned rebalancing policy effectively mitigates service failures and ensures a more uniform distribution of bikes across the network compared to the baseline models examined, demonstrating the feasibility of the proposed framework.

Contributions

With respect to the existing literature on BSS rebalancing, and in particular to the system formulation this work builds upon [6, 7], the contributions of this thesis can be summarized as follows:

1. A unified MDP formulation for rebalancing dockless BSS, which jointly models stochastic user demand in continuous time, realistic travel speed distributions, and battery depletion dynamics on a discrete-cell graphical representation of the urban road network, extending the system model in [7] with a multi-area formulation.
2. An extension of O’Mahony’s classic static rebalancing scheme [8] into an hourly, demand-adaptive baseline, used along with a free scenario (without interventions) to provide a comparison between the absence of rebalancing, periodic static redistribution, and the proposed dynamic framework. The devised solution leverages a novel MARL approach based on PPO with GAT, which includes a continuously adaptive on-policy actor-critic scheme.
3. An empirical validation on real-world Citi Bike trip data from Manhattan, New York, quantifying the benefits of the proposed dynamic approach: up to a 90.5% reduction in service failures with respect to the free scenario and a further 54.4% reduction with respect to the static baseline, depending on the study area.

Thesis Outline

The remainder of this thesis is organized as follows:

- Chapter 2: **Theoretical Background** introduces the mathematical foundations of Reinforcement Learning (RL), MDPs, Deep Reinforcement Learning (DRL), PPO, and MARL training paradigms such as Centralized Training Decentralized Execution (CTDE).
- Chapter 3: **State of the Art** reviews literature on static and dynamic BSS rebalancing, data-driven optimization in micro-mobility, spatial fairness, and multi-agent fleet management techniques.
- Chapter 4: **System Model** formalizes the graph representation of the urban BSS, user demand modeling via Continuous-Time Markov Chain (CTMC), fleet battery dynamics, and the multi-area environment extension.
- Chapter 5: **Proposed Framework** details the data preprocessing pipeline using Manhattan Citi Bike records, the discrete MDP formulation (state space, action set, and reward function), the Graph Attention Network v2 (GATv2)-based PPO network architecture, and the multi-area parameter sharing mechanism.

- Chapter 6: **Simulations & Results** describes the experimental setup across two distinct subareas in Manhattan, compares the proposed dynamic strategy against static rebalancing baselines, and analyzes performance metrics and policy decision behavior.
- Chapter 7: **Conclusions and Future Work** summarizes the main findings and contributions of this thesis and highlights potential directions for future work.

2

Theoretical Background

This chapter presents the theoretical basis required to understand the methods discussed in the following chapters. It first introduces the basic concepts of Reinforcement Learning (RL) and Markov Decision Process (MDP). Tabular Q-learning and artificial neural networks are then discussed as the conceptual foundations of Deep Reinforcement Learning (DRL). The main paradigms of the DRL are subsequently presented, with particular emphasis on Proximal Policy Optimization (PPO), the central algorithm used in this work. Finally, the discussion is extended to the multi-agent setting, focusing on its formalization, and reference training architectures.

2.1 Basics of Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning dedicated to studying the behavior of an autonomous agent with its environment to achieve the expected cumulative return. Unlike supervised learning, where the model uses a predetermined set of labeled instances to learn, or unsupervised learning, which seeks patterns in unlabeled data, RL is interactive by nature, combining learning and decision-making. The following concepts are discussed in detail in [9].

RL is based on the repeated interaction between an autonomous agent and an environment. At each time step t , the agent observes information about the current condition of the environment, chooses an action A_t according to its policy, and receives feedback from the resulting transition. This feedback consists of a scalar reward R_{t+1} and a new state S_{t+1} , which provide the basis for subsequent decisions. The agent learns through trial and error, analyzing the results of its actions, and adjusting its behavior accordingly.

The interaction between the agent and the environment are formally described by

a Markov Decision Process (MDP). An MDP is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where:

- $\mathcal{S}$ represents the set of states (the state space).
- $\mathcal{A}$ represents the set of possible actions available to the agent (the action space).
- $\mathcal{P}_{ss'}^a = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$ is the state transition probability matrix, defining the transition probability from state s to state s' due to action a .
- $\mathcal{R}_s^a = \mathbb{E}(R_{t+1} \mid S_t = s, A_t = a)$ specifies the expected immediate reward received after taking action a in state s .
- $\gamma \in [0, 1)$ is the discount factor, which determines the present value of future rewards.

The core RL interaction can therefore be depicted as shown in Figure 2.1.

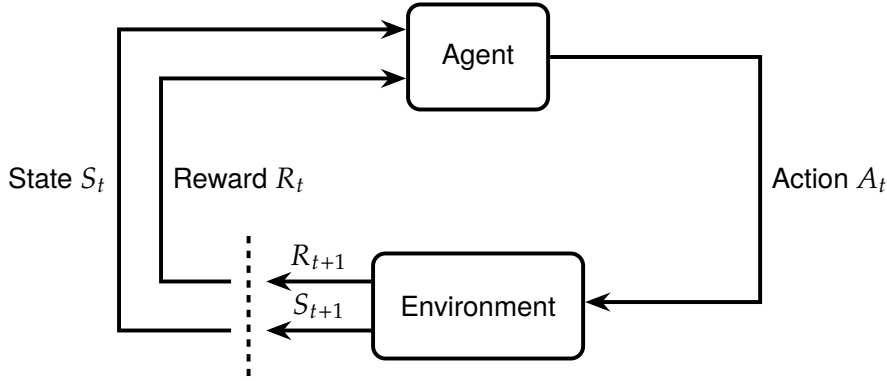


Figure 2.1: The agent-environment interaction in a Markov Decision Process.

The Markov property is a fundamental assumption associated with MDPs. A stochastic process is said to exhibit Markov property if its conditional probability distribution for future states depends only on the current state and action, being independent of past states and actions. This means that the current state contains all relevant past information necessary to describe the conditional distribution of future transitions [9].

The main objective of the RL agent is the maximization of long-term reward, rather than immediate reward. This idea is formalized in terms of the return G_t , which denotes the discounted sum of rewards collected from time step t onward. For continuing tasks, the discount factor γ helps to ensure that the infinite sum of future rewards remains bounded under standard assumptions on the reward signal. Discounted return is given as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \begin{cases} \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, & \gamma \in (0, 1) \\ R_{t+1}, & \gamma = 0 \end{cases}. \quad (2.1)$$

The parameter γ controls the weight of future rewards: if $\gamma = 0$, then the agent is

myopic and only considers immediate reward R_{t+1} ; as $\gamma \rightarrow 1$ the agent becomes more future-oriented and assigns greater importance to future rewards.

To evaluate the desirability of states and actions, RL makes use of value functions. The agent's behavior is governed by a policy $\pi(a | s)$, which defines the probability of selecting action a given state s . The state value function $V^\pi(s)$ computes the expected return starting from state s under policy π :

$$V^\pi(s) = \mathbb{E}_\pi [G_t | S_t = s] . \quad (2.2)$$

Likewise, the action value function $Q^\pi(s, a)$ determines the expected return of taking action a in state s according to policy π :

$$Q^\pi(s, a) = \mathbb{E}_\pi [G_t | S_t = s, A_t = a] . \quad (2.3)$$

A key property of value functions is their recursive nature, meaning they satisfy the so-called Bellman equations. The Bellman expectation equation expresses the value of a state as the expected immediate reward plus the discounted value of the subsequent state:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a | s) \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a [\mathcal{R}_s^a + \gamma V^\pi(s')] . \quad (2.4)$$

In RL, the ultimate goal is to determine an optimal policy π^* , which yields the highest expected return for each state and leads to optimal value functions: $V^*(s) = \max_\pi V^\pi(s)$ and $Q^*(s, a) = \max_\pi Q^\pi(s, a)$. The following subsections present standard techniques for implementing these maximization procedures in practice.

2.1.1 Tabular Q-learning

Q-learning is a model-free RL technique, meaning it does not require a prior model of the environment's transition dynamics. The objective is to approximate the optimal action-value function $Q^*(s, a)$, which gives the maximum expected return starting at state s and following the optimal policy after taking action a . The value estimation is updated incrementally after each interaction with the environment, represented by the transition tuple $(S_t, A_t, R_{t+1}, S_{t+1})$:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)] , \quad (2.5)$$

where α is the learning rate, γ is the discount factor, and $R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')$ is the update target, also known as the Temporal Difference (TD) target.

The Q-function in the tabular format is expressed as a table with one entry per state-action pair. Watkins and Dayan [10] showed that Q-learning converges to the optimal action-value function provided that all state-action pairs are continually updated (adequate exploration) and the learning rate meets specific stochastic approximation conditions. However, this tabular approach faces severe limitations when applied to continuous state spaces or high-dimensional discrete environments. In the last scenario, the size of the Q-table grows exponentially with the number of states and actions, leading to the *curse of dimensionality*¹. This limitation motivated the development of Deep Reinforcement Learning (DRL), and specifically, deep Q-learning, where the Q function is approximated by a neural network.

2.1.2 Neural Networks

An Artificial Neural Network (ANN) is a network of interconnected units, called neurons, modeled on the working principles of a biological nervous system [9, 12]. The parameters of a neural network are typically learned by minimizing a loss function. The backpropagation algorithm computes the gradient of the loss with respect to the network parameters, which are then updated using a gradient-based optimization method [13]. Depending on their connectivity pattern, ANNs can be broadly classified as feed-forward networks, which contain no recurrent connections, and recurrent networks, which include feedback connections. A feedforward ANN, as illustrated in Figure 2.2, receives a feature vector x as input, processes it through one or more hidden layers h by applying weights w_{ij} , biases, and nonlinear activation functions, and produces an output y .

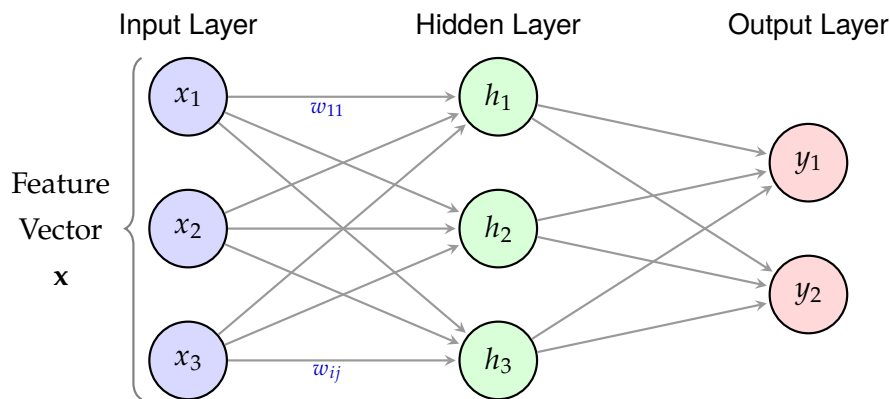


Figure 2.2: Feedforward ANN [9].

¹The curse of dimensionality refers to the rapid growth in computational and memory requirements caused by an increasing number of state and action variables [11].

The universal approximation theorem by Hornik et al. [14] is an important result from neural network theory which proves that a feedforward neural network with a single hidden layer and a sufficient number of neurons can approximate any continuous function on compact sets with arbitrary precision, if the activation function satisfies certain conditions (i.e., suitable regularity and non-degeneracy conditions). This result provides a theoretical motivation for using neural networks as function approximators for value functions and policies in DRL.

2.2 Deep Reinforcement Learning

Although classical RL establishes a solid theoretical basis using MDPs, traditional tabular approaches, such as conventional Q-learning, are limited when applied to complex, realistic applications. Tabular methods involve explicitly storing state values or state-action values in memory. However, as the number of state and action variables increases, the number of discrete state-action combinations can grow exponentially. In continuous or high-dimensional spaces, explicit tabular representations become impractical or impossible [9, 15]. To address these limitations, DRL combines RL with deep neural networks, which are used as flexible function approximators for policies, value functions, and action-value functions [16]. By using parameters θ and ϕ to approximate policies $\pi_\theta(a|s)$, state values $V_\phi(s)$, or action values $Q_\phi(s, a)$, the learning process can generalize across states without requiring a visit to every possible state [17].

Based on how they model the decision process, DRL algorithms generally fall into three main paradigms:

- Value-based approaches: Aim to approximate the optimal action-value function to implicitly find the optimal policy.
- Policy-based approaches: Aim to directly optimize the policy distribution, without relying on learned value functions.
- Actor-Critic approaches: Combine policy optimization with value estimation. The actor selects actions according to the current policy, while the critic evaluates them using a value function and provides feedback to improve the actor.

Value-based Approaches

In value-based DRL algorithms, the optimal action-value function $Q^*(s, a)$ is approximated with a deep neural network $Q_\phi(s, a)$. The agent updates the network parameters by implicitly deriving the policy from the greedy action $a^* = \arg \max_{a \in \mathcal{A}} Q_\phi(s, a)$.

The most widely used value-based algorithm is Deep Q-Network (DQN), which introduced two techniques to stabilize learning in complex environments: experience replay to reduce temporal correlations between consecutive transitions, and *target networks* to stabilize the bootstrapped targets² [18].

The main strength of value-based approaches is their high efficiency in terms of data use, since transitions from the past saved in the replay memory can be reused multiple times for updating network parameters. However, standard value-based methods are naturally suited to discrete action spaces, since selecting an action requires evaluating or maximizing $Q(s, a)$ over the available actions. In addition, some value-based methods can suffer from overestimation bias and optimization instabilities [19].

Policy-based Approaches

Instead of estimating the value of an action, policy-based methods directly model the policy itself as a probability distribution $\pi_\theta(a | s)$ over actions, parameterized by a set of weights θ . The aim is to optimize the objective function $J(\theta)$ by updating θ in the direction of the performance gradient $\nabla_\theta J(\theta)$ using gradient ascent [9]:

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t) . \quad (2.6)$$

Policy gradient methods operate by computing an estimator of the policy gradient and plugging it into a stochastic gradient ascent algorithm [20]. Based on the policy-gradient theorem, the gradient can be estimated without explicitly knowing the transition dynamics of the environment [21]:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(A_t | S_t) G_t \right] , \quad (2.7)$$

where G_t is a sample of the discounted return from time step t onward. The score function $\nabla_\theta \log \pi_\theta(A_t | S_t)$ determines how the policy parameters should be modified, while the return weights the update according to the observed quality of the sampled action.

Since the policy is modeled as a probability distribution over actions, the trade-off between exploration and exploitation is determined directly by its shape: sampling from a nearly uniform distribution promotes exploration, while a highly peaked distribution concentrates the probability mass on few actions, corresponding to exploitation.

²Bootstrapping refers to the procedure of updating estimates of the values of a state using estimates of the values of subsequent states, rather than waiting until the end of the episode [9].

This trade-off can be quantified through the Shannon entropy [22] of $\pi_\theta(\cdot | S_t)$:

$$\mathfrak{S}_{\pi_\theta}(S_t) = - \sum_{a \in \mathcal{A}} \pi_\theta(a | S_t) \log(\pi_\theta(a | S_t)) \quad (2.8)$$

which attains its minimum value of 0 for a deterministic policy and its maximum value of $\log |\mathcal{A}|$ for a uniform distribution over the $|\mathcal{A}|$ actions. Maximizing this term encourages policy toward a more uniform distribution of actions, promoting exploration of the action space.

The main advantage of policy-based approaches lies in their straightforward application to continuous action spaces, achieved by predicting the parameters of continuous probability distributions (such as the mean and variance of Gaussian distributions). On the other hand, dependence on full trajectories introduces high gradient variance, making convergence difficult without expensive interactions with the environment. Moreover, because trajectories depend on the current policy parameters, old trajectories cannot be reused after an update, leading to poor sample efficiency.

Actor-Critic Approaches

Actor-Critic models combine the strengths of value and policy-based methods into a single architecture, aimed at lowering variance while remaining versatile in both discrete and continuous spaces [23]. This architecture is divided into two distinct modules:

- The Actor: Encodes the policy $\pi_\theta(a | s)$ and focuses on making decisions and exploring.
- The Critic: Encodes a value function $V_\phi(s)$ to assess the quality of the actor's decisions by estimating the expected future return from state s .

To reduce variance during the policy update, the Actor-Critic approach utilizes the advantage function, $A^{\pi_\theta}(s, a)$, which measures the relative advantage of action a over the expected value of state s under the current policy [24]:

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s). \quad (2.9)$$

The advantage function is subsequently used as a weighting term in the policy-gradient estimator:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(A_t | S_t) A^{\pi_\theta}(S_t, A_t)] . \quad (2.10)$$

The key benefit of the Actor-Critic architecture is its capability to minimize gradient variance through the critic's baseline evaluation, while remaining versatile in both

discrete and continuous spaces. However, a major challenge in training such models is the simultaneous optimization of two interdependent neural networks, in which inaccurate value estimates from the critic potentially affect the actor’s policy update.

This Actor-Critic formulation forms the backbone of modern policy optimization methods. To prevent drops in performance caused by uncontrolled policy updates, more sophisticated algorithms, such as PPO, build on the actor-critic formulation and control the size of policy updates through a clipped surrogate objective [20]. Furthermore, when scaling these concepts to environments where multiple decision-makers interact simultaneously, this single-agent approach must be adapted to handle non-stationary settings, leading to Multi-Agent Reinforcement Learning (MARL).

2.3 Proximal Policy Optimization

The PPO framework is a family of RL algorithms based on policy gradient optimization techniques that combine the advantages of Trust Region Policy Optimization (TRPO) approach with the computational simplicity of first-order stochastic optimization methods. The main concepts related to the PPO framework described in this section are based on Schulman et al. [20].

In traditional policy gradient algorithms, an excessively aggressive update to policy parameters may result in a collapse of performance, as even small errors in the estimation can lead to policies that are difficult to recover from. TRPO addresses this issue by enforcing a Kullback-Leibler (KL) divergence constraint between the old policy $\pi_{\theta_{\text{old}}}$ and the new policy π_{θ} [20, 25]:

$$\max_{\theta} \mathbb{E}_t \left[\frac{\pi_{\theta}(A_t | S_t)}{\pi_{\theta_{\text{old}}}(A_t | S_t)} \hat{A}_t^{\text{adv}} \right] \quad \text{subject to} \quad \mathbb{E}_t [\text{KL}(\pi_{\theta_{\text{old}}}(\cdot | S_t), \pi_{\theta}(\cdot | S_t))] \leq \delta, \quad (2.11)$$

where $\hat{A}_t^{\text{adv}}$ represents the estimated advantage at time step t . Despite its theoretical validity, TRPO requires approximating second-order information through Hessian-vector products and solving a constrained optimization subproblem, which makes it more complex to implement than first-order methods. PPO was introduced to overcome these limitations.

As an Actor-Critic algorithm, PPO offers several variants, the most widely adopted being PPO-Clip. PPO-Clip limits the incentive to change the probability of an action too aggressively by clipping the ratio between the new and old policies. It introduces a novel surrogate objective, which penalizes large deviations from the old policy without requiring the KL divergence constraint used in TRPO [26]. Let $r_t(\theta)$ be the probability

ratio between the new and old policies:

$$r_t(\theta) = \frac{\pi_\theta(A_t | S_t)}{\pi_{\theta_{\text{old}}}(A_t | S_t)}. \quad (2.12)$$

The surrogate objective function optimized by PPO-Clip is defined as:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t^{\text{adv}}, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t^{\text{adv}})], \quad (2.13)$$

where ϵ is a hyperparameter that defines the clipping interval (typically set to 0.2), and the clip operator restricts the value of the ratio to the interval $[1-\epsilon, 1+\epsilon]$. The Equation (2.13) is used as a pessimistic lower bound on the unconstrained objective:

- If $\hat{A}_t^{\text{adv}} > 0$, the action performed better than average; however if $r_t(\theta) > 1 + \epsilon$, the objective is clipped to prevent over-optimistically increasing the probability of that action.
- If $\hat{A}_t^{\text{adv}} < 0$, the action performed worse than average; decreasing the probability of the action improves the objective, but, if $r_t(\theta) < 1 - \epsilon$, clipping removes the incentive to reduce that probability further [20].

Accurately estimating the advantage $\hat{A}_t^{\text{adv}}$ is crucial for balancing the bias-variance trade-off in policy gradients. PPO accomplishes this using Generalized Advantage Estimation (GAE) [24, 26], which parametrizes a set of advantage estimates using a single decay parameter $\lambda \in [0, 1]$. First, the TD error δ_t at time step t is defined as:

$$\delta_t = R_{t+1} + \gamma V_\phi(S_{t+1}) - V_\phi(S_t), \quad (2.14)$$

where R_{t+1} is the immediate reward, γ is the discount factor, and $V_\phi(S_t)$ is the current state-value estimate. The GAE estimator is defined as the weighted sum of future TD errors [24]. In practical implementations with a finite horizon of length T , it is defined as:

$$\hat{A}_t^{\text{adv}} = \begin{cases} \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}, & \gamma \in (0, 1) \\ \delta_t, & \gamma = 0 \end{cases}. \quad (2.15)$$

When the neural network architecture shares parameters between the policy network (actor) and the value function network (critic), a unified global loss function is minimized. This combined loss consists of three components: the clipped surrogate objective function L_t^{CLIP} , the value function estimation error L_t^{VF} , and an additional entropy term S to encourage exploration:

$$L_t^{\text{CLIP+VF+S}}(\theta, \phi) = \mathbb{E}_t[-L_t^{\text{CLIP}}(\theta) + c_1 L_t^{\text{VF}}(\phi) - c_2 \mathfrak{S}_{\pi_\theta}(S_t)], \quad (2.16)$$

where c_1 and c_2 are scalar coefficients. The entropy term $\mathfrak{S}_{\pi_\theta}(S_t)$ measures the uncertainty or diversity in the action probability distribution. Including entropy in the loss function prevents the policy from converging prematurely to deterministic choices, encouraging continued exploration of the action space and avoiding policy collapse [20, 26]. The term $L_t^{VF}(\phi)$ is the mean squared-error between the value predicted by the critic network $V_\phi(S_t)$ and the target value V_t^{target} : $L_t^{VF}(\phi) = (V_\phi(S_t) - V_t^{\text{target}})^2$. This helps the critic correctly evaluate the state to have an adequate baseline for advantage computation.

Training a policy with PPO is an iterative process that alternates between two distinct phases, as illustrated in Algorithm 1:

1. Sampling phase: The agent interacts with the environment using the old policy $\pi_{\theta_{\text{old}}}$ to collect trajectories of states, actions, and rewards over a fixed time horizon.
2. Optimization phase: Advantage estimates and value targets are computed from the collected trajectories, and the surrogate objective L^{CLIP} is optimized for multiple mini-batch epochs using stochastic gradient ascent.

Reusing the collected data for multiple optimization epochs can improve sample efficiency relative to methods that perform only one update per batch.

Algorithm 1 PPO [20, 26]

```

Initialize policy parameters  $\theta$  and set  $\theta_{\text{old}} \leftarrow \theta$ 
for iteration = 1, 2, ... do
  for actor = 1, 2, ...,  $N_A$  do
    Run policy  $\pi_{\theta_{\text{old}}}$  in environment for  $T$  time steps
    Compute advantage estimates  $\hat{A}_1^{\text{adv}}, \dots, \hat{A}_T^{\text{adv}}$  using GAE
  end for
  Optimize surrogate objective  $L^{\text{CLIP}}$  with respect to  $\theta$ , for  $K$  epochs and mini-batch
  size  $M \leq N_A T$ 
  Update old policy parameters:  $\theta_{\text{old}} \leftarrow \theta$ 
end for

```

2.4 Multi-Agent Reinforcement Learning

In RL, a multi-agent system is composed of multiple interacting agents, which make decisions autonomously, within a shared environment [15]. The generalization of the MDPs to the multi-agent case is the Markov game [27]. A Markov game with N agents is defined as a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}^i, \mathcal{P}, \mathcal{R}^i, \gamma \rangle$, where $\mathcal{N} = 1, \dots, N$ denotes the set of $N > 1$ interacting agents, $\mathcal{S}$ is the set of observed states, $\mathcal{A} = \mathcal{A}^1 \times \dots \times \mathcal{A}^N$ is the joint action space of all agents, $\mathcal{P}$ is the transition probability function, $\mathcal{R}^i$ is the reward function

for agent i , and γ is the discount factor. Unlike the single-agent case, the transition between states is determined by the joint action of all agents, and each agent's return also depends on it. In a partially observable scenario, each agent i receives only a private observation of its environment that does not allow it to identify the state with certainty [11, 15]. Its policy π_i is therefore conditioned on its local action-observation history τ_i rather than on the global state [28]. This extension is the reference framework for most modern applications of MARL.

Depending on the nature of the interaction between agents, MARL problems are classified into three categories:

- Fully cooperative: All agents receive the same reward $R = R^1 = \dots = R^N$ for state transitions and work toward a common goal.
- Fully competitive: The agents have conflicting objectives. In a zero-sum game, the sum of their rewards is zero for every state transition.
- Mixed: Agents pursue individual goals in situations involving both cooperation and competition.

MARL training schemes can be classified according to how information is shared during training and how actions are selected during execution, as shown in Figure 2.3. In Distributed Training Decentralized Execution (DTDE), each agent learns its own policy independently, without sharing information with others. In Centralized Training Centralized Execution (CTCE), the joint policy of all agents is modeled as a single centralized system that maps the set of distributed observations to individual actions. In Centralized Training Decentralized Execution (CTDE), each agent has an individual policy that maps local observations to a distribution of individual actions. Furthermore, in CTDE, during training, agents are provided with additional information, which is then discarded during execution.

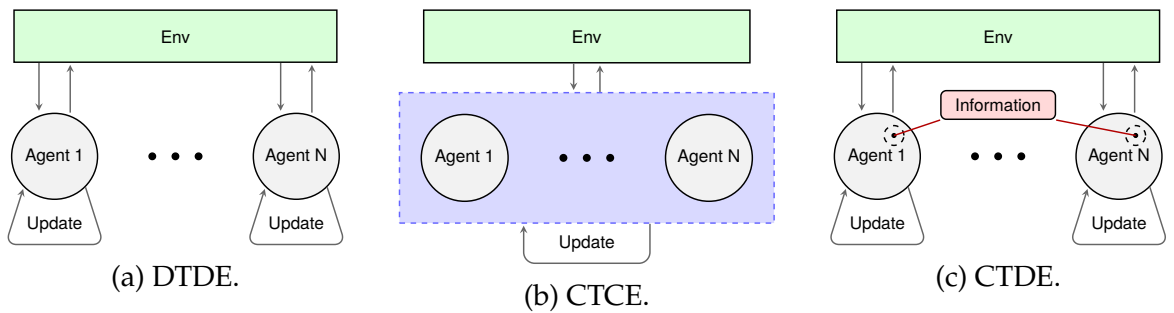


Figure 2.3: Training schemes in the multi-agent setting [27].

2.4.1 Multi-Agent Actor-Critic Framework

In multi-agent settings, the challenge of non-stationarity arises: since the policies of all agents are updated simultaneously during training, the environment's dynamics appear non-stationary from the perspective of any individual agent. To address this issue, modern multi-agent algorithms widely adopt the CTDE paradigm. As pioneered by Lowe et al. [29] in the context of Actor-Critic architectures, CTDE bridges the gap between single-agent stability and multi-agent deployment. In a CTDE Actor-Critic framework, each agent i has an individual decentralized actor π_i , parametrized by θ_i , which takes actions A_i purely based on local observations o_i . In addition, during training, each agent has a centralized critic Q_i^π which takes as input the observations and actions of all agents, as illustrated in Figure 2.4. The gradient of the policy for each agent i is expressed as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E} \left[\nabla_{\theta_i} \log \pi_i(A_i | o_i) Q_i^\pi(\mathbf{x}, A_1, \dots, A_N) \right], \quad (2.17)$$

where $\mathbf{x}$ represents the state information, and Q_i^π is the centralized action-value function of agent i . By conditioning the critic on the global state and joint actions, centralized training provides a stable baseline that successfully resolves the non-stationarity issue. The primary advantage of this paradigm is its flexibility, as it can be applied to cooperative, competitive, and mixed scenarios.

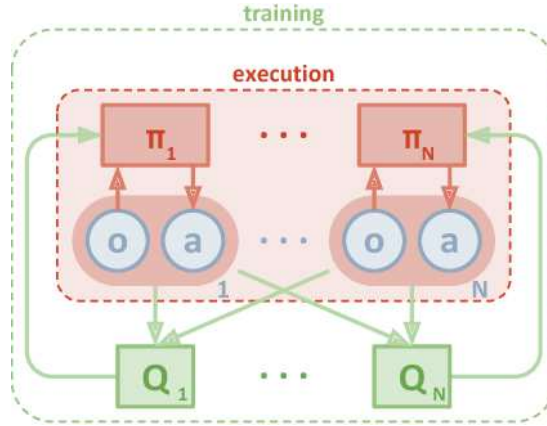


Figure 2.4: Overview of multi-agent decentralized actor, centralized critic approach [29].

2.4.2 Multi-Agent PPO

The centralized-critic Actor-Critic framework can also be combined with PPO, resulting in the Multi-Agent Proximal Policy Optimization (MAPPO) algorithm [30]. MAPPO extends the standard PPO framework to cooperative multi-agent environments by following the CTDE paradigm. The algorithm learns a policy π_θ and a value function V_ϕ , which are represented by two separate neural networks. During execution, the policy of each agent i receives only its local observation o_{t_i} and selects an action according to $\pi_\theta(A_{t_i} | o_{t_i})$. Therefore, each agent can act independently without any knowledge about the observations or actions of the other agents. During training, instead, the centralized critic can use additional information, such as the global state S_t , to estimate the state-value function, rather than a joint action-value function. This is used as a centralized baseline for evaluating the advantage estimates of the agents, and enables the algorithm to efficiently address the problem of non-stationarity, while using the PPO clipping technique to limit excessively large policy updates. Since the critic is only used during training, the availability of centralized information does not compromise decentralized execution.

The main difference between standard PPO and MAPPO concerns the information provided to the value function. The actor remains conditioned on local observations, while the critic can exploit centralized information to produce more informative value estimates and advantage functions.

For each agent i , let $r_{t_i}(\theta)$ be the probability ratio between the new and the old policies:

$$r_{t_i}(\theta) = \frac{\pi_\theta(A_{t_i} | o_{t_i})}{\pi_{\theta_{\text{old}}}(A_{t_i} | o_{t_i})}. \quad (2.18)$$

The clipped surrogate objective is defined as:

$$L_{MAPPO}^{CLIP}(\theta) = \mathbb{E}_{t,i}[\min(r_{t_i}(\theta)\hat{A}_{t_i}^{\text{adv}}, \text{clip}(r_{t_i}(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_{t_i}^{\text{adv}})], \quad (2.19)$$

where $\hat{A}_{t_i}^{\text{adv}}$ is the advantage estimate of agent i . The critic is trained by minimizing the value function loss:

$$L_{MAPPO}^{VF}(\phi) = \mathbb{E}_t[(V_\phi(S_t) - \hat{R}_t)^2], \quad (2.20)$$

where $\hat{R}_t$ denotes the discounted reward-to-go.

The overall training procedure of MAPPO is summarized in Algorithm 2.

Algorithm 2 MAPPO [30, 31]

Initialize actor parameters θ
Initialize centralized critic parameters ϕ
for iteration $t = 1, \dots, T$ **do**
 for agent $i = 1, \dots, N$ **do**
 Observe the local observation o_{t_i}
 Sample an action $A_{t_i} \sim \pi_\theta(\cdot \mid o_{t_i})$
 Estimate the state value $V_\phi(S_t)$ using centralized information
 end for
 Execute the joint action $\mathbf{A}_t = (A_t^1, \dots, A_t^N)$
end for
Compute the advantage estimates $\hat{A}_{t_i}^{\text{adv}}$ using GAE
Optimize surrogate objective $L_{\text{MAPPO}}^{\text{CLIP}}$ with respect to θ and minimize the value function loss $L_{\text{MAPPO}}^{\text{VF}}(\phi)$ with respect to ϕ
Update parameters θ and ϕ

3

State of the Art

This chapter discusses the implementation of data-driven optimization methods in the urban micro-mobility sector. It begins with an introduction to the problem of rebalancing in Bike-Sharing Systems (BSSs), then moves on to the application of RL methods to solve it, and concludes with MARL models, developed to address issues related to scalability and spatiotemporal complexity.

3.1 Rebalancing in Bike-Sharing Systems

BSSs, especially the dockless ones, exhibit pronounced spatial and geographical disparities in user demand. This directionality imbalance manifests as a large influx of bicycles into central areas during the morning commute, followed by an equally pronounced exodus in the afternoon. Residential areas experience the opposite pattern [5]. This disparity in the demand requires that the vehicles be continually repositioned (rebalancing) for redistribution of the fleet throughout the territory.

There are two types of rebalancing operations. Static rebalancing, also known as Static Repositioning Problem (SRP), is executed under the assumption of complete knowledge of the system's state when there is no demand from the users. The entire process, conducted mainly at night, aims to restore the system to its optimal configuration. Specifically, the primary goal is to get the stations ready to absorb the impact of the expected demand at the beginning of the day or of a certain time period that will be referred to as the "coverage duration" preceding the launch of dynamic processes [32, 33]. On the other hand, the Dynamic Repositioning Problem (DRP) is carried out simultaneously with the users' utilization of the service. Such a mode of operation necessitates flexible planning taking into account the continuous changes that occur due to the active renting [33, 34].

Early work on bike-sharing rebalancing mainly focused on formulating static repositioning as a vehicle-routing and inventory-management problem. Chemla et al. [32] defined the problem as finding a minimum-cost route that restores the inventory level of each station through pickup and delivery operations. During the same period, O'Mahony and Shmoys [8] investigated the rebalancing problem using operational data from the Citi Bike system in New York City. Their study considered both overnight rebalancing, aimed at preparing the system for the following day, and rebalancing during rush-hour periods. They combined data analysis with routing models and clustering techniques to identify suitable bike locations and repositioning operations.

Another line of research aimed to reduce the reliance on dedicated rebalancing vehicles by involving users in the redistribution process. Singla et al. [35] proposed an incentive-based method in which users were offered alternative pickup and drop-off locations in exchange for monetary rewards. Subsequent studies focused on integrating inventory decisions with vehicle-routing decisions. Schuijbroek et al. [36] showed that the rebalancing plan and the routes used to transport bikes between stations must be considered jointly, since inventory decisions directly affect the feasibility and cost of the resulting vehicle routes.

In 2018, Chiariotti et al. [37] proposed a dynamic rebalancing approach based on historical demand data. Their method used past demand observations to forecast the state of the network and determine when rebalancing should be performed. The experiments showed that a dynamic strategy capable of responding to demand fluctuations can outperform strategies based on a fixed static schedule. The increasing availability of trip and station data further encouraged the development of data-driven methods for demand prediction and rebalancing decision-making. Pan et al. [38] proposed a DRL framework for rebalancing dockless BSSs through user incentives. Their approach used spatial and temporal information to determine suitable incentive policies for users and to improve the distribution of bikes across the system. At a broader level, Albuquerque et al. [39] systematically reviewed machine learning applications in BSSs. Their review identified demand prediction and classification as the main application areas of machine learning in this field. These techniques can support rebalancing operations by extracting spatial and temporal demand patterns and by providing more accurate information for fleet redistribution.

Beyond these operational concerns, BSSs also raise socio-spatial equity issues. The bikes tend to be found only in high-demand central areas, leaving the suburbs and peripheral areas untouched. There is a strong link between this situation and the existence of a deep socioeconomic contrast, since the tendency is for the system to concentrate in wealthy areas, thereby discriminating against those groups historically

deprived or poorer communities [5]. The risk is to trigger a process of “fragmented urbanism” if there are no public policies aimed at addressing such a problem, meaning that while only the privileged class enjoys the use of technology-enabled new forms of mobility, the poorest communities cannot benefit from it [4]. These considerations motivate the increasing attention to spatial fairness in recent optimization and learning-based rebalancing approaches, discussed in the following sections.

3.2 Reinforcement Learning in Shared Mobility

RL has gained popularity as an efficient technique within intelligent transportation systems for dealing with complex and stochastic decision-making tasks [40]. An application of RL can be observed in the context of Vehicle Routing Problem (VRP). This is an optimization problem related to logistics and smart city management that aims to optimize the routes of delivery vehicles with limited capacity [41]. While classical Operations Research (OR) approaches, such as mixed-integer linear programming, face the problem of dimensionality since they are based on assumptions regarding a static system [42], RL is a data-driven approach capable of adapting to new operational environments. Through the representation of the transportation network as a MDP, an agent can learn optimal policies via continuous interaction with the environment [40]. In this framework, RL acts as an approximate Dynamic Programming (DP) method, enabling optimal sequential decision-making without suffering from the *curse of dimensionality* that limits exact DP in large-scale mobility systems [43]. This enables the processing of heterogeneous, time-varying inputs (such as demand fluctuations or traffic flows), thereby shifting control strategies from a purely reactive paradigm to a proactive one [6].

RL has found numerous applications in addressing DRP problems within BSSs. Earlier works have proposed tabular Q-learning and actor-critic models to balance the fleet of bikes in small-scale networks while minimizing lost demand [44]. Nevertheless, due to the growing spatial and temporal scales of BSSs, research has progressively shifted toward DRL to handle extremely high-dimensional state and action spaces [45]. Furthermore, recent studies have integrated DRL with neural networks, which take into account spatiotemporal dependencies of urban transportation network [38, 46].

Recent literature has expanded beyond traditional optimization frameworks to incorporate areas such as temporal accuracy, spatial fairness, and continuous intervention. Liang et al. [47] formulated the DRP by moving away from the conventional approach of dividing time into fixed intervals. By utilizing a continuous-time DRL model, they were able to incorporate real-time, asynchronous decision-making to miti-

gate demand loss. The concept of algorithmic fairness in shared systems was proposed by Cederle et al. [5], whose methodology demonstrated how the learning-based repositioning can account for equity issues, offering a balanced service without discrimination against outliers or underprivileged areas. Additionally, to address the structural problem associated with traditional techniques, i.e., the need for global relocation procedures, Scarpel et al. [6] proposed a technique combining Double Deep Q-Network (DDQN) and GATs. By doing so, this framework allows for network optimization via localized loading, unloading, and charging of a single truck [6]. Based on the spatiotemporal criticality score that categorizes the regions of the city as critical, stable, or surplus, such a technique manages to provide an equitable service and avoid mobility deserts [6].

3.3 Fleet Management through Multi-Agent RL

Applying RL to BSSs rebalancing presents unique challenges which make the use of multi-agent model more effective than the single agent model. In BSSs, the number of repositioning agents, status of stations, and action space keep changing dynamically over time. A single agent faces the problem of exponential growth in action space, since it has to take into account positions of all agents, number of bikes on each station, and actions in the whole network [40, 42]. A shift to a multi-agent framework, where each individual repositioning vehicle is considered an autonomous agent, results in a reduced action space dimensionality and increased learning stability [40]. Independent MARL, on the other hand, suffers from a drawback called non-stationarity; in this scenario, since all the agents are learning together and affecting the environment at the same time, the environment is not stable anymore, and the agents are unable to converge to any good policies [42]. To overcome this limitation, cooperative methods utilize the paradigm of CTDE, where the environment provides all agents with global state/action information in a centralized critic while training, and each agent operates independently using its own local observation while executing [42]. In this context, one notable innovation is the use of Mean Field Reinforcement Learning (MFRL). This technique uses the interaction between an agent and the mean impact of other agents to effectively cope with the dynamically changing number of agents as well as the problem of dimensionality of the state space [42].

Furthermore, Li et al. [48] demonstrate that MARL outperforms conventional repositioning approaches, achieving 76% to 133% higher user demand satisfaction than long-term repositioning and VRP baselines while maintaining system efficiency across varying fleet sizes and transport capacities. This shift is not merely a matter of scal-

ability; it enables the modeling of spatiotemporal system dynamics, the management of real-time operational uncertainties, and the attainment of an optimal trade-off between demand satisfaction and operational costs, goals that neither centralized nor independent approaches can achieve.

However, despite all these encouraging trends, current approaches to MARL continue to exhibit several shortcomings in managing asynchronous decision-making, dynamically varying numbers of agents, and spatial fairness at the same time [42]. The solution to the mentioned challenges constitutes the key open problem of modern shared mobility rebalancing studies.

4

System Model

This chapter formalizes the framework and model underlying the dokless BSS. The urban infrastructure is first abstracted as a graph-based topology, which defines the system states and the local actions of the rebalancing truck. User demand is then modeled as a Continuous-Time Markov Chain (CTMC), together with a model of fleet battery consumption driven by realistic travel speeds. Finally, the framework is extended to a multi-area setting, yielding a generalized, area-independent architecture that supports a single, scalable decision-making policy across different urban regions. The model follows the formulation introduced in [7] and further developed in [6]. For reference, the symbols introduced in this chapter are summarized in Table A.1 in Appendix A.

4.1 System Representation

The urban area served by the BSS is represented as an undirected, weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes, each corresponding to a cell of the urban area, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges representing the road segments connecting them. An edge $(i, j) \in \mathcal{E}$ exists if there is a direct travel path between cell i and cell j , and its weight is the physical distance l_{ij} between the two cells. The state of cell i at time $t \in \mathbb{R}_+$ is defined by the number of available bikes $S_i(t) \in \mathbb{N}_0$, and the system-wide bike availability at time t is the vector:

$$\mathbf{S}(t) = (S_1(t), S_2(t), \dots, S_d(t)), \quad d = |\mathcal{V}|, \quad (4.1)$$

where d is the total number of nodes in the network. Each node has its own capacity constraints. This includes a minimum available quantity $S_{\min}$, representing the thresh-

old of bikes the policy aims to guarantee at all times. Conversely, a maximum number of bikes is defined only at the macroscopic system level, rather than for individual cells, due to the aggregated nature of the modeling approach. This temporal discretization allows for the modeling of dynamic intra-day behaviors, while the system itself evolves in continuous time: bike movements, arrivals, and departures may occur at any instant.

Let $\mathcal{B}$ denote the set of all bikes in the system. Each bike $b \in \mathcal{B}$ has a position $p_b(t) \in \mathcal{V}$, indicating its current cell, and a battery charge level $c_b(t) \in [0, C_{\max}^b]$, where $C_{\max}^b$ is the maximum battery capacity expressed in units of distance. A bike is considered active if its charge exceeds a specific threshold θ , and depleted otherwise:

$$b \text{ is active} \iff c_b(t) > \theta \cdot C_{\max}^b, \quad \theta \in (0, 1). \quad (4.2)$$

Only active bikes can be used to start a new trip. This operational distinction refines the state of cell i into an active count $S_i^A(t)$ and a depleted count $S_i^D(t)$, with $S_i(t) = S_i^A(t) + S_i^D(t)$. The system also interacts with an external region beyond the boundary of the graph $\mathcal{G}$, which absorbs and generates bike flows. The external region is assumed to have infinite capacity, and any bike originating from it is considered fully charged.

A rebalancing truck operates within the network, traveling along the edges of $\mathcal{G}$ to redistribute the bikes and recharge depleted ones. The truck's position at time t is denoted as $p_{\text{truck}}(t) \in \mathcal{V}$, and its current inventory load is $r_{\text{truck}}(t) \in \{0, 1, \dots, R_{\max}^{\text{truck}}\}$, where $R_{\max}^{\text{truck}}$ is the maximum carrying capacity. At each decision-making step, the truck selects an action from the discrete set

$$\mathcal{A} = \{\text{STAY, UP, DOWN, LEFT, RIGHT, DROP, PICKUP, CHARGE}\}, \quad (4.3)$$

where the four directional actions move the truck to an adjacent cell along $\mathcal{E}$, and the remaining three act on the cell the truck currently occupies. These local actions affect cell i as follows:

- **DROP:** Unloads $\delta_{\text{unload}}(t)$ bikes at the current cell i , updating the state as:

$$S_i(t) \leftarrow S_i(t) + \delta_{\text{unload}}(t) \quad \text{and} \quad r_{\text{truck}}(t) \leftarrow r_{\text{truck}}(t) - \delta_{\text{unload}}(t) \quad (4.4)$$

subject to the constraints:

$$\begin{cases} r_{\text{truck}}(t) \geq \delta_{\text{unload}}(t) \\ S_i(t) + \delta_{\text{unload}}(t) \leq S_i^{\max} \end{cases}.$$

- PICKUP: Loads $\delta_{\text{load}}(t)$ bikes from the current cell i , updating the state as:

$$S_i(t) \leftarrow S_i(t) - \delta_{\text{load}}(t) \quad \text{and} \quad r_{\text{truck}}(t) \leftarrow r_{\text{truck}}(t) + \delta_{\text{load}}(t) \quad (4.5)$$

subject to constraints:

$$\begin{cases} S_i(t) \geq \delta_{\text{load}}(t) \\ r_{\text{truck}}(t) + \delta_{\text{load}}(t) \leq R_{\text{max}}^{\text{truck}} \end{cases}.$$

- CHARGE: Instantly restores the battery level of any depleted bike b at the current cell, or within the truck's inventory, to its full capacity C_{max}^b .

4.2 Modeling User Demand

The user-driven bike movement process is modeled as a piecewise-homogeneous Continuous-Time Markov Chain (CTMC), where transition rates remain constant within a time slot k but may vary across different slots. The truck's operations are treated as exogenous to the process: the CTMC model exclusively describes the stochastic generation and consumption of bike trips by users, independent of the truck's intervention. This separation reflects the architecture of the simulator, which generates user trips uncoupled from the rebalancing logic.

Let $g_{(i,j)}(t)$ denote the number of bikes currently in transit from cell i to cell j , aggregated into the set $\mathbf{T}(t) = \{g_{(i,j)}(t) : i, j \in \mathcal{V}, i \neq j\}$. Given that only active bikes can initiate a journey, they are guaranteed to possess sufficient battery to complete their trip, becoming depleted only upon arrival if the threshold is crossed. The state of the demand process at time t is defined as:

$$\mathbf{X}(t) = \left(S_1^A(t), S_1^D(t), \dots, S_d^A(t), S_d^D(t), S_{\text{ext}}(t), \{g_{(i,j)}(t)\} \right). \quad (4.6)$$

The state space of the CTMC is $\mathcal{S} \times \mathcal{T}$, where $\mathcal{S} = \{(S_1, \dots, S_d, S_{\text{ext}}) \in \mathbb{N}_0^{d+1}\}$ covers all possible cell and external occupancy configurations, and $\mathcal{T} \in \mathbb{N}_0^{d(d-1)}$ covers all possible in-transit configurations, with each $g_{(i,j)} \in \{0, 1, \dots, |\mathcal{B}|\}$.

Travel requests follow a homogeneous Poisson process within each time slot: the number of events in an interval of length t is Poisson-distributed with parameter λt , and inter-arrival times are independent and exponentially distributed with mean $1/\lambda$. This assumption captures the memoryless nature of user demand and enables efficient simulation of the departure and arrival processes within the dockless BSS.

The total departure rate from cell i during time slot k is divided into an internal component (trips ending inside $\mathcal{V}$) and an external component (trips leaving the modeled area):

$$\lambda_i(k) = \lambda_i^{\text{int}}(k) + \lambda_i^{\text{ext}}(k). \quad (4.7)$$

Internal and external departure rates are formulated as:

$$\lambda_i^{\text{int}}(k) = \lambda_i(k) \sum_{j \in \mathcal{V}} p_{i \rightarrow j}(k), \quad (4.8)$$

$$\lambda_i^{\text{ext}}(k) = \lambda_i(k) p_{i \rightarrow \text{ext}}(k), \quad (4.9)$$

where $p_{i \rightarrow j}(k)$ is the probability that a trip starting at cell i during time slot k concludes at cell j , and $p_{i \rightarrow \text{ext}}(k)$ is the probability that the bike exits the system boundary. These probabilities satisfy the normalization condition:

$$\sum_{j \in \mathcal{V}} p_{i \rightarrow j}(k) + p_{i \rightarrow \text{ext}}(k) = 1. \quad (4.10)$$

The rate associated with trips from cell i to cell j is defined as:

$$\lambda_{i \rightarrow j}(k) = \lambda_i(k) p_{i \rightarrow j}(k). \quad (4.11)$$

A trip between these two cells can only occur while $S_i^A(t) > 0$. Upon departure, the system state transitions as follows:

$$S_i^A(t) \rightarrow S_i^A(t) - 1, \quad g_{(i,j)}(t) \rightarrow g_{(i,j)}(t) + 1 \quad \text{for } i \neq j. \quad (4.12)$$

Symmetrically, the arrival rate at cell i aggregates internal arrivals from other cells and external arrivals entering from outside the spatial domain:

$$\mu_i(k) = \sum_{j \in \mathcal{V}} \lambda_j(k) p_{j \rightarrow i}(k) + \lambda^{\text{ext}}(k) p_{\text{ext} \rightarrow i}(k), \quad \text{with } \sum_{i \in \mathcal{V}} p_{\text{ext} \rightarrow i}(k) = 1. \quad (4.13)$$

A bike traveling between i and j completes its journey at rate $\rho_{i \rightarrow j}$, defined as the inverse of the mean travel duration between the two cells:

$$\rho_{i \rightarrow j} = \left(\frac{l_{ij}}{\mathbb{E}[v]} \right)^{-1}, \quad (4.14)$$

where $\mathbb{E}[v]$ is the expected travel speed, sampled as described in Section 4.3. If $g_{(i,j)}(t) > 0$, the total completion rate for that route is $g_{(i,j)}(t) \cdot \rho_{i \rightarrow j}$. Upon trip completion, the

state transitions are defined as:

$$g_{(i,j)}(t) \rightarrow g_{(i,j)}(t) - 1, \quad \begin{cases} S_j^A(t) \rightarrow S_j^A(t) + 1, & \text{if the bike arrives active} \\ S_j^D(t) \rightarrow S_j^D(t) + 1, & \text{otherwise} \end{cases} . \quad (4.15)$$

Compiling the departure and arrival rates above yields the infinitesimal generator matrix $Q(k)$ for each time slot k , governing the CTMC evolution over the joint state space $\mathcal{S} \times \mathcal{T}$. Rather than solving the CTMC analytically, sample paths are generated numerically: departure and travel-completion events are drawn from their respective exponential distributions and inserted into a time-ordered event list. The resulting process, however, is not continuously observed: the simulation is executed in fixed increments of $\delta_a = 30$ s, and at each increment, all events with a (continuously) recorded timestamp that lies within that interval are applied to the state in their chronological order. This is equivalent to observing the CTMC $\mathbf{X}(t)$ only at discrete intervals of time $t_m = m\delta_a$, $m \in \mathbb{N}_0$, resulting in a Discrete-Time Markov Chain (DTMC) $\{\mathbf{X}(t)_m\}_m$ obtained by sampling the underlying continuous-time process at a fixed rate.

4.3 Battery Dynamics

Bikes consume battery proportionally to distance traveled. If a bike with initial charge $c_b(t)$ travels from cell i to cell j and has sufficient charge to complete the trip (i.e. $c_b(t) \geq l_{ij}$), its residual charge after a travel time $\tau \geq l_{ij}/v$ is

$$c_b(t + \tau) = c_b(t) - \tau \cdot v, \quad (4.16)$$

where v is the average travel speed for that trip. Travel speed is not constant; instead, it is sampled independently per trip from a truncated Gaussian distribution with mean μ_v , standard deviation σ_v , and bounds $[v_{\min}, v_{\max}]$, which captures realistic traffic fluctuations while excluding unrealistic extremes:

$$f(v \mid \mu_v, \sigma_v, v_{\min}, v_{\max}) = \frac{1}{\sigma_v} \frac{\varphi\left(\frac{v - \mu_v}{\sigma_v}\right)}{\Phi\left(\frac{v_{\max} - \mu_v}{\sigma_v}\right) - \Phi\left(\frac{v_{\min} - \mu_v}{\sigma_v}\right)}, \quad v \in [v_{\min}, v_{\max}], \quad (4.17)$$

where $\varphi(\cdot)$ and $\Phi(\cdot)$ denote the standard normal probability density function and cumulative distribution function, respectively. The expected value $\mathbb{E}[v]$ used for trip completion rates in Equation (4.14) is the mean of this truncated distribution.

4.4 The Multi-Area Setting

The mathematical formulation detailed above describes a single operating area. This work, however, considers n areas simultaneously, denoted by $a = 1, \dots, n$, each corresponding to a geographically distinct portion of the city (e.g., different regions of Manhattan). Every area a is an independent instance of the previously established model, characterized by its own graph $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a)$ consisting of $d_a = |\mathcal{V}_a|$ nodes, an independent population of bikes $\mathcal{B}_a$, and a dedicated rebalancing truck. Crucially, the number of nodes d_a need not be uniform across areas, as different parts of the city can be partitioned into networks of varying sizes and topological configurations.

Each area a evolves according to its own CTMC model, defined by time-slot-dependent transition rates $\lambda_i^{(a)}(k)$ and $\mu_i^{(a)}(k)$, which are estimated independently from the demand experienced in that region. The joint state of the multi-area system at time t is the tuple of the individual area states:

$$\mathbf{X}_{\text{joint}}(t) = \left(\mathbf{X}_1(t), \mathbf{X}_2(t), \dots, \mathbf{X}_n(t) \right). \quad (4.18)$$

The multi-area framework does not constitute a single coupled macroscopic CTMC, but rather n parallel and structurally identical instances of the same basic model, differing solely in their graph topologies $\mathcal{G}_a$ and demand parameters.

This structural independence is fundamental for enabling the control of all n trucks through a single, shared decision-making policy, as will be discussed in Chapter 5. Since each area is defined by the same set of state variables, such as node occupancy, truck position, truck load, and battery levels, regardless of its spatial dimension d_a , a policy formulated over this generalized representation can be evaluated and applied identically to any area. The formulation presented in the previous sections is area-independent: each system variable ($S_i(t)$, l_{ij} , $p_{\text{truck}}(t)$, $r_{\text{truck}}(t)$, $\mathcal{A}$) is defined per area but retains the same semantics across areas. The model therefore adapts naturally to the multi-area setting without requiring any structural modifications other than the inclusion of the area index a .

5

Proposed Framework

This chapter translates the system model introduced in Chapter 4 into a concrete RL framework for dynamic rebalancing in dockless BSSs. It first describes the simulation environment and the preprocessing pipeline used to construct the operational graph and stochastic demand model. The rebalancing problem is then formulated as an MDP, defining the state-action representations, the transition dynamics, and the reward function. Furthermore, it presents the learning architecture, which is based on PPO and Graph Attention Network v2 (GATv2) layers. Finally, the framework is extended to multiple geographically distinct areas through parameter sharing, enabling a single policy to be trained across different network topologies and demand patterns.

5.1 Environment and Data Pipeline

The simulation environment is constructed through an offline preprocessing pipeline that combines urban road-network data with historical bike-sharing records. The pipeline is executed before training and its outputs are cached for subsequent use. The trip records are obtained from Citi Bike’s published journey data for Manhattan, New York, covering the period September to October 2024 [49]. The underlying road network is extracted from OpenStreetMap using the OSMnx library. The retrieved network is restricted to the bike-accessible roads within a bounding box around the area of interest and reduced to its largest strongly connected component. This preprocessing step ensures that the retained road network is connected and that the relevant road nodes remain mutually reachable. To remove artificial subdivisions caused by dense or irregular intersections, road nodes located within 15 meters of one another are merged into a single intersection. Isolated nodes that are not connected to the main network are then removed. Edge-level attributes, including distance, speed, and travel time, are

attached to the resulting road graph.

The cleaned road network is discretized using a regular grid into *diamond-shaped* cells (i.e., squares with sides of 300 m rotated by 45°). This discretization defines the topological graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, introduced in Chapter 4, where:

- $\mathcal{V}$ the set of cells containing at least one road intersection,
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges connecting adjacent cells.

Each cell $i \in \mathcal{V}$ is represented as a node in the navigation graph of the truck and it is uniquely identified by its central node n_i , which is the road node closest to the geometrical center of the cell. Each node is considered as a bike station and is characterized by a unique ID n_i , geographical coordinates $(\text{lat}_i, \text{lon}_i)$, request rate λ_i and arrival rate μ_i , and a list of available bikes in accordance with [7]. The shortest-path distance between every pair of road nodes is computed offline using Dijkstra's algorithm [50] and stored in a distance matrix D .

Figure 5.1 shows the resulting graph representation of Manhattan. The blue points represent the operational cells and their corresponding central road nodes, while the red lines represent the road connections used by the truck and the bike-movement simulator.

Trip Simulation

Historical trip records are aggregated into k time slots and grouped by origin i , destination j , and type of day (workday/weekend). For each origin-destination pair, the departure rate is estimated based on its frequency of occurrence using the parameter of the Poisson distribution $\lambda_{i \rightarrow j}(k)$ introduced in Chapter 4 (Equation (4.11)). Because historical observations are spatially sparse, many origin-destination pairs contain no recorded trips even though demand may exist. These zero entries therefore cannot automatically be interpreted as a genuine absence of demand. To obtain a complete demand model, missing rates are estimated using Inverse Distance Weighting (IDW). The rate associated with an unobserved destination is interpolated from nearby observed destinations, with greater weight assigned to geographically closer cells. This procedure converts the sparse empirical matrix into the well-defined Probability Mass Function (PMF) $p_{i \rightarrow j}(k)$ used by the event simulator.

The temporal dynamics of trips are implemented as a discrete-event simulation based on CTMC. By exploiting the property of overlapping independent Poisson processes, the individual departure events of each cell are aggregated into a single aggregate process per cell and the destination of individual trips is determined by sampling from the distribution $p_{i \rightarrow j}(k)$. The probability that a global departure event is assigned

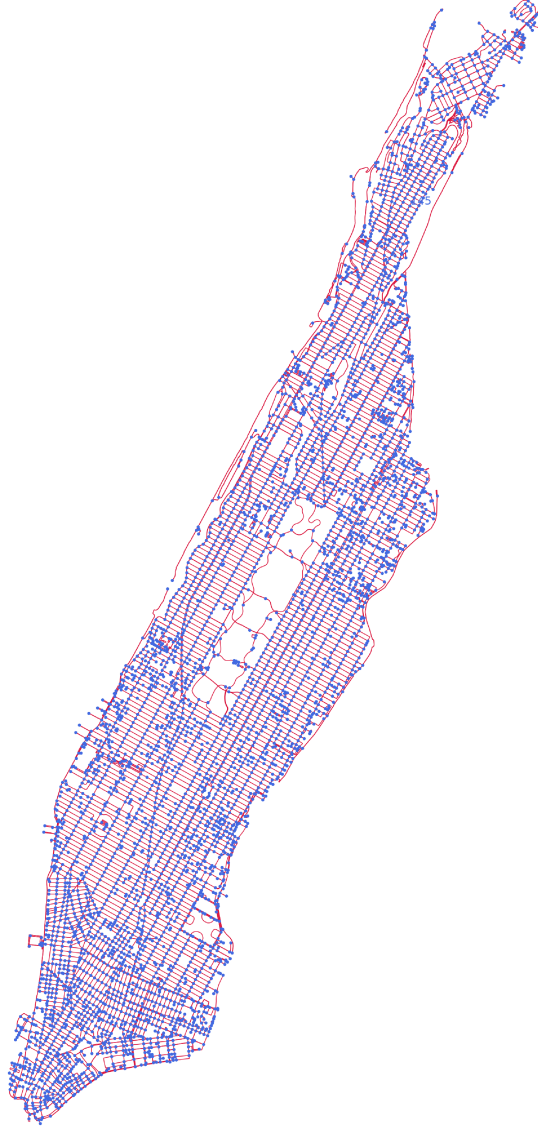


Figure 5.1: Graph representation of Manhattan, New York, used in the dockless BSS environment.

to the specific trip from station i to station j is based on the demand rate:

$$p_{i \rightarrow j} = \frac{\lambda_{i \rightarrow j}(k)}{\lambda_{\text{global}}}, \quad \sum_{i=1}^j \sum_{j=1}^j p_{i \rightarrow j} = 1, \quad (5.1)$$

where λ_{global} is the total system-wide departure rate [7].

A trip fails if no bike with sufficient battery charge to complete the journey is available in the starting cell. However, the failure is not immediate: the simulator first

checks neighboring cells within a fixed search radius (250 m), sorted by distance, and uses the closest one equipped with a suitable bike if the origin cell itself does not have one; a trip handled in this way is marked as *deviated*. Only if no cell within the radius has a usable bike is the trip recorded as a *service failure*, which serves as the primary indicator of rebalancing performance.

5.2 Problem Formulation

The rebalancing problem is modeled as a discrete-time MDP, which is overlaid on the continuous-time dynamics of the simulation. The simulation time is divided into intervals of $\delta_a = 30$ s. An action a selected by the agent occupies the system for a discrete number of steps L_a :

$$L_a = \left\lceil \frac{\tau_a}{\delta_a} \right\rceil, \quad (5.2)$$

where τ_a is the real-world duration of action a . During the L_a steps in which the agent waits for the action to complete, the underlying CTMC simulation continues to evolve, updating the demand and bike states. In particular, since the underlying process can be observed only at the fixed-step intervals, as previously described in Chapter 4, the agent's decision epochs precisely correspond to the sampling times of the DTMC $\{\mathbf{X}(t_m)\}$, without any further discretization at the MDP level being performed.

The state representation provided to the agent at each decision point incorporates local node information and a global state vector. Each cell $i \in \mathcal{V}$ is described by the following features:

- **Bike counts** ($S_i(t)$): Total number of bikes in cell i at time t .
- **Critic (risk) score**: A predictive indicator that quantifies the expected shortage or surplus of bikes during the next time slot. Let D_i denote the peak net outflow from cell i , A_i the inflow from scheduled arrivals, and $S_{\min}$ the minimum stock threshold. Define $E_i = D_i + S_{\min}$. The critic score is given by:

$$\text{cs}_i = \begin{cases} -1 + \exp\left((1 + 0.2 A_i) \left(1 - \frac{S_i(t)}{E_i}\right)\right), & E_i > 0 \\ -S_i(t), & E_i = 0 \end{cases}. \quad (5.3)$$

A cell is considered critical when $\text{cs}_i > 0.05$. The score becomes increasingly positive when the current bike stock is insufficient to cover the projected outflow, and negative when the cell has a sufficient surplus.

- **Eligibility score**: An exponentially decaying trace of recent truck presence. The

score is set to 1 while the truck occupies the cell and subsequently decays according to the cell type. For interior cells, the decay factor is $\eta_{\text{int}} = 0.997$, whereas border cells use $\eta_{\text{border}} = 0.999$. The higher retention factor used for border cells produces a slower decay and compensates for the fact that peripheral cells are visited less frequently. This feature discourages repeated visits to recently served cells while promoting broader spatial coverage.

- **Truck occupancy:** An integer variable indicating the presence of one or more service trucks in the cell. In the multi-area setting, an additional indicator identifies whether the cell contains the truck currently controlled by the shared policy.

In addition, at each step, a separate vector of agent-level features is computed. It contains one-hot representations of the truck’s previous action, its current cell, and the cells currently classified as critical, together with boundary indicators and five scalar features: the fraction of the total fleet currently deployed in the system (including the truck’s own load), the fraction of the fleet that is inactive at the depot, whether the truck’s current cell has a surplus of bikes, whether it is completely empty of bikes, and the average critical score across all cells. This vector is stored in the environment observation for logging purposes.

Action Space

The agent interacts with the environment by performing actions from a predefined set, each of which can be carried out in the real world by the truck. It selects at every decision point one action from the discrete set $\mathcal{A}$, introduced in Equation (4.3): {STAY, UP, DOWN, LEFT, RIGHT, DROP, PICKUP, CHARGE}. The set of actions for the rebalancing truck is divided into two main categories: movement actions and operational actions, as in [7]. Movement actions allow the truck to navigate the grid of cells:

- **STAY:** The truck remains in its current position; this is a no-op action that takes exactly one discrete time step.
- **UP:** Moves the truck from its current cell to the central node of the cell directly above it.
- **DOWN:** Moves the truck to the central node of the cell directly below it.
- **LEFT:** Moves the truck to the central node of the cell to the left.
- **RIGHT:** Moves the truck to the central node of the cell to the right.

If an adjacent cell is not available (for example, at the boundaries of the operational area), the movement action is discarded and converted to a STAY action. Operational

actions allow the truck to interact with the bikes; the time required to perform them depends solely on the movement needed to complete the operation:

- **DROP:** The truck drops off a bike at its current location. If the system has reached the maximum number of allowed bikes and the truck is empty, this operation is equivalent to the STAY action.
- **PICK UP:** The truck picks up a bike. The choice of bike is based on a weighted metric that prioritizes those closest to the truck and with the lowest battery level.
- **CHARGE:** This is a composite action that combines the pickup and drop-off phases, focusing on bikes in urgent need of recharging to instantly restore their battery charge.

As with movements, if an operational action is not feasible, the default action becomes STAY. Furthermore, if the truck becomes completely full or empty during these operations, it automatically travels to the depot, increasing the total duration of the action.

Transition Dynamics

Formalized in Chapter 2, the transition probability function $\mathcal{P}_{ss'}^a = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$ formally exists and belongs to the MDP framework. It is not used or approximated in closed form: it emerges implicitly from the execution of the event-based simulator. The transition dynamics are stochastic (described by the CTMC) because they depend on user demand, battery levels, trip durations, and the truck's actions. The absence of an explicit, tractable transition model motivates the use of model-free RL: the agent improves its policy exclusively based on the sampled trajectories generated by interaction with the simulator, without explicitly representing $\mathcal{P}_{ss'}^a$.

Reward Function

The reward function is designed to encourage the movement of bikes toward cells experiencing shortages while discouraging inefficient, repetitive, or unnecessary actions. The reward configuration is empirical rather than theoretically optimal and was determined through iterative tuning. If the selected action is invalid, the agent receives a fixed negative penalty ($r_{\text{invalid}} = -0.25$) and no additional reward terms are evaluated. For valid actions, the reward is composed of the following terms.

Base Cost. Every (valid) action incurs a minimum, constant cost of $r_{\text{base}} = -0.08$, in order to encourage time efficiency and prevent inactivity from being reward-neutral.

Stay Penalties. It is a penalty applied when a truck remains (STAY action) within the same cell. Choosing STAY in a cell whose critic score marks it as critical results

in a penalty equal to $r_{\text{critic}} = -0.3$; remaining in a non-critical cell results in a smaller, generic penalty equal to $r_{\text{stay}} = -0.08$ (or $r_{\text{no-critic}} = 0$ if the cell requires no action).

Drop / Pick-up Terms. The insertion of a bike into a cell that was critical and becomes balanced as a result produces the largest positive term $r_{\text{drop-reb}} = 1.2$; inserting into a cell that remains critical afterward produces the smallest term $r_{\text{drop-critic}} = 0.7$; dropping at a cell that is already in surplus is discouraged by a penalty equal to $r_{\text{drop-surp}} = -0.6$. Similarly, picking-up from a critical cell is discouraged with a negative reward $r_{\text{pickup-critic}} = -1.0$; picking from a cell that is already unbalanced but not critical incurs a slight penalty $r_{\text{pickup-unbal}} = -0.1$, while picking from a cell with an actual surplus is rewarded with $r_{\text{pickup-surp}} = 0.6$.

Loop Penalty. To discourage immediate reversals and redundant operations, the simulator checks the two most recent actions. A penalty of $r_{\text{loop}} = -0.15$ is applied when the action sequence corresponds to an immediate movement reversal, two consecutive STAY, or pick up and drop a bike in the same cell (PICKUP $\rightarrow$ DROP or DROP $\rightarrow$ PICKUP).

Charging. It involves a reward for recharging discharged bikes located in critical or normal cells, respectively equal to 1.5 and 0.5, and a penalty for unnecessary recharging of bikes that are already fully charged, amounting to -0.2 for both critical and non-critical cells.

Eligibility-based shaping. The eligibility trace directly determines the reward: returning to a cell whose eligibility is still high (above 0.7) results in an $r_{\text{elig-pen}} = -0.03$, discouraging the action; conversely, visiting a cell whose eligibility has fallen below the threshold 0.2 guarantees an $r_{\text{elig-bonus}} = 0.05$, encouraging broader coverage of the operational area. Arriving at a cell with an empty truck when the truck is not needed at that location results in a penalty $r_{\text{elig-empty}} = -0.03$.

Coverage. A global coverage term is applied once per step, independently of the selected action. Let m_{cover} be the number of cells whose eligibility score has decayed below its staleness threshold ($\theta_{\text{int}} = 0.15$ for interior cells, and $\theta_{\text{border}} = 0.3$ for border cells). The coverage penalty is:

$$r_{\text{cover}} = -\omega_{\text{cover-pen}} \cdot \frac{m_{\text{cover}}}{|\mathcal{V}|}, \quad (5.4)$$

where $\omega_{\text{cover-pen}} = 0.15$.

All the valid-action reward terms are summed into a single scalar reward. The resulting reward is then clipped to the interval $[-2, 3]$, before being stored for training. This clipping limits the influence of any individual transition on the subsequent policy update.

5.3 PPO with Graph Attention Networks

The proposed approach uses an on-policy actor-critic framework based on Proximal Policy Optimization (PPO). PPO directly optimizes a stochastic policy over the discrete action space $\mathcal{A}$ while constraining each update to remain close to the previous policy (as specified by the clipped surrogate objective in Equation (2.13)), which results in stable learning even in situations of highly non-stationary reward distributions.

Network Architecture

To process road graphs with variable numbers of cells and irregular topologies, the policy is implemented as a Graph Neural Network (GNN), based on GATv2 layers. Unlike a conventional graph-convolutional layer with fixed attention coefficients, GATv2 computes the attention scores dynamically, as a function of both source and destination node features. This enables the network to learn which neighboring cells are most relevant to the current decision.

The network takes as input the graph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the node-feature vectors, and a scalar edge feature, represented by the normalized distance between cells l_{ij} . Its architecture is shown in Figure 5.2. The first GATv2 layer (*GATv2 Layer 1* in Figure 5.2) maps the node features to a 256-dimensional hidden representation using 4 attention heads of 64 dimensions each. The outputs of the heads are concatenated and passed through Rectified Linear Unit (ReLU)¹ activation function. The second GATv2 layer (*GATv2 Layer 2* in Figure 5.2) produces another 256-dimensional representation using the same four-head structure, followed by a second ReLU activation. A global mean-pooling operation aggregates the node-level embeddings into a single 256-dimensional graph representation. This representation is passed to two separate output heads. The actor head is a three-layer Multi-Layer Perceptron (MLP)² network that maps the global representation to the logits of the categorical distribution of actions $|\mathcal{A}|$. The critic head has the same three-layer structure and outputs the scalar state-value estimate $V(s)$. Since both heads share the same two-layer GATv2 base architecture, the same forward pass produces both the action distribution and the value estimate required for the PPO update.

¹ReLU is a nonlinear activation function defined as $f(x) = \max(0, x)$. It is applied after the dense layers to enable the learning of complex relationships with low computational complexity.

²MLP is a feedforward neural network architecture consisting of sequential fully connected layers.

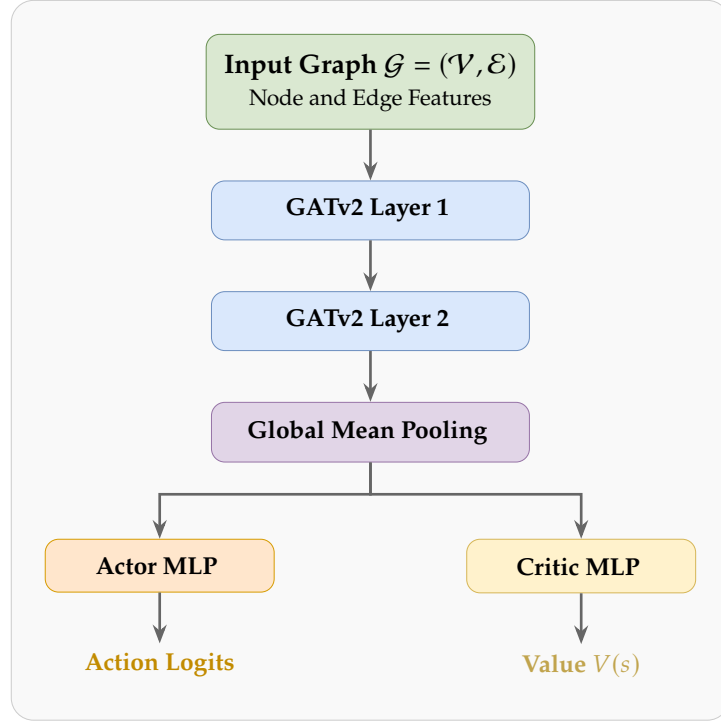


Figure 5.2: GATv2-based Actor-Critic Neural Network Architecture.

Training Mechanisms

The PPO training procedure uses an on-policy rollout buffer with capacity $B = 4096$ steps. During rollout collection, the current policy interacts with the environment, and the buffer stores the state, selected action, log-probability of that action, received reward, critic value estimate, and episode-completion flag. Once the buffer is full, it is used for one policy update and then cleared. No experience is reused across updates.

The advantages $\hat{A}_t^{\text{adv}}$ are computed using GAE, as previously described in Chapter 2 (Equations (2.14) and (2.15)), with discount factor $\gamma = 0.99$ and GAE parameter $\lambda = 0.95$. The objective function to be maximized for the policy is defined by the clipped surrogate objective (Equation (2.13)), where the clipping parameter $\epsilon = 0.2$ limits the deviation between the new and old policies. The total loss optimized by the network (Equation (2.16)) uses parameter $c_1 = 0.25$ and an entropy coefficient $c_2 = 0.02$. At every update, the rollout buffer is processed for 6 epochs, each split into mini-batches of size 512, and gradients are clipped to a maximum norm before the optimizer step to prevent excessively large parameter updates.

The network parameters are updated using the Adam optimizer. Adam adapts the effective learning rate of each parameter by maintaining estimates of the first and second moments of the gradients. This property is particularly useful in PPO because the gradient estimates are generated from noisy, on-policy data whose distribution

changes as the policy evolves.

PPO does not require a separate exploration schedule over actions. Exploration arises directly from sampling actions from the stochastic policy $\pi_{\theta}(\cdot|S_t)$ and is regulated indirectly by the entropy bonus.

5.4 Scaling to Multiple Areas

The framework is extended to n geographically distinct areas, $\{\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_n\}$, each characterized by its own road topology, demand process, and rebalancing vehicle. Since the attention layers of the GATv2 network are invariant to graph size, the same set of parameters ψ is used across all n areas. The policy therefore applies the same decision rule to graphs with different sizes, connectivity, and demand patterns.

The shared training mechanism consists of two main phases, as shown in Figure 5.3:

1. Round-robin experience collection: At each decision step, control is cyclically assigned to a vehicle in a given area. The generated transition $(S_t, A_t, R_{t+1}, S_{t+1})$, containing the specific graph of the current area, is stored in a single shared experience buffer.
2. Single update: Once the buffer reaches its capacity B , a PPO optimization step is performed on the aggregated data from all areas, updating the single set of weights ψ . An area that completes its episode before the others temporarily leaves the rotation until all areas have completed their episodes.

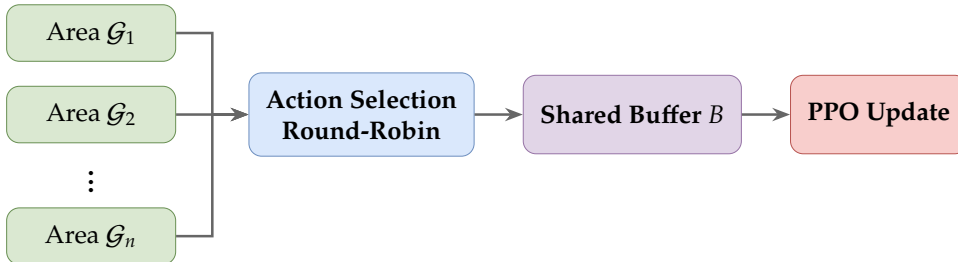


Figure 5.3: Flow of the multi-area extension: round-robin sampling, shared buffering, and joint updating of network parameters.

The shared-learning strategy provides two main benefits. First, exposure to different road topologies and demand patterns can improve the robustness of the policy and reduce overfitting to a single urban configuration. Second, because the policy is not explicitly given the identity of the area it is controlling, it can potentially be applied to a previously unseen area using only the graph structure and the associated node features.

6

Experimental Validation

This chapter presents an evaluation of the proposed RL framework. First, it describes in detail the experimental setup, the geographic areas under study, and the evaluation metrics. Then, it analyzes the approaches drawn from the literature that serve as a benchmark to highlight the limitations of planned fleet management. Finally, it presents the results, focusing exclusively on performance, decision-making behavior, and the multi-area generalization of the policy in the context of dynamic rebalancing.

6.1 Simulation Setup

This section details the experimental setup used to evaluate the proposed framework, outlining the study areas, the simulation configuration, and the metrics adopted to assess the rebalancing performance.

Experimental Areas

To avoid the excessive memory and computational costs associated with modeling Manhattan as a single, massive cell graph, the simulation focuses on a restricted urban area. Specifically, the model is evaluated on two geographically contiguous subareas along the West Side corridor of Manhattan, New York, referred to throughout this work as Area 1 and Area 2. Figure 6.1 provides an overview of the Manhattan road network, with the two selected subareas highlighted along the West Side corridor.

Each area represents an independent instantiation of the system model, featuring its own cell graph, demand profile, and dedicated rebalancing truck. Area 1 (Figure 6.2a) represents the northern region, spanning roughly from 55th Street to 76th Street. It covers approximately 2.58 km^2 and encompasses Lincoln Square, the lower Upper

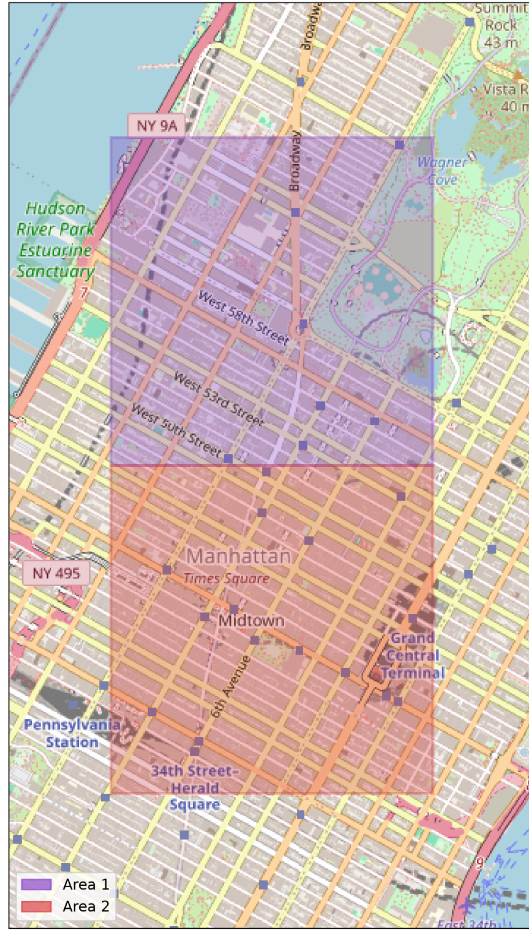
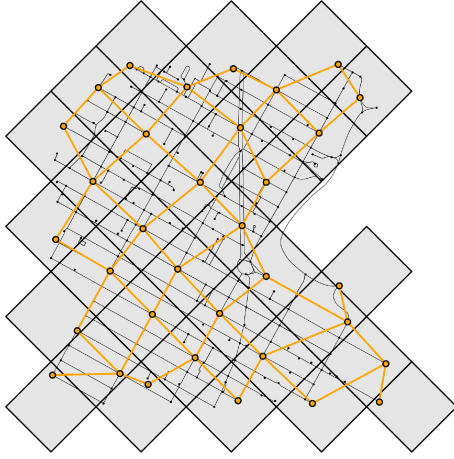


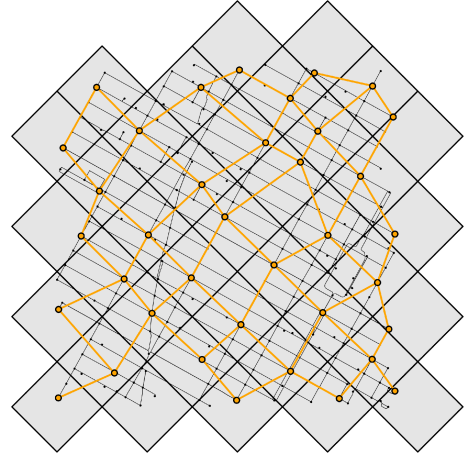
Figure 6.1: Overview of Manhattan, New York, with the two selected subareas highlighted.

West Side, and the southwestern sector of Central Park. Area 2 (Figure 6.2b) represents the southern region, extending roughly from 34th Street to 55th Street. It covers an additional 2.58 km^2 of high-density commercial and residential districts, including Hell’s Kitchen, Midtown West, and Times Square. Notably, the inclusion of Central Park in Area 1 alters the node distribution and subsequent demand patterns, creating significant structural differences between the two zones even though they share equal surface areas (totaling approximately 5.16 km^2).

Both environments are built on the same underlying road network and the same set of historical trip data (Citi Bike trip records for Manhattan covering September and October 2024 [49]) and processed independently using the preprocessing pipeline described in Chapter 5. The two areas differ only in their spatial extent, the number of cells (34 for Area 1 and 37 for Area 2), and the resulting demand statistics. The ability to handle these structurally distinct operational graphs demonstrates the architectural scalability and spatial flexibility of the framework.



(a) Area 1: Upper West Side and Lincoln Square ($\sim 2.58 \text{ km}^2$).



(b) Area 2: Hell's Kitchen and Midtown West ($\sim 2.58 \text{ km}^2$).

Figure 6.2: Graph representations of the two selected subareas in Manhattan, New York.

Training Configuration

To ensure comparable initial conditions rather than starting from an arbitrary point in the week, each training episode begins at the identical simulated timestamp (specifically, on Monday at time slot 0). Given $k = 8$ time slots per day and 56 total slots per episode, a single episode corresponds exactly to one simulated week. The environment parameters are set consistently with the system model of Chapter 4: a bike is considered depleted when its charge falls below $\theta = 0.2$ of the maximum battery capacity. Travel speeds for both bikes and the rebalancing truck are sampled from two distinct truncated Gaussian distributions (Section 4.3). For bikes, the parameters are $\mu_v = 15 \text{ km/h}$, $\sigma_v = 5 \text{ km/h}$, with bounds $v_{\min} = 5 \text{ km/h}$ and $v_{\max} = 25 \text{ km/h}$. For the truck, the same standard deviation $\sigma_v = 5 \text{ km/h}$ is used over a wider range ($v_{\min} = 10 \text{ km/h}$, $v_{\max} = 70 \text{ km/h}$), while mean μ_v is determined by a lookup table parameterized by hour and day, populated with representative values for typical urban traffic congestion, consistent with the demand flows modeled in the simulation.

To prevent computational bottlenecks during inter-episode transitions, the event buffer for the upcoming episode is asynchronously pre-computed by a background process during the current training phase. A dedicated process is launched once training begins, and it is queried for the pre-calculated value whenever a new episode starts.

Table 6.1 reports the hyperparameters used for the training runs. These values are the result of iterative empirical tuning.

Category	Hyperparameter	Value
Optimization	Learning Rate α	$5 \cdot 10^{-5}$
	Mini-batch Size	512
	Rollout Buffer Capacity B	4096
	Update Epochs per Rollout	8
	Gradient Clipping	0.5
Training Setup	Episodes	250
	Time slots per episode	56
PPO	Clipping Coefficient ϵ	0.2
	GAE Parameter λ	0.95
	Value-loss Coefficient	0.25
	Entropy Coefficient	0.02
Discounting	Discount Factor γ	0.99

Table 6.1: Hyperparameters used for training the shared PPO policy.

Evaluation Metrics

Training metrics are continuously logged and made available through a dedicated results dashboard built with Dash and Plotly. The dashboard provides a three-level view of a specific training session:

1. Episode-level trends: Reward, service failures, policy loss, value loss, and policy entropy are plotted by training episode number, along with their current cumulative average, to highlight the overall trend beyond the variation from one episode to the next.
2. Per-episode diagnostics: For a selected training episode, the dashboard displays the distribution of actions taken and the average reward obtained per action. This view was crucial during development for identifying degenerate policies (e.g., over-reliance on movement actions without actual rebalancing). It also includes time series data on distributed bikes, truck load, and depot occupancy over the course of the episode.
3. Spatial visualization: Overlays a heat map on the road network to map metrics like critic score, eligibility score, visit count, or operations per cell. This provides a qualitative assessment about how the policy is distributing its attention on the area or focusing on a few cells.

Since the framework simultaneously trains a single shared policy across multiple areas, all dashboard views can be filtered by individual area or aggregated, facilitating comprehensive policy inspection.

The following metrics, calculated per episode, are used to evaluate rebalancing performance:

- Service failures: The count of relocation requests unmet by either the origin cell or the fallback search radius. This serves as the primary benchmark for rebalancing quality: fewer failures directly correlate with higher bike availability.
- Total episode reward: The sum of the clipped per-step rewards over an episode. Since the reward acts as a proxy objective, this metric functions primarily as a health indicator for the training process rather than a standalone performance measure.
- Policy entropy: The unnormalized entropy of the action distribution (Equation (2.8)), averaged per batch. A gradual decrease is expected as the policy specializes; conversely, a sudden collapse to zero signals premature, potentially degenerate convergence.
- Policy and value loss: The loss metrics used to assess the optimization stability. Persistently high value losses act as an early warning for an inaccurate critic, which could subsequently destabilize policy updates.

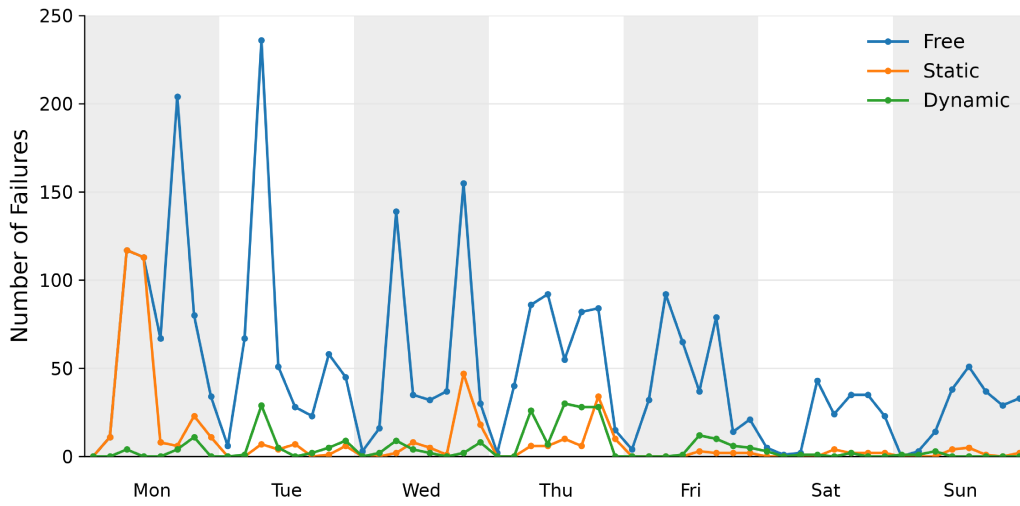
Following each training episode, a validation phase evaluates the newly saved policy snapshot on an unseen episode using the same stochastic action sampling. Operating on a rotation across all areas, this step yields a single, aggregated mean failure score. This score is checked against the historical best and retained only if it represents an improvement, ultimately identifying the optimal final policy configuration.

6.2 Comparison between Static and Dynamic Rebalancing

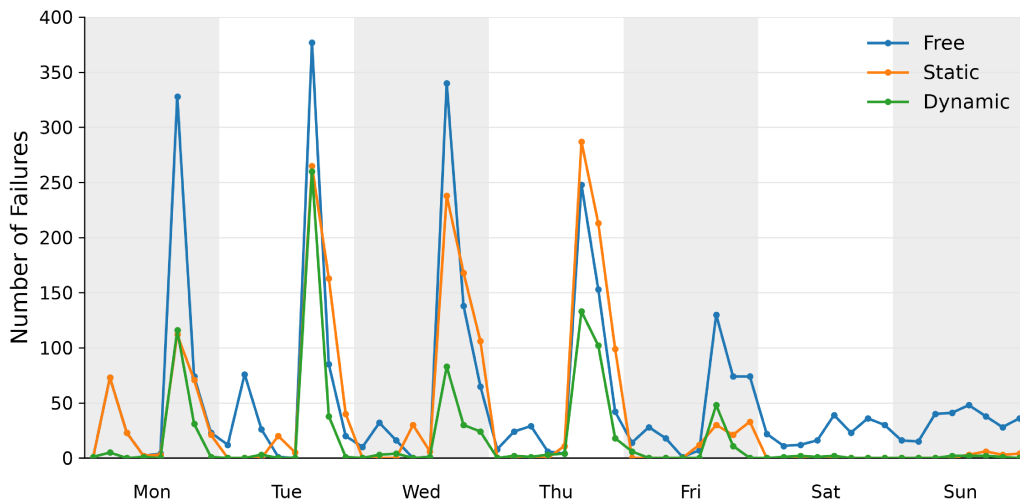
Evaluating the learned policy requires establishing robust baselines. This thesis considers two benchmark scenarios: a free scenario, where no rebalancing occurs and redistribution relies solely on user trips, and a Static Rebalancing (SR) baseline, modeled after the classical O'Mahony's approach traditionally employed by Citi Bike [8]. In the SR baseline, bike redistribution occurs instantly twice a day, at 1 AM and 1 PM. In contrast, the dynamic approach proposed in this work adapts O'Mahony's framework to execute hourly rebalancing, allocating bikes based on the projected net flow (departures minus arrivals) for the upcoming time slot. To guarantee a fair comparison, all three configurations (free, static, and dynamic) are evaluated using identical fleet sizes over one simulated week (Monday-Sunday).

Figure 6.3 shows the number of failures per time slot for Area 1 and Area 2, respectively, comparing the free, static, and dynamic configurations. The same pattern

emerges for both study areas: the dynamic approach constantly rebalances the system more effectively than the static baseline scenario, which in turn consistently outperforms the free scenario. Nevertheless, Area 2 exhibits severe demand peaks on Tuesday and Thursday, which reveal key operational differences between the three configurations. On Tuesday, the extreme demand saturates the system's capacity, so a residual number of failures persists even under dynamic control. On Thursdays, sharp fluctuations in demand cause the rigid static policy to perform worse than the free scenario, since its scheduled re-allocations incorrectly place vehicles in cells that are not experiencing the spike; the dynamic policy, on the other hand, adapts in real time and avoids this degradation.



(a) Area 1.



(b) Area 2.

Figure 6.3: Number of failures per time slot for the free, static, and dynamic configurations, in Area 1 and Area 2.

This qualitative trend is strongly supported by the aggregated weekly metrics. Over the simulated week, the free scenario accumulated a total of 2760 service failures in Area 1 and 3037 in Area 2. In the static baseline scenario, the numbers of failures decreased to 498 and 2069 for Area 1 and Area 2 respectively, corresponding to reductions of 82.0% and 31.9% compared to the free scenario. The dynamic approach achieved the best overall performance, reducing the total number of failures to 262 in Area 1 (90.5% reduction from the free scenario and an additional 47.4% reduction from the static baseline) and 943 in Area 2 (68.9% reduction from the free scenario and an additional 54.4% reduction from the static baseline).

This confirms that periodic redistribution, based on a predetermined schedule, already offers a measurable advantage over the complete absence of intervention, but that a continuously adapting approach is capable of correcting imbalances in demand that a static scheme with two interventions per day is unable to anticipate or address.

In literature, static rebalancing schemes are evaluated solely on their redistribution efficacy, often neglecting the actual travel time required by the trucks. The implemented simulator explicitly models this cost: for every scheduled event, the reallocation is carried out via a truck route based on the Traveling Salesman Problem (TSP) through the cells involved, recording the precise travel time. Figure 6.4 shows the duration of this rebalancing for the static configuration, which has two non-zero peaks per day due to the planned rebalancing events at 1 AM and 1 PM. The free scenario is not considered, since no intervention is performed, and the dynamic configuration is intentionally excluded from this specific temporal metric, as the agent continuously maneuvers the fleet throughout the day.

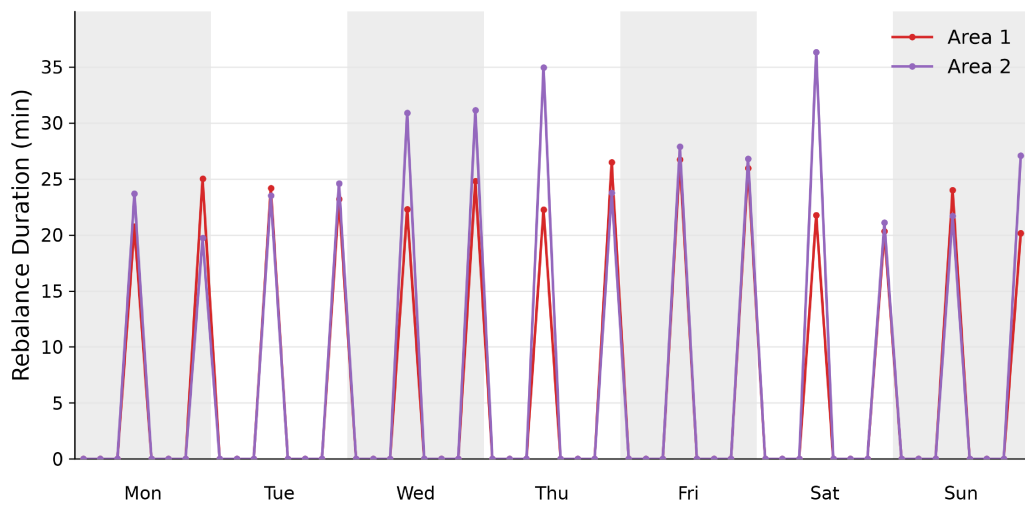


Figure 6.4: Rebalance duration per time slot for the static configurations, in Area 1 and Area 2.

Figure 6.4 shows that, in nearly all events, Area 2 requires a rebalancing time equal to or greater than that of Area 1, with the most pronounced gap occurring midweek and on Saturdays. This is consistent with the higher number of failures already observed for Area 2 in both the free and static configurations (Figure 6.3): a greater imbalance caused by demand necessitates the redistribution of a larger number of bikes during each scheduled event, which directly results in a longer rebalancing time. Area 2 places a structurally greater demand on the rebalancing system than Area 1 and, consequently, requires more extensive intervention to keep the system in balance.

6.3 Analysis of the Results

Building on the simulation environment and the PPO-based framework defined in the previous chapters, this section presents the training and evaluation of the proposed rebalancing agent. The shared policy was trained jointly across both study areas for 250 episodes, using the hyperparameter configuration defined in Table 6.1, which resulted from a tuning process.

The training process demonstrates robust convergence and optimization stability. The overall training metrics show that the agent goes through two distinct learning phases, separated by a significant transition around episode 145. As shown in Figure 6.5, during the first 140 episodes, the policy entropy (see (2.8)) remains high and comparatively stable (~ 1.5), promoting broad, uniform exploration across the action space. At episode 145, a major structural adjustment occurs, characterized by a sharp temporary drop. Following this transition, entropy undergoes a smooth, monotonic decay towards $\approx 0.2\text{--}0.4$, indicating a shift from exploration to exploitation.



Figure 6.5: Policy entropy across training episodes.

The critic network’s value loss demonstrates early stability, staying strictly under 2 throughout the initial 140 episodes. Even during the major policy transition and the subsequent exploitation phase, the value loss remains firmly bounded below 9, indicating robust value estimation without divergence. Figure 6.6 illustrates this trend.

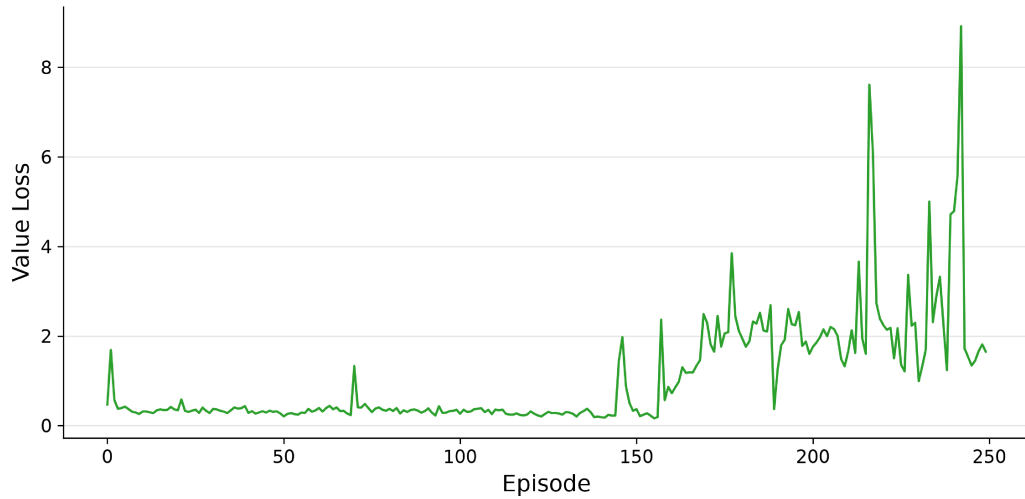
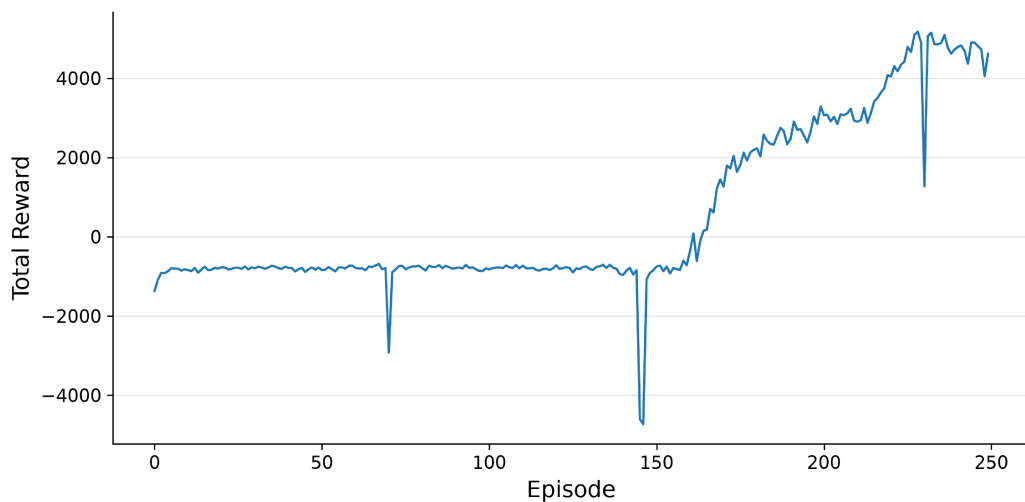
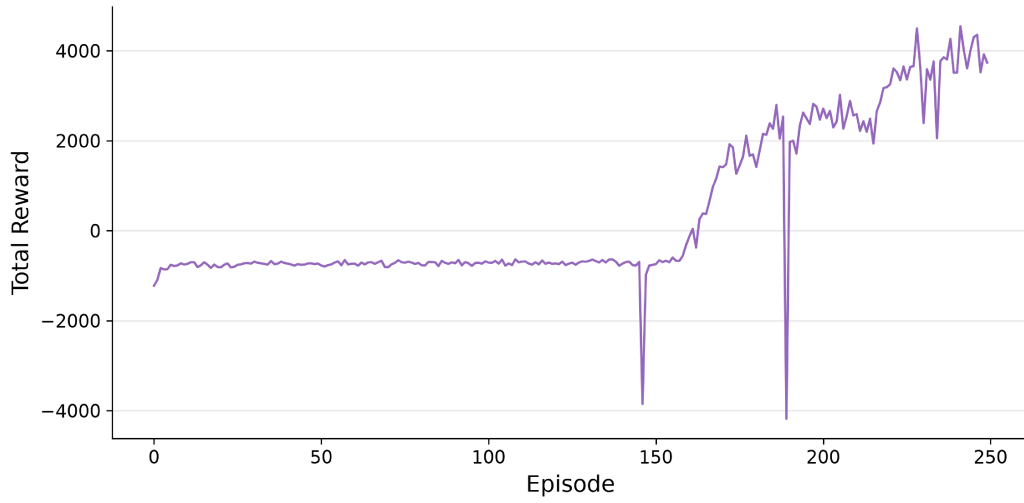


Figure 6.6: Critic network’s value loss across training episodes.

Initially, the total reward fluctuates around a negative baseline of ≈ -1000 in both study areas, as shown in Figure 6.7. Following the structural adjustment at episode 145, the reward rises steadily and significantly, ultimately exceeding 5000 in Area 1 and 4000 in Area 2, reflecting the successful convergence of the agent’s strategy.



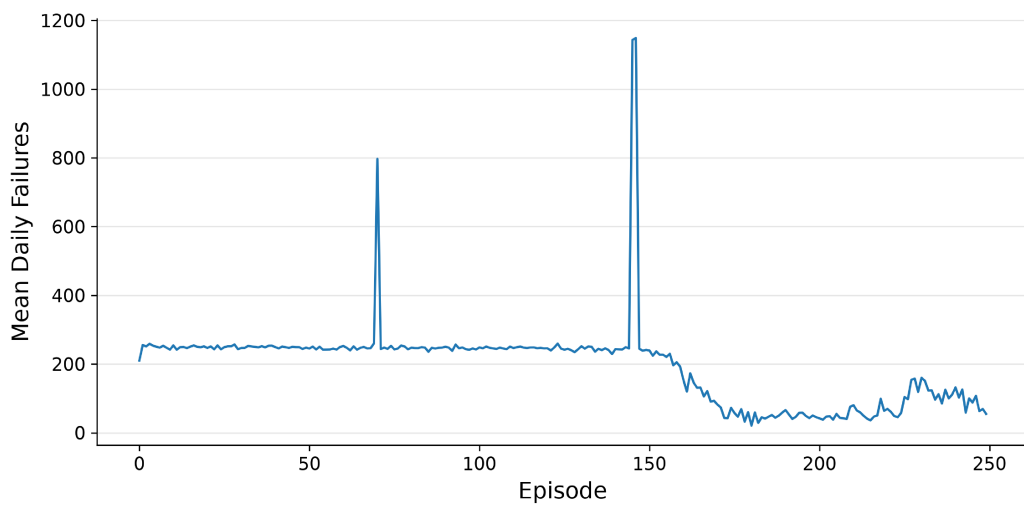
(a) Area 1.



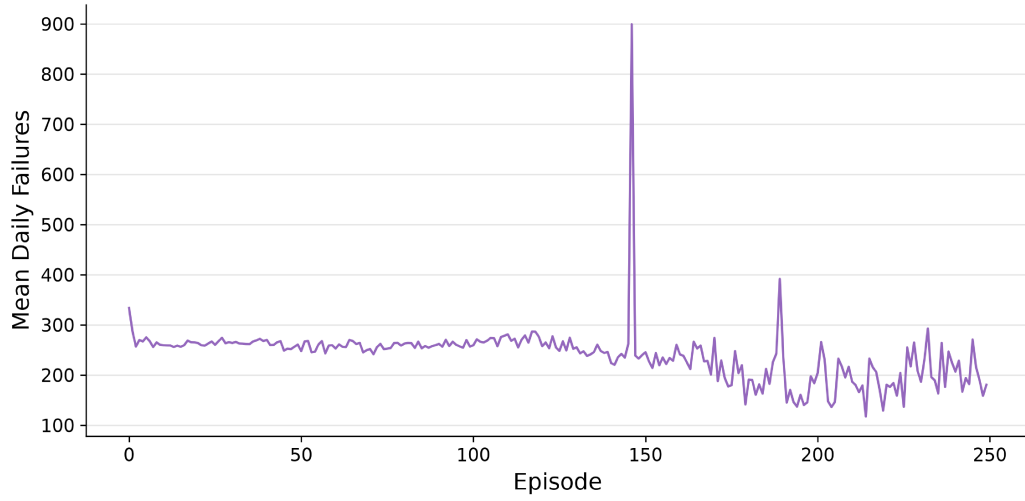
(b) Area 2.

Figure 6.7: Evolution of total reward during training for Area 1 and Area 2.

As illustrated in Figure 6.8, in the exploration phase, mean daily failures remain stable near the baseline rate of ~ 250 in Area 1 and ~ 250 – 300 in Area 2. The episode 145 transition triggers localized spikes, reaching ~ 1150 in Area 1 and ~ 900 in Area 2. After episode 145, failures decrease rapidly: Area 1 drops to a highly stable range of fewer than 150 per day, while Area 2 stabilizes at a higher residual range of 150 to 250 per day. This performance gap highlights that, although the shared policy optimizes both areas simultaneously, Area 2 presents inherently more challenging spatial constraints.



(a) Area 1.



(b) Area 2.

Figure 6.8: Mean daily failures during training for Area 1 and Area 2.

Furthermore, the analysis of the best validation episode reveals key insights into the learned agent’s operational logic.

Regarding the distribution of actions per episode, shown in Figure 6.9, data are recorded jointly for the two areas, unlike the other metrics previously described. During the best validation episode, the policy gives high priority to rebalancing actions, executing DROP and PICKUP as the predominant choices. Navigation actions (UP, DOWN, LEFT, RIGHT) are selected uniformly (approximately 2000 times each) for positioning, while passive actions (STAY and CHARGE) are never chosen. This directly reflects the mean reward function (Figure 6.10): DROP (≈ 0.42) and PICKUP (≈ 0.25) yield strictly positive average rewards, directional movements incur a uniform movement cost (≈ -0.2), and STAY is heavily penalized (≈ -0.6).

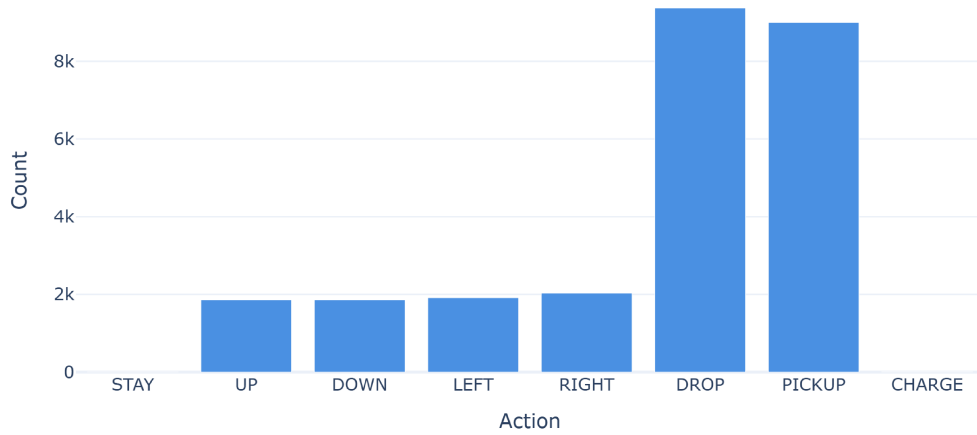


Figure 6.9: Joint distribution of actions executed during the best validation episode.

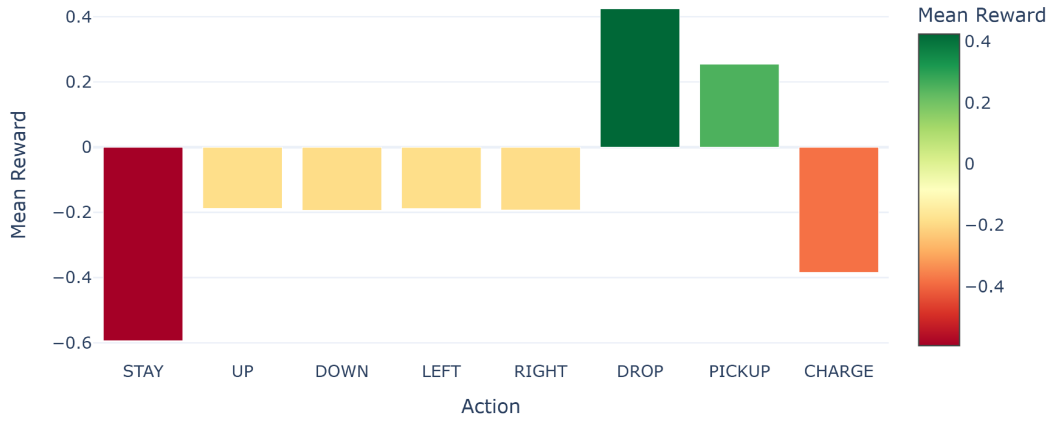


Figure 6.10: Mean reward function distribution across the action space.

Figure 6.11 shows how rebalancing operations are distributed evenly throughout the network, rather than being concentrated on a few high-demand nodes. In Area 1, cell operational shares range from 0.66% at outer boundaries to a maximum of 5.39% in the central hub. Similarly, Area 2 exhibits balanced coverage ranging from 0.56% to 5.47%. This confirms that the shared policy avoids spatial overfitting.



Figure 6.11: Spatial distribution of rebalancing operations for Area 1 and Area 2 during the best validation episode.

As shown in Figure 6.12, service failures are heavily concentrated in specific locations on the grid. In Area 1, most grid cells have a failure rate of 0.00%, with failures concentrated at a single critical point (5.71%) on the upper-right perimeter. In Area 2, failures are concentrated in a cluster in the lower left (8.50% and 4.48%). These critical

points persist across several training runs and hyperparameter settings and represent physical bottlenecks (such as local demand spikes that exceed truck capacity) rather than RL convergence errors.

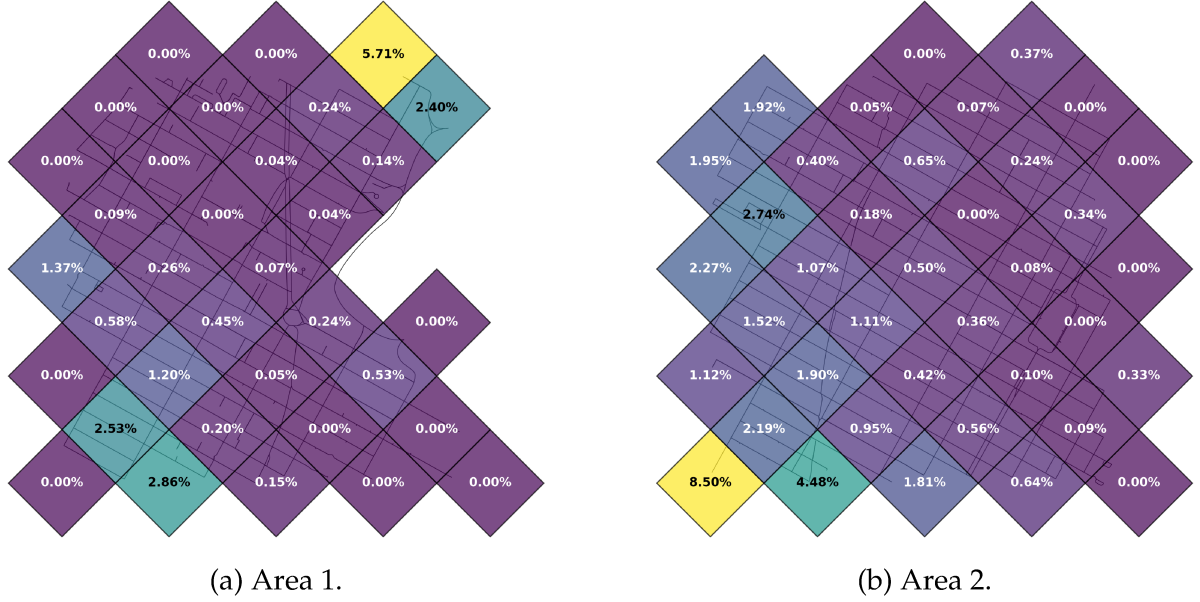


Figure 6.12: Spatial distribution of the service failure rate for Area 1 and Area 2 during the best validation episode.

The key finding of this work is that a single shared network is capable of effectively controlling fleet operations in two distinct urban settings without requiring specific redesign or fine-tuning for each area, demonstrating a strong capacity for generalization in the management of multi-area fleets.

Learning Rate Comparison

To evaluate the impact of the learning rate, defined as α in Equation (2.5), an alternative configuration trained with $\alpha = 1 \cdot 10^{-4}$ was compared to the training configuration under analysis ($\alpha = 5 \cdot 10^{-5}$).

As illustrated by the entropy comparison in Figure 6.13, the highest learning rate leads to a premature decrease in entropy starting around episode 70, with a drop to ≈ 0.1 by episode 100. However, this rapid shift from exploration triggers severe policy instability, causing entropy to oscillate chaotically between 0 and 0.9 for the rest of the training. In contrast, the training model ($\alpha = 5 \cdot 10^{-5}$) maintains high entropy until episode 140, before undergoing a controlled and monotonic decline.

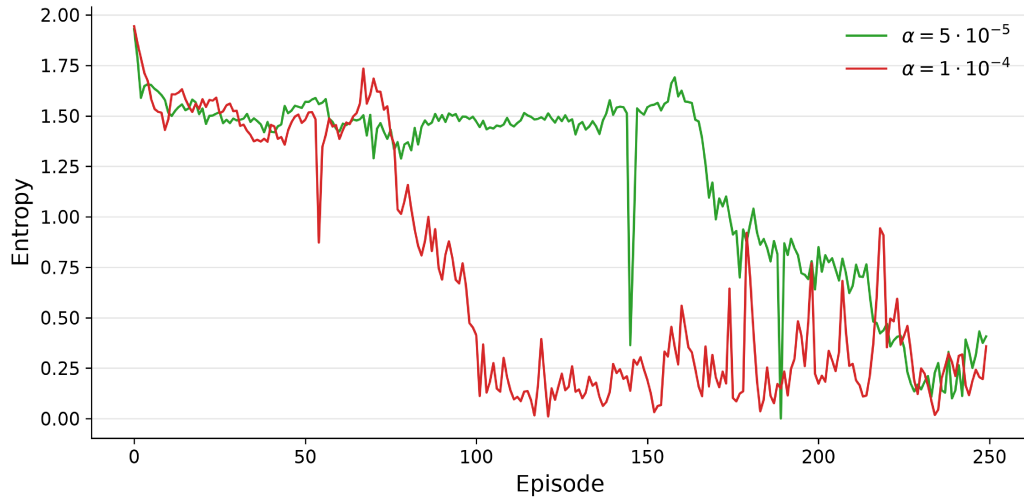


Figure 6.13: Comparison of policy entropy curves for different learning rates.

This optimization instability is shown through the dynamics of value loss (Figure 6.14) where $\alpha = 1 \cdot 10^{-4}$ experiences a huge value loss spike reaching nearly 120 around episode 100 and maintains persistent noise throughout training.

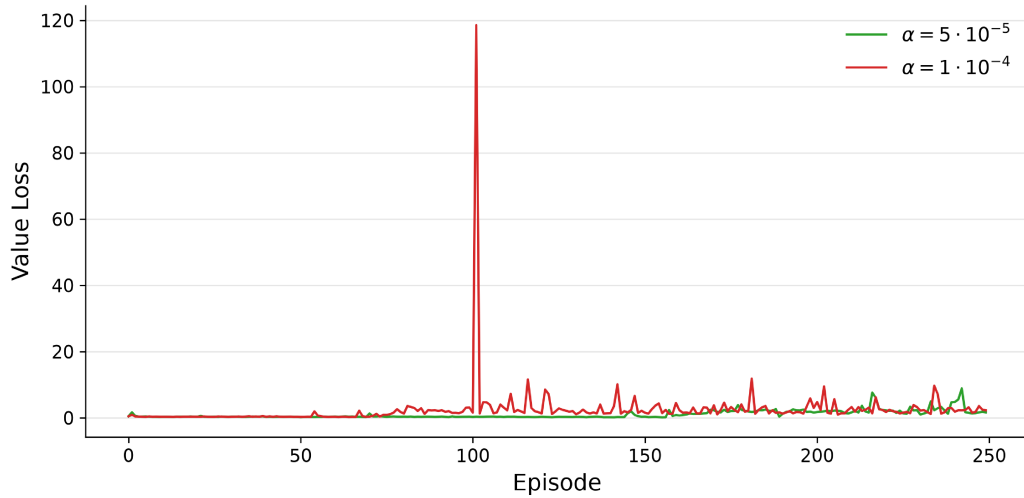
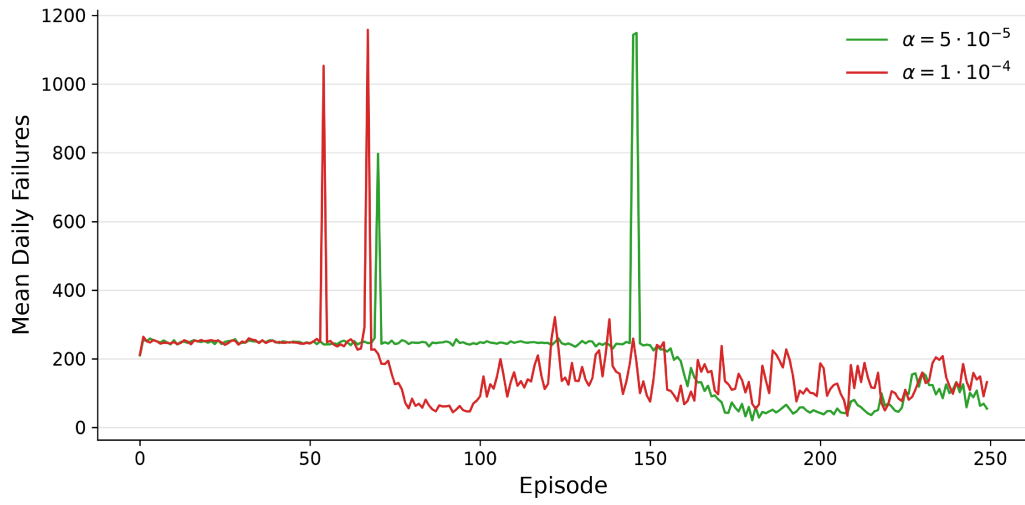
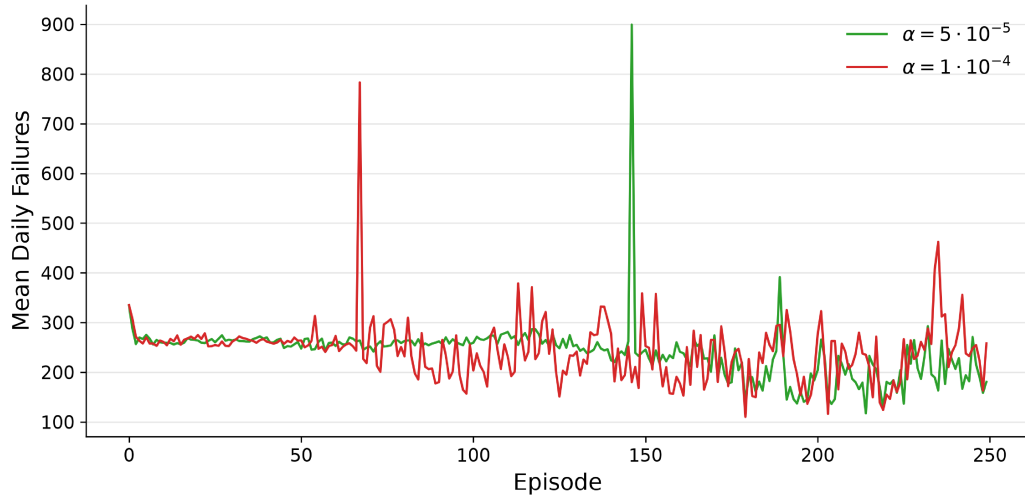


Figure 6.14: Comparison of value loss dynamics for different learning rate.

Consequently, a higher learning rate compromises evaluation performance in both study areas, as shown in Figure 6.15. In Area 1, although $\alpha = 1 \cdot 10^{-4}$ achieves an early decline in failures around episode 70, its performance fluctuates continuously between 50 and 250 daily failures; the conservative baseline ($\alpha = 5 \cdot 10^{-5}$) achieves far greater long-term stability, keeping failures consistently below 150. In Area 2, $\alpha = 1 \cdot 10^{-4}$ shows very pronounced irregular spikes and settles into a lower failure range (150–350) compared to the selected training configuration (150–250).



(a) Area 1.



(b) Area 2.

Figure 6.15: Comparison of mean daily failures across training episodes for Area 1 and Area 2 for different learning rates.

The above results are consistent with the notion that although an increased learning rate accelerates early policy adjustments, it results in poor stability in critic estimation and sub-optimal fleet rebalancing strategies. For these reasons, choosing $\alpha = 5 \cdot 10^{-5}$ is essential for achieving reliable convergence and minimal operational failure rates.

Conclusions

This thesis presented a data-driven, dynamic rebalancing framework for dockless BSSs, formulated as an MDP on a discrete-cell graph and solved through a MARL approach that combines PPO with GAT. The framework was extended to a generalized multi-area setting, in which a single, area-independent policy is trained via parameter sharing across geographically distinct urban regions.

The proposed approach was empirically validated on two structurally different subareas of Manhattan, using historical Citi Bike trip data [49]. Compared against a free (no-intervention) scenario and a static baseline scenario, the learned dynamic policy consistently outperformed them, demonstrating that only a continuously adapting policy is capable of addressing imbalances in demand.

The training process proved to be stable, with the policy transitioning smoothly from the exploration phase to the exploitation phase, and the trained agent distributing its interventions evenly across the entire network, rather than over-adapting to a few high-demand nodes. A comparison of the two study areas revealed that, despite sharing the same policy, Area 2 consistently exhibits a higher number of service failures and longer rebalancing times than Area 1, indicating inherently more challenging spatial and operational constraints. An analysis of the learning rate’s sensitivity further confirmed that a lower, more conservative learning rate is essential for reliable convergence.

Overall, the results demonstrate that a single, shared MARL policy can effectively and reliably manage fleet rebalancing across multiple, structurally distinct urban areas, without requiring area-specific redesign or fine-tuning, confirming the feasibility of the proposed framework as a scalable alternative to traditional static rebalancing schemes.

7.1 Future Work

While the proposed data-driven dynamic rebalancing framework achieves significant improvements in managing dockless BSSs, several promising directions remain open for future research.

A first direction concerns scaling the deployment of the agents and enhancing their operational coordination. Although the current framework already trains a shared policy across multiple areas, each area is served by a single rebalancing vehicle. Expanding the framework to allow multiple rebalancing vehicles to operate simultaneously within the same geographic area would transform the scenario from a single truck per region to a dense environment with multiple agents. This would require advancing the MARL architecture to support the concurrent operation of several trucks, developing synchronization mechanisms that allow agents to coordinate simultaneous pick-up and drop-off tasks in real time, and designing cooperative reward structures that prevent redundant actions and optimize routing when multiple trucks navigate the same environment.

A second direction concerns the spatial organization of the operational areas. In the present formulation, urban subareas are treated as strictly compartmentalized regions; allowing geographical zones to overlap, however, would create dynamic boundary sectors. In these overlapping areas, trucks assigned to neighboring areas could collaboratively manage localized demand spikes or balance cross-border inventories, improving the rebalancing process across the entire urban network.

Building on these concepts of multi-agent coordination and dynamic boundaries, a natural next step is to extend the simulation environment to the entire Manhattan district. Moving from specific sub-zones to a scale that encompasses the entire city would rigorously test the generalization and robustness of the parameter-sharing strategy. Managing Manhattan’s entire road network would validate the framework’s ability to handle large state spaces and highly heterogeneous demand models, demonstrating its potential as a centralized management tool for large-scale urban micro-mobility operations.

Finally, future research could explore hybrid rebalancing strategies that combine an agent-based autonomous routing system with dynamic incentive schemes for users. By introducing variable pricing or discount mechanisms, the system could crowd-source part of the repositioning effort, encouraging users to remove vehicles from areas where there is an excess supply. Integrating human behavior modeling into the RL environment would complete this framework.



System Model Notation

Symbol	Meaning
$\mathcal{G}$	Graph representing the BSS
$\mathcal{V}$	Set of nodes (cells) in the graph
$\mathcal{E}$	Set of edges representing road segments
l_{ij}	Travel distance between cell i and cell j
$S_i(t)$	Number of available bikes at cell i at time t
$\mathbf{S}(t)$	System-wide state vector of bike availability at time t
$d = \mathcal{V} $	Total number of nodes in the network
k	Time slot index within a day
K	Total number of time slots per day
$\mathcal{B}$	Set of all bikes in the system
$p_b(t)$	Position of bike b at time t
$c_b(t)$	Battery charge level of bike b at time t
$C_{\max}^b$	Maximum battery capacity of bike b
θ	Battery charge threshold for active bikes
$S_i^A(t), S_i^D(t)$	Number of active/depleted bikes at cell i at time t
$S_{\text{ext}}(t)$	Number of bikes in the external region at time t
$p_{\text{truck}}(t)$	Position of the rebalancing truck at time t
$r_{\text{truck}}(t)$	Number of bikes currently loaded on the truck at time t
$R_{\max}^{\text{truck}}$	Maximum carrying capacity of the rebalancing truck
$\mathcal{A}$	Discrete set of available truck actions
$\delta_{\text{unload}}(t)$	Number of bikes unloaded from the truck at time t
$\delta_{\text{load}}(t)$	Number of bikes loaded onto the truck at time t
$g_{(i,j)}(t)$	Number of bikes in transit from cell i to cell j at time t
$\mathbf{T}(t)$	Set of all bikes in transit at time t

Symbol	Meaning
$\mathbf{X}(t)$	State of the demand process (CTMC) at time t
$\mathcal{S}, \mathcal{T}$	State spaces for cell configurations and in-transit configurations
$\lambda_i(k)$	Total departure rate from cell i in time slot k
$\lambda_i^{\text{int}}(k)$	Internal departure rate from cell i in time slot k
$\lambda_i^{\text{ext}}(k)$	External departure rate from cell i in time slot k
$\lambda_{i \rightarrow j}(k)$	Departure rate from cell i to cell j in time slot k
$\mu_i(k)$	Total arrival rate at cell i in time slot k
$p_{i \rightarrow j}(k)$	Probability of a trip starting at i ending at j in time slot k
$\rho_{i \rightarrow j}$	Travel completion rate from cell i to cell j
$\mathbf{Q}(k)$	CTMC infinitesimal generator matrix in time slot k
v	Travel speed of a bike
μ_v, σ_v	Mean and standard deviation of the travel speed distribution
$v_{\min}, v_{\max}$	Minimum and maximum travel speed limits
n	Total number of areas
a	Area index
$\mathbf{X}_{\text{joint}}(t)$	Joint state of the multi-area system at time t

Table A.1: List of symbols used in Chapter 4.

Acknowledgments

This study was carried out within the MOST (the Italian National Center for Sustainable Mobility) and received funding from NextGenerationEU (Italian PNRR – CN00000023 - D.D. 1033 17/06/2022 - CUP C93C22002750006).

Ringraziamenti

Arrivare alla fine di un percorso significa, inevitabilmente, voltarsi indietro. E guardando questi anni mi accorgo che questo traguardo, pur portando il mio nome, non è stato raggiunto da sola. È fatto di incontri, presenze, parole, risate, difficoltà condivise e piccoli gesti che, giorno dopo giorno, hanno contribuito a rendere possibile il cammino fino a qui.

Queste righe sono per tutte le persone che, in modi diversi e in momenti diversi, ne hanno fatto parte.

Al professor Gian Antonio Susto, a Edoardo, Federico, Marco e Matteo, per avermi accompagnata durante gli ultimi mesi di questo percorso con disponibilità, pazienza e attenzione. Grazie per il tempo dedicato, per l'aiuto prezioso e per aver saputo trasmettermi, oltre alle conoscenze, anche la passione e la curiosità verso gli argomenti trattati.

Casa

A Mamma e Papà, per avermi accompagnata fin dai primi passi, assistendo a ogni cambiamento, a ogni conquista e anche a ogni caduta. Siete stati presenti nei traguardi più importanti, a volte fisicamente, altre semplicemente con un pensiero, ma soprattutto avete saputo esserci nelle giornate ordinarie, quelle che nessuno ricorda ma che, alla fine, costituiscono una vita. Avete raccolto le mie preoccupazioni, ascoltato i miei momenti di sconforto, trovato una parola capace di rimettere ordine quando tutto sembrava più complicato e continuato a spronarmi a dare il massimo anche quando io stessa facevo fatica a farlo. Mi avete sostenuta non soltanto nel percorso universitario, ma in ogni aspetto della vita, senza mai chiedermi di essere diversa da quella che sono.

A Rachele, la mia ancora. Da quando eravamo bambine siamo cresciute insieme attraverso bisticci, risate, cambiamenti e tutto quello che è successo nel mezzo. Sei stata la mia spalla nei momenti più difficili e la compagna delle giornate più leggere, una delle poche persone davanti alle quali non ho mai avuto bisogno di fingere che andasse tutto bene. Con te posso essere esattamente quella che sono, senza spiegazioni e senza filtri. Sei la persona più importante della mia vita e, pensando a tutti questi anni, faccio fatica a immaginare come li avrei attraversati senza di te.

A nonna Angela, ad ogni passo fatto fin qui che porta con sé anche qualcosa di tuo. A casa tua sembrava sempre possibile fare ciò che altrove non avremmo mai fatto. C'era una libertà diversa, una complicità fatta di piccoli gesti e la sensazione che, almeno lì, tutto fosse concesso. Sei stata presente nei momenti importanti e ancora oggi so che, per noi, metteresti la tua vita davanti alla nostra senza esitazione. Mi hai insegnato, soprattutto attraverso il tuo modo di essere, a diventare una donna indipendente, a non sentirmi inferiore e a non rinunciare a fare ciò che fanno gli altri soltanto perché qualcuno potrebbe pensare che non sia per me. Questo traguardo porta con sé anche il desiderio di restituirti, almeno simbolicamente, una parte delle possibilità che la vita non ti ha concesso.

A nonno Erminio, alle prime lezioni di guida in Vespa e poi in macchina, alle discese dalle scale sulla ringhiera, ai fossi attraversati e a quel modo un po' folle di affrontare la vita che forse, in parte, ho ereditato proprio da te. Alla metafora di vita che mi hai lasciato: come la macchina ha sempre la retromarcia, anche noi possiamo sempre tornare indietro e cambiare strada.

A nonna Camilla, a quella casa sempre pronta ad accoglierci, a qualcosa di caldo preparato per noi, a patto di non mangiare le "fugasette" prima del previsto, e a quel modo discreto, quasi silenzioso, di dimostrare l'affetto. Ci sono persone che hanno bisogno di molte parole per dire ciò che provano e altre che riescono a farlo in modo più semplice.

A zia Susanna, la mia seconda mamma, a tutti quei momenti di quotidianità che, con il tempo, hanno finito per diventare ricordi preziosi. Grazie per aver sempre saputo riconoscere il mio valore in ogni occasione e per avermi aiutata a riconoscerlo a mia volta.

A zia Sonia, alla zia giovanile e capace di trasformare ogni occasione in una piccola vacanza. Alla tua allegria, alla leggerezza che porti con te e alla capacità di far sembrare speciale anche ciò che, in apparenza, non ha nulla di straordinario.

A zio Gabriele e zio Gian Paolo, per la presenza e l'accompagnamento che hanno attraversato questi anni, facendone parte con discrezione, affetto e costanza.

A Gianluca, Leonardo e Lorenzo. Siamo cresciuti quasi come fratelli, condividendo la scuola, i pomeriggi in piscina, i cartoni davanti alla televisione e le recite improvvisate durante i pranzi di Natale. Abbiamo avuto il privilegio di crescere nello stesso tempo, osservandoci cambiare senza quasi rendercene conto.

A Sofia, la piccola di casa, alla passione per Tigotà, ai pranzi al McDonald's e alle lezioni di tedesco. Sei arrivata quando gli altri erano già cresciuti, ma hai saputo trovare il tuo posto, portando con te una spensieratezza tutta tua.

A Lucky, a quel musino che mi accompagna da anni, sempre pronto a ricevere le coccole e i biscotti.

Crescere Insieme

A Beatrice, la mia compagna di viaggio dalla casetta dell'asilo fino all'età adulta. Insieme abbiamo affrontato e superato debolezze, crisi, traguardi importanti e la vita stessa, potendo sempre contare ciecamente l'una sull'altra. Sei una delle poche persone capaci di farmi dimenticare completamente di essere in mezzo agli altri, come se ci fossimo soltanto noi due, insegnandomi a ubriacarmi di vita. In un mondo in cui ho spesso sentito il bisogno di controllare tutto, tu mi hai ricordato che qualche volta è molto più bello semplicemente vivere.

Ad Anna, Elena e Pamela, il primo gruppetto di amiche, quello nato ai tempi delle medie e che, nonostante gli anni e le strade diverse, continua a rappresentare una parte importante della mia storia. Ad Anna, per avermi sempre spronato a lasciare da parte la timidezza e a uscire dalla mia bolla. A Elena, con la tua dolcezza che ha saputo mantenere intatto il nostro rapporto anche quando la distanza e il tempo hanno reso più rare le occasioni per vederci. Dalle prime guide insieme a oggi, alcune cose sono semplicemente rimaste uguali. E a Pamela, il mio genitore 1, capace di sostenere le mie pazzie senza mai provare davvero a fermarle, salvo poi doverne sopportare le conseguenze.

A Benedetta, per la rara delicatezza con cui sai starmi accanto. Al genitore 2 che vive in te, ma soprattutto alla persona che sei quando riesci ad ascoltare senza giudicare e senza sentire il bisogno di trovare una soluzione a ogni cosa. A volte non servono risposte: serve soltanto qualcuno che rimanga. E tu hai saputo farlo.

A Giulia, alle tinte nei capelli e alle torte comparse nei momenti giusti per tirarmi su di morale, ai futuri giretti in moto e a tutte le volte in cui sei stata una complice perfetta nei nostri piccoli disastri. Sei una di quelle persone con cui anche le situazioni più improbabili finiscono per diventare ricordi da conservare.

Ad Alice e Nicole, le mie inseparabili compagne di banco. Ai nostri mondi profondamente diversi che, invece di allontanarsi, hanno trovato il modo di incontrarsi e costruire qualcosa insieme. Forse è proprio questa la parte più bella di alcune amicizie: non avere bisogno di essere uguali per sentirsi vicini.

Ad Anna, alle corse in autobus in mezzo a una folla di persone, all'oroscopo, al manifesting e a quella risata che riesce sempre a farsi riconoscere. Alle conversazioni nate quasi per caso durante i tragitti e a tutte quelle volte in cui, con la tua spontaneità, sei riuscita a rendere più leggere anche le giornate più ordinarie.

Ad Alessandro, Daniel, Elia e Irene. Alle nostre conversazioni, ai pettegolezzi, alle passioni condivise, al castello di Howl e alle molteplici uscite mai avvenute, ma soprattutto a quel rapporto che, nonostante il tempo e i mesi trascorsi senza vederci, riesce ogni volta a riprendere da dove lo avevamo lasciato, come se in mezzo non fosse passato neanche un giorno.

Ad Alessandra, Alessandro, Chiara, Davide, Marco e Sofia, per le serate di giochi, le uscite, le risate e la possibilità di condividere passioni diverse, senza che fosse necessario essere uguali per stare bene insieme. Ad Alessandro, il mio compagno di covid, di conversazioni in piena notte e di momenti decisamente “spiritati”; a Marco, per quegli abbracci e quelle parole capaci di avvolgere; e a tutti voi, per aver dato leggerezza a tanti momenti di questi anni.

Lungo il Cammino

Ad Alessandro, alla persona con cui, durante la triennale, sognavo di diventare amica e che poi, quasi senza accorgermene, è diventata parte della mia vita. Sei stato quella boccata d’aria di cui avevo bisogno in un mondo di ingegneri che non sempre ci ha rappresentati fino in fondo. Mi hai insegnato che l’università poteva essere anche leggerezza, risate, complicità e qualcosa che andasse oltre gli esami.

A Leonard, al filo rosso che, nonostante le tante occasioni in cui avremmo potuto incontrarci senza mai riuscirci, ha continuato a legare le nostre strade fino a farle incrociare quasi per caso. Alle discussioni un giorno sì e uno no, alla scelta di mettere da parte ciò che era successo e provare davvero a vedere dove potessimo arrivare. E a quella manina che, figurativamente e non, riesce in qualche modo a trovare sempre la mia.

Ad Anna, per i pomeriggi in biblioteca trascorsi a fare gossip, per il supporto reciproco negli stessi problemi di cuore e per essere stata una delle persone che mi ha aiutata maggiormente a crescere a livello personale.

A Caterina, per la capacità di accorgersi che qualcosa non va ancora prima che trovi le parole per raccontarlo, e per essermi rimasta accanto in un momento delicato senza farmi sentire sola. A Sofia, per gli audio-podcast dell’Erasmus, diventati quasi una presenza quotidiana capace di accorciare la distanza e farmi sentire, anche da lontano, un po’ più vicina a casa. E per aver saputo accogliermi nei momenti in cui avevo meno voglia di parlare e più bisogno semplicemente di sentirmi capita.

A Giorgia e Samuele, a quei laboratori che sembravano non finire mai e che, grazie a voi, sono riusciti almeno qualche volta a diventare più leggeri. Alle giornate super stressanti e alle risate che riuscivano a spezzare la tensione quando serviva. A Giorgia co-fondatrice del fan club e a Samuele “girl”, perché ormai questi ruoli sono diventati parte integrante del nostro rapporto. In mezzo a scadenze, esperimenti e stanchezza, aver avuto voi accanto ha fatto decisamente la differenza.

Strade Parallele

Alla ginnastica, a Sonia, Sara e a chi mi ha accompagnata durante gli anni. A quello che è stato molto più di uno sport: una valvola di sfogo, un luogo in cui lasciare fuori dalla porta

le giornate difficili e ritrovare, per qualche ora, un po' di leggerezza. Alle pause "pranzo", alle corse in cortile, al sostegno prima delle esibizioni e delle gare, alle risate e all'immane "szszsz" quando qualche esercizio non veniva esattamente come avrebbe dovuto. Al destino, che anni dopo mi ha fatto rincontrare inaspettatamente Sonia in dolce attesa, dandomi la spinta per iniziare ad allenare le nuove generazioni. Alle bimbe: vedervi mi fa immedesimare in voi, facendomi rivivere le stesse paure e insicurezze che ho sperimentato da piccola. E proprio per questo sento di avere una piccola responsabilità nei vostri confronti: ricordarvi che molte delle paure che sembrano enormi possono essere affrontate e superate, che spesso è anche una questione di mentalità e che avete molte più capacità di quelle che pensate. In fondo, anche Rapunzel ha dovuto affrontare qualche difficoltà prima di poter diventare la principessa che era destinata a essere.

A Klagenfurt e a tutte le persone che ne hanno fatto parte. A un luogo capace di essere, nello stesso momento, paradisiaco e "prigione". Alle partite a biliardo con lo scettro al McMullens, ai pomeriggi passati al lago, alle patatine del signor Klagenfurt e alla multa dell'ultimo giorno. Ma soprattutto alle persone che hanno dato un significato a quei mesi. Perché alla fine sono raramente i luoghi a rimanere davvero: sono le persone, le serate, le situazioni improbabili e le storie che continuiamo a raccontare anche una volta tornati a casa. A quell'Erasmus semplicemente "so la la".

Oltre il Tempo

A chi, in questi anni, mi ha aiutata a crescere. Alla maestra Marisa e alla professoressa Donà, per avermi insegnato a credere un po' di più in me stessa e a essere fiera di ciò che sono, anche quando un fallimento avrebbe potuto convincermi del contrario. A Mouhieddine, per avermi aiutata a mettere da parte la timidezza e a comprendere che anche la mia opinione ha valore e merita di essere ascoltata. A Federico, per avermi mostrato che vulnerabilità e sensibilità non sono debolezze da correggere, ma parti di noi che possono diventare una forma diversa di forza. Ho imparato che non esiste sempre una sola prospettiva possibile e che due visioni differenti possono coesistere.

E infine, a chi mi protegge da lassù. A nonno Romeo, ai viaggi in camion quando ero piccola e a quell'augurio di vedermi raggiungere questi traguardi. Mi piace pensare che, da qualche parte, tu sia ancora qui a guardarmi arrivare fin qui e possa essere fiero della tua "cocca". A nonna Maria, al mio angelo custode, di cui sono sempre più convinta di sentire la presenza. Nonostante a volte facessi fatica a ricordarti le cose quotidiane, il mio nome e il ricordo di me non sembravano mai sfuggirti. E poi quel 99 alla laurea triennale, proprio nei giorni in cui avresti compiuto quegli anni: per tutti un 100 mancato, per me un piccolo segno della tua presenza, arrivato proprio nel momento in cui raggiungevo uno dei miei primi grandi traguardi. Voglio pensare che, in qualche modo, tu fossi lì con me.

Grazie a tutti voi, a chi ha condiviso con me anni interi e a chi ha attraversato soltanto una parte del cammino, a chi mi ha insegnato qualcosa, a chi mi ha fatto ridere, a chi mi ha raccolto nei momenti difficili e a chi, semplicemente, ha reso più bella una giornata qualunque.

References

- [1] Ecem Basak and Çağatay Iris. “Urban life matters: The heterogeneous effects of on-demand bike-sharing platforms on urban public transit”. In: *Transportation Research Part A: Policy and Practice* 203 (2026), p. 104740.
- [2] Wenxiang Li et al. “Exploring the spatial variations of transfer distances between dockless bike-sharing systems and metros”. In: *Journal of Transport Geography* 92 (2021), p. 103032. ISSN: 0966-6923.
- [3] Ganmin Yin et al. “Traffic prediction and road space optimization for the integration of dockless bike-sharing and subway”. In: *Sustainable Cities and Society* 121 (2025), p. 106162.
- [4] Zheyang Chen, Dea van Lierop, and Dick Ettema. “Dockless bike-sharing systems: what are the implications?” In: *Transport Reviews* 40.3 (2020), pp. 333–353.
- [5] Matteo Cederle et al. “A Fairness-Oriented Reinforcement Learning Approach for the Operation and Control of Shared Micromobility Services”. In: *2025 American Control Conference (ACC)*. 2025, pp. 565–570.
- [6] Edoardo Scarpel et al. *Fully Dynamic Rebalancing in Dockless Bike-Sharing Systems via Deep Reinforcement Learning*. 2026. arXiv: 2605.14501 [eess.SY]. URL: <https://arxiv.org/abs/2605.14501>.
- [7] Edoardo Scarpel. *Fully Dynamic Rebalancing of Dockless Bike Sharing Systems using Deep Reinforcement Learning*. Master’s thesis, Università degli studi di Padova. 2024. URL: <https://hdl.handle.net/20.500.12608/84368>.
- [8] Eoin O’Mahony and David Shmoys. “Data analysis and optimization for (citi) bike sharing”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 29. 1. 2015.
- [9] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [10] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3 (1992), pp. 279–292.

- [11] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "A comprehensive survey of multiagent reinforcement learning". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2 (2008), pp. 156–172.
- [12] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. "Artificial neural networks: A tutorial". In: *Computer* 29.3 (1996), pp. 31–44.
- [13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.
- [14] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.
- [15] Vincent François-Lavet et al. "An introduction to deep reinforcement learning". In: *Foundations and Trends in Machine Learning* 11.3-4 (2018), pp. 219–354.
- [16] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.
- [17] Kai Arulkumaran et al. "Deep reinforcement learning: A brief survey". In: *IEEE signal processing magazine* 34.6 (2017), pp. 26–38.
- [18] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.
- [19] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.
- [20] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).
- [21] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems* 12 (1999).
- [22] Claude Elwood Shannon. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.
- [23] Vijay Konda and John Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems* 12 (1999).
- [24] John Schulman et al. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).
- [25] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. Pmlr. 2015, pp. 1889–1897.

- [26] Rui Zheng et al. "Secrets of rlhf in large language models part i: Ppo". In: *arXiv preprint arXiv:2307.04964* (2023).
- [27] Sven Gronauer and Klaus Diepold. "Multi-agent deep reinforcement learning: a survey". In: *Artificial Intelligence Review* 55.2 (2022), pp. 895–943.
- [28] Jakob Foerster et al. "Counterfactual Multi-Agent Policy Gradients". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. Association for the Advancement of Artificial Intelligence (AAAI). 2018.
- [29] Ryan Lowe et al. "Multi-agent actor-critic for mixed cooperative-competitive environments". In: *Advances in neural information processing systems* 30 (2017).
- [30] Chao Yu et al. "The surprising effectiveness of ppo in cooperative multi-agent games". In: *Advances in neural information processing systems* 35 (2022), pp. 24611–24624.
- [31] Chao Yu et al. *The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games*. 2022. arXiv: 2103.01955 [cs.LG]. URL: <https://arxiv.org/abs/2103.01955>.
- [32] Daniel Chemla, Frédéric Meunier, and Roberto Wolfler Calvo. "Bike sharing systems: Solving the static rebalancing problem". In: *Discrete Optimization* 10.2 (2013), pp. 120–146.
- [33] Patrice Leclaire and Florent Couffin. "Method for Static Rebalancing of a Bike Sharing System". In: *IFAC-PapersOnLine* 51.11 (2018), pp. 1561–1566.
- [34] Claudio Contardo, Catherine Morency, and Louis-Martin Rousseau. *Balancing a dynamic public bike-sharing system*. Vol. 4. Cirrelt Montreal, Canada, 2012.
- [35] Adish Singla et al. "Incentivizing users for balancing bike sharing systems". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 29. 1. 2015.
- [36] Jasper Schuijbroek, Robert C Hampshire, and W-J Van Hoeve. "Inventory rebalancing and vehicle routing in bike sharing systems". In: *European Journal of Operational Research* 257.3 (2017), pp. 992–1004.
- [37] Federico Chiariotti et al. "A dynamic approach to rebalancing bike-sharing systems". In: *Sensors* 18.2 (2018), p. 512.
- [38] Ling Pan et al. "A deep reinforcement learning framework for rebalancing dockless bike sharing systems". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 1393–1400.
- [39] Vitória Albuquerque, Miguel Sales Dias, and Fernando Bacao. "Machine learning approaches to bike-sharing systems: A systematic literature review". In: *ISPRS International Journal of Geo-Information* 10.2 (2021), p. 62.

- [40] Ammar Haydari and Yasin Yılmaz. “Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.1 (2022), pp. 11–32.
- [41] Syed Mohib Raza, Mohammad Sajid, and Jagendra Singh. “Vehicle routing problem using reinforcement learning: recent advancements”. In: *Advanced machine intelligence and signal processing*. Springer, 2022, pp. 269–280.
- [42] Yao Jing et al. “Efficient Bike-sharing Repositioning with Cooperative Multi-Agent Deep Reinforcement Learning”. In: *ACM Trans. Sen. Netw.* (2024). ISSN: 1550-4859.
- [43] Young-Hyun Seo et al. “Rebalancing docked bicycle sharing system with approximate dynamic programming and reinforcement learning”. In: *Journal of Advanced Transportation* 2022.1 (2022), p. 2780711.
- [44] Supriyo Ghosh et al. “Dynamic repositioning to reduce lost demand in bike sharing systems”. In: *Journal of Artificial Intelligence Research* 58 (2017), pp. 387–430.
- [45] Yexin Li, Yu Zheng, and Qiang Yang. “Dynamic bike reposition: A spatio-temporal reinforcement learning approach”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 1724–1733.
- [46] Xu Geng et al. “Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 3656–3663.
- [47] Jiaqi Liang et al. *A reinforcement learning approach for dynamic rebalancing in bike-sharing systems*. Tech. rep. Bureau de Montreal, Université de Montreal, 2024.
- [48] Xinghua Li et al. “Dynamic Repositioning in Dock-less Bike-sharing System: A Multi-agent Reinforcement Learning Approach”. In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. 2022, pp. 170–175.
- [49] Citibike. *System Data*. URL: <https://citibikenyc.com/system-data>.
- [50] Arturo Gonzalez-Escribano, Diego R Llanos, and Hector Ortega-Arranz. *The shortest-path problem: Analysis and comparison of methods*. Springer Nature, 2022.