smart cities

MDPI

Article

Robust Reinforcement Learning Strategies with Evolving Curriculum for Efficient Bus Operations in Smart Cities

Yuhan Tang [1], Ao Qu [1,*], Xuan Jiang [2], Baichuan Mo [1], Shangqing Cao [2], Joseph Rodriguez [3], Haris N Koutsopoulos [3], Cathy Wu [1] and Jinhua Zhao [4]

[1] Department of Civil and Environmental Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA; yhtang@mit.edu (Y.T.); baichuan@mit.edu (B.M); cathywu@mit.edu (C.W.)
[2] Department of Civil and Environmental Engineering, University of California at Berkeley, Berkeley, CA 94720, USA; xuanjiang@berkeley.edu (X.J.); caoalbert@berkeley.edu (S.C.)
[3] Department of Civil and Environmental Engineering, Northeastern University, Boston, MA 02115, USA; rodriguez.josep@northeastern.edu (J.R.); h.koutsopoulos@northeastern.edu (H.K.)
[4] Department of Urban Studies and Planning, Massachusetts Institute of Technology, Cambridge, MA 02139, USA; jinhua@mit.edu
* Correspondence: qua@mit.edu

Highlights:

What are the main findings?

- A novel RL framework is proposed for bus operations, integrating three high-level actions: holding, skipping, and turning around to reduce passenger waiting times.
- The model incorporates LSTM to capture both Markov and non-Markov processes, enhancing decision-making based on past actions and future predictions.
- Domain Randomization is used to improve the model's robustness against unpredictable traffic conditions and passenger demand.
- Curriculum learning allows the RL agent to efficiently learn complex action spaces, demonstrating improvements over traditional holding-only strategies.

What are the implications of the main finding?

- The proposed RL framework demonstrates that reinforcement learning can effectively handle the complexity of bus operations with curriclumn learning integrated.
- The integration of domain randomization within RL enhances the model's adaptability to real-world variations, offering a more robust solution for public transit systems.
- The combination of holding, skipping, and turning around strategies in the RL model provides greater flexibility and efficiency in bus operations, significantly outperforming traditional single-strategy approaches.
- This approach highlights the potential of RL to provide scalable and efficient operation in multi-agent environments, paving the way for smarter and more adaptive urban transportation systems.

Abstract: Public transit systems are critical to the quality of urban life, and enhancing their efficiency is essential for building cost-effective and sustainable smart cities. Historically, researchers sought reinforcement learning (RL) applications to mitigate bus bunching issues with holding strategies. Nonetheless, these attempts often led to oversimplifications and misalignment with the goal of reducing the total time passengers spent in the system, resulting in less robust or non-optimal solutions. In this study, we introduce a novel setting where each bus, supervised by an RL agent, can appropriately form aggregated policies from three strategies (holding, skipping station, and turning around to serve the opposite direction). It's difficult to learn them all together, due to learning complexity, we employ domain knowledge and develop a gradually expanding action space curriculum, enabling agents to learn these strategies incrementally. We incorporate Long Short-Term Memory (LSTM) in our model considering the temporal interrelation among these actions. To address

check for updates

Citation: Tang, Y.; Qu, A.; Jiang, X.; Mo, B.; Cao, S.; Rodriguez, J.; Koutsopoulos, H.; Wu, C.; Zhao, J. Robust Reinforcement Learning Strategies with Evolving Curriculum for Efficient Bus Operations in Smart Cities. *Smart Cities* **2024**, *7*, 3658–3677. https://doi.org/10.3390/smartcities7060141

Academic Editor: Pierluigi Siano

Received: 28 September 2024
Revised: 12 November 2024
Accepted: 21 November 2024
Published: 29 November 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

the inherent uncertainties of real-world traffic systems, we impose Domain Randomization (DR) on variables such as passenger demand and bus schedules. We conduct extensive numerical experiments with the integration of synthetic and real-world data to evaluate our model. Our methodology proves effective, enhancing bus schedule reliability and reducing total passenger waiting time by over 15%, thereby improving bus operation efficiency and smoothering operations of buses that align with sustainable goals. This work highlights the potential of robust RL combined with curriculum learning for optimizing public transport in smart cities, offering a scalable solution for real-world multi-agent systems.

Keywords: bus oeration; deep reinforcement learning; intelligent transportation systems; curriculum learning; smart cities

1. Introduction

Public transit systems, particularly bus services, are foundational to urban infrastructure and human mobility. The precise and timely operation of these systems is a cornerstone of urban mobility. Historically, reinforcement learning (RL) has been sought after to optimize bus schedules, predominantly addressing the persistent challenge of bus bunching [1]. However, traditional RL methods often oversimplify the problem, leading to solutions that may not adequately minimize passenger waiting times or optimize bus schedules and the overall system performance [2]. These challenges arise from several inherent complexities:

First, bus operations in urban areas face numerous unpredictable variables, from sudden surges in passenger demand to unforeseen traffic congestion, which traditional RL models struggle to accommodate. Second, the temporal dynamics between bus stops, intertwined with factors like varying passenger demands at different times of the day and changing traffic conditions, require models with memory capabilities to make informed decisions. Lastly, the vast and diverse nature of real-world traffic data, with its inherent uncertainties, poses a challenge for any model to generalize effectively.

Recent attempts to address these issues lean on over-simplified strategies. For instance, some models resort to merely holding buses at stations or optimizing routes without taking into account the dynamic nature of urban traffic and passenger demand.

To address these challenges, we introduce a comprehensive approach in this paper. Our methodology begins with the deployment of a robust RL model that has a Long Short-Term Memory (LSTM) network integrated. The incorporation of LSTM allows our model to account for the temporal dynamics of bus operations, enabling it to make decisions based on historical and real-time data. We also employ Domain Randomization (DR) to introduce variability and robustness into our model, preparing it for the uncertainties of real-world operations.

In summary, the contributions of this paper are:

- We introduce an LSTM-integrated RL model tailored for improving bus operations to capture both Markov and non-Markov processes.
- We implement DR to enhance the robustness of our model, ensuring its effectiveness amidst the uncertainties of urban traffic and dynamic passenger demands.
- We implement Curriculum Learning with masked LSTM to learn the bus operation strategies step by step from holding and skipping to turning around and combining all three controls.
- Our approach not only leads to a significant reduction in passenger waiting times but also offers a more holistic optimization of bus schedules, ensuring timely and efficient urban mobility.

2. Related Works

2.1. Bus Bunching Problem

Bus bunching, the phenomenon where two or more buses arrive at a stop simultaneously, occurs when the headway between buses is not preserved as defined by the schedule. Many attempts have been made in the past to address bus bunching by implementing control policies that preserve the desired headway. Daganzo explores an adaptive control scheme that adjusts the holding time of buses at different control points to maintain headway [3]. The method requires a smaller slack compared to the conventional strategy that introduces a large slack in the schedule to make bus bunching become unlikely. This line of strategy is further explored by Andres et al. who improve the control scheme by adding a predictive model for future headway given a series of headway [4]. Sánchez-Martínez et al. set up an optimization model that computes the holding times of all buses at all control stops with dynamic running time and demand [5]. In a separate paper, Daganzo and Pilachowski formulate the desired cruise speed, which is the stochastic equilibrium cruise speed, as a function of the spacing between buses [6]. Thus, a bus can slow down or accelerate to meet the desired cruise speed, with which bus bunching can be avoided. Bartholdi et al. introduce the idea of "self-equalizing" headway in that the headway between buses is not bounded by a predefined value [7]. Instead, headway can reach equilibrium given that it is confined above a certain threshold, achieved by holding at control points [7]. Estrada et al. also consider the possibility of extending the green light phase at certain intersections to adjust the headway between buses [8]. Lastly, Wu et al. consider both passing and distributed passenger boarding as alternatives to control the headway dynamically [9].

2.2. Traditional Method in Bus Operations

Traditionally, agencies use holding control in bus operations. Ideally, bus operation is defined by the accumulation of bus arrival times at succeeding control points [10]. The set of the control points $\{t_{n,s}\}$ is defined by:

$$t_{n,s+1} = t_{0,0} + nH + \sum_{i=0}^{s} p_i, \quad n, s = 0, 1, 2 \ldots \tag{1}$$

where the buses, n, are treated as "agents" and the control points, s, as "time". The actual travel time $a_{n,s}$ represents the state of the agents. p_i represents the planned travel time or planned inter-arrival time between successive control points on a bus route. It accounts for the time a bus is expected to take to travel from one control point to the next, incorporating scheduled stops and typical traffic conditions. The uncontrolled travel time $u_{n,s}$ for bus n between stops s and $s+1$ is:

$$\begin{aligned} u_{n,s} \equiv a_{n,s+1} - a_{n,s} &= c_s + \beta_s(h_{n,s} - H) + \gamma_{n,s} \\ &= c_s + \beta_s\left(a_{n,s} - a_{n-1,s} - H\right) + \gamma_{n,s} \end{aligned} \tag{2}$$

where $a_{n,s}$ represent the arrival time of n^{th} bus at control point s; $h_{n,s} = a_{n,s} - a_{n-1,s}$ represents the headway ahead of bus n at the control point s; H is the actual headway; c_s represents the travel time from s to $s+1$ assuming all buses are on-time; β_s is a constant representing the average increase in bus arrival time from s to $s+1$ due to a unit increase in headway time. $\gamma_{n,s}$ represents a stochastic term that accounts for unpredictable variations in the travel time of bus n between stops s and $s+1$.

In traditional bus operations, there are two systems—Frequency-Based Systems and Scheduled Systems.

In frequency-based bus systems, buses run at regular intervals throughout the day. This approach is generally preferred in densely populated urban areas where there is high

demand for public transit. Passengers find it convenient as they don't need to check the schedule; however, it can easily cause bus bunching because buses tend to be too close to each other. To address this, when operating, each bus may hold at a stop based on the delay of the previous bus. The modified $a_{n,s+1}$ can be calculated by the following equation:

$$a_{n,s+1} = a_{n,s} + c_s + D_{n,s} + \beta_s(h_{n,s} - H) + \gamma_{n,s} \tag{3}$$

$$D_{n,s} = \left[d_s - (\beta_s + \alpha)(h_{n,s} - H)\right]^+ \tag{4}$$

where α is a weighting parameter. However, such a control strategy passes down the delay. If bus n delays, then bus $n+1$ also has to delay due to the extra holding time $D_{n+1,s}$. The delay of the $(n+1)$-th bus also causes extra holding time $D_{n+2,s}$, resulting in the delay of the $n+2$-th bus. This will create a vicious cycle, leading to delays for all following buses due to the delay of the n-th bus, as is shown in Figure 1. Also, previous studies focus on alleviating bus bunching while often ignoring the overall system performance.

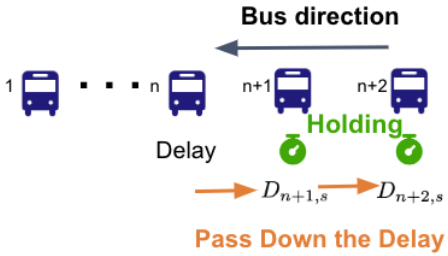

Figure 1. Delay Passing Down Caused by Holding.

On the other hand, scheduled bus systems adhere to a stringent timetable, with vehicle departures occurring at specific, pre-established times. Although scheduled bus systems do not pass down the delay, buses can get bunched if the delay is greater than the headway, which often happens in urban areas. The disadvantages of these two bus control systems inspire our work of using combined strategies to control the whole bus system actively.

2.3. Reinforcement Learning Application in Bus Operations

Due to the recent advance in Deep Reinforcement Learning (DRL), which can effectively model the feedback of various control strategies over a long time horizon, DRL appears as a promising strategy for improving bus operations [11]. Many works have explored building a bus operation policy that preserves the headway among buses. Chen et al. develop a multi-agent reinforcement learning (MARL) scheme to find the optimal holding time at stops from a discretized action space that minimizes the deviation from the scheduled headway [12]. Wang et al. adopt a multi-agent deep reinforcement learning (MDRL) scheme to find the optimal holding time for each bus at each stop based on the amount of variance in the headway the action leads to, which is the reward [13]. Such an approach allows the action imposed on each individual bus to interact with the entire system, not only with the leading and the trailing buses. Alesiani et al. introduce a reward function that penalizes solutions with long travel times and long holding times [14]. Tham et al. construct an action space in which a bus can choose to skip zero, one, or two stations to improve bus operation [15]. Poliziani et al. use the simulator BEAM CORE to evaluating transit enhancements [16]. Rodriguez et al. implemented a cooperative strategy which combines holding and stop-skipping together in bus operation [17]. Still, no study considers the overall costs incurred by the passengers and lacks a mixed-strategy action space that uses holding, turning around, and skipping as potential solutions.

2.4. Curriculum Learning

Curriculum learning is a popular and effective method for improving RL agent performance and efficiency in a variety of environments by guiding agent actions. Several studies have investigated the impact of task sequencing on the learning performance and sample efficiency of RL agents. In general, many propose a curriculum learning framework that intelligently orders tasks to gradually increase in difficulty. By strategically sequencing tasks, agents were shown to not only achieve faster convergence and higher reward states, but improved generalization across a range of environments [18]. Traffic and transportation optimization is also a commonly examined environment for evaluating all types of reinforcement learning algorithms. Other attempts, such as the ones proposed by [19,20], use curriculum learning to improve upon learning efficiency in various frameworks, however not explicitly in the bus operation setting. Additionally, others have examined curriculum learning discretely through the lens of an "adversary", but use a deterministic algorithm rather than a model to define the adversary's actions throughout training.

3. The Framework of the Training Environment

SimPy facilitates discrete event simulations, where the behavior of agents can be modeled as processes that interact with each other and with the environment and learned by RL model [21]. Therefore, we use Simpy to formulate the bus line corridor model with m uniformly distributed stations, $s_j, j = 1, 2, ..., m$. There are n buses, $b_i, i = 1, 2, ..., n$, driving on this route with a constant average speed v. The general setup of the framework is shown in Figure 2. The extended times, which may occur when skipping or turning around actions are done, are included when setting the actions in the environment. Here are the assumptions:

- The route is a simple loop with stations evenly distributed along [13].
- When there is bus bunching at station s_j, the probability for the passengers to board each bus is equal.
- Boarding and alighting occur concurrently and adhere to a proportional pattern based on passenger counts. Each passenger's time to board and exit is consistently τ_b and τ_a minutes at every stop. The bus must wait for both processes to complete before it can depart, with the longer of the two processes dictating the overall stop duration at a certain station.
- At bus stop s_j, the arrival of passengers is described by a Poisson distribution with a rate of λ_j. For a passenger at stop s_j, their destination is selected randomly and with equal chance from the next $\lfloor (m - j)/2 \rfloor$ stops.
- There is a capacity C for each bus.

As is shown in Figure 2, here's the general setup of the framework:

- The delay of bus n incur at bus stop s and between stops s and $s + 1$ caused by traffic or other factors is denoted by $D_{n,s}$.
- The bus system is a frequency-based system. All the buses are initialized at station s_j with a constant interval of headway H.
- If a bus decides at station s_j to skip the next station s_{j+1}, it must alight all the passengers whose destinations are s_{j+1} at station s_j and make the next stop at station s_{j+2}.
- If a bus decides to turn around at station $s_j, j \leq \lfloor m/2 \rfloor$. It must alight all the passengers whose destinations are $s_k, k = j + 1, ..., \lfloor m/2 \rfloor$. The symbol $\lfloor \cdot \rfloor$ denotes the floor function, which rounds down to the nearest integer.

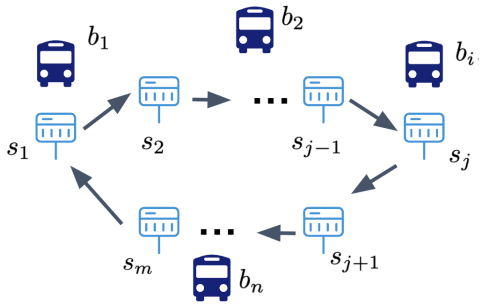

Figure 2. Representationof the Bus Line simulated in Simpy.

In our training environment, 5 passenger statuses are defined as shown in Table 1. Passengers might be waiting at a station, get on a bus and alighted. When alighted, he might still be waiting since the passenger hasn't arrived or he might arrive and leave the system. As a penalty, when waiting time is longer than the headway, passengers will have a 50% possibility of leaving the station as a penalty for a long wait.

Table 1. Passenger Status.

Status	Meaning	In or Out of the System
0	Waiting	Passengers are in the transit system
1	On bus	
2	Waiting after Alighted	
3	Arrived	Passengers are out of the transit system
4	Left after a long wait	

4. Methodology

4.1. A PPO Algorithm to Improve the Efficiency of Bus Operations

4.1.1. Action and Agent

In bus operation, every bus is an agent. In our paper, we use three actions based on experience: holding, skipping, and turning around. Holding means buses will wait at one stop. Skipping means skipping a stop. Turning around means alighting all passengers and serving the other direction. Each bus (agent) makes an action at the station, interacts with the environment, and gets the reward as shown in Figure 3.

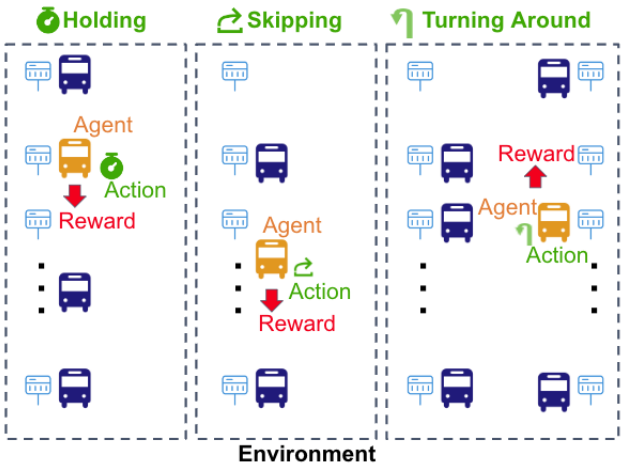

Figure 3. The Training Environment.

Holding strategy is the most popular way since it is easier to implement with formed equations [3]. On the other hand, Daganzo, et al. [10] propose the skipping and turning around strategy, but without giving equations. Since RL can perform complex decision-making and balance exploration and exploitation after learning the policies, our paper builds our bus operation model based on RL.

4.1.2. Observation

At time step t, a bus agent b_i can obtain partial information about the transport environment as its current observation $o^t_{b_i}$. In this bus route simulation environment, a single observation consists of the following components: bus location, passenger count on the bus, and action status. Specifically, the location is represented by the station index, transformed into a continuous scale based on the distance traveled from the last station proportional to the travel time, which can be periodically reset at each full loop around the stations. This continuous scale helps capture the exact positioning between stations, important for modeling the dynamics of movement and timing.

The number of passengers currently on the bus is a direct integer count, reflecting the current load and influencing decisions related to stopping, skipping, or turning around. The actions undertaken by the bus are encoded in terms of the most recent action (holding, skipping, turning around). The action is encoded as a variable, while the duration since the last action started provides a temporal depth to the observation, capturing the time aspect of the bus's recent operations.

Thus, the complete observation for a bus agent b_i at time t can be expressed as $o^t_i = [o^t_{i,L}, o^t_{i,N_{pax}}, o^t_{i,H}, o^t_{i,T}, o^t_{i,Ego}]$, referring to location $o^t_{b_i,L}$, passenger count on the bus $o^t_{i,N_{pax}}$, headway $o^t_{i,H}$, and time $o^t_{i,T}$ respectively. In our multi-agent RL, only one agent takes an action at the same time. Ego $o^t_{i,Ego}$ means the current bus to take action.

These components act as the dynamics of a bus system vary with time due to differing passenger loads throughout the day and also vary spatially as buses move through different segments of their routes which may have varying degrees of passenger demand. These observations allow the reinforcement learning model to adapt its policy, aiming to optimize the stated objectives of the system.

4.1.3. Reward

The total reward is defined as $R = -(\alpha \frac{T_w}{N_w} + \beta \frac{T_b}{N_b})$, where T_w refers to the total passenger waiting time, T_b refers to the total passenger time on bus, N_w refers to the total number of waiting passengers, N_b refers to the total number of passengers on bus, and $\alpha, \beta \in \mathbb{R}$ are the weighting parameters. In the following section, we use 10^6 seconds as the unit of reward.

4.1.4. Actor-Critic Learning

In order to maximize the discounted cumulative reward in the long term, to keep the training procedure more stable, and to increase learning efficiency, we use proximal policy optimization (PPO) proposed by Schulman et al. [22], a type of actor-critic method designed to improve the stability and reliability of policy gradient methods by optimizing a surrogate objective function. The design of the actor-critic framework is shown in Figure 4. In order to avoid larger parameter updates and big policy ratios, $r(\theta)$ is forced to stay within a small interval $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter. The objective function is:

$$J^{\text{PPO}}(\theta) = \mathbb{E}\left[\min\left(r(\theta)\hat{A}_{\theta_{\text{old}}}, \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}}\right)\right] \tag{5}$$

θ is the set of current parameters of the policy network and it defines the current policy that maps states to actions. $r(\theta)$ is the probability ratio between the current policy and the previous policy for a given action. $\hat{A}_{\theta_{\text{old}}}$ is the advantage function, calculated based on the previous policy θ_{old}. The advantage function $\hat{A}_{\theta_{\text{old}}}(s,a)$ estimates how much better or worse a specific action a is, compared to the average action in a particular state s. It serves as a way to encourage actions that are expected to yield higher returns. ϵ is a hyperparameter that controls the range within which the probability ratio $r(\theta)$ is allowed to vary without penalty. The operation $\text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)$ constrains the ratio such that it does not exceed $1+\epsilon$ and does not fall below $1-\epsilon$. The PPO's objective function selects the lower value of the two between the unadjusted value and this clipped ratio. As a result, this diminishes the incentive to push policy updates to extreme values in pursuit of higher rewards.

The actor network, defined for each bus b_i, is given by:

$$\text{actor}_{b_i}\left(o_{b_i}^t\right) = \sigma\left(\text{MLP}\left(o_{b_i}^t\right)\right),$$

where σ is the softmax function ensuring that the outputs can be interpreted as probabilities for each possible action. MLP stands for Multilayer Perceptron, used to process the input observations and produce an output that represents potential actions for each bus b_i. This probability distribution over actions allows the agent to explore the action space effectively while exploiting learned behaviors that lead to higher rewards.

The critic network is structured similarly but with a crucial difference in its output layer:

$$\text{critic}_{b_i}\left(o_{b_i}^t\right) = \text{MLP}\left(o_{b_i}^t\right),$$

which outputs a single value. This value represents the critic's estimate of the value function $V^{\pi}(s_{b_i}^t)$, where $s_{b_i}^t$ is the state of the bus at time t. The value function approximates the expected return from state $s_{b_i}^t$ following the policy π dictated by the actor.

Both networks share parameters through most of their architecture up to the final layer, which is designed to minimize both computational overhead and the number of parameters needed to be learned in order to simplify the learning process and speeding up convergence. The shared layers allow the networks to benefit from common feature representations learned from the environment, which capture the complexities of the bus operation, including the dynamics of passenger flow and the timing of stops.

The whole Actor-Critic Framework for bus environment is shown in Figure 4. The Bus Environment provides observations that form the state (S_t) at each time step t, representing relevant bus operational information, including Location ($O_{i,L}^t$), the Number of Passengers $\left(O_{i,Npax}^t\right)$, Headway ($O_{i,H}^t$), Time $\left(O_{i,T}^t\right)$, and an Ego feature $\left(O_{i,Ego}^t\right)$ to capture characteristics specific to the main bus being analyzed. These features are tracked for each bus, b_1 through b_n, creating a dynamic representation of the transportation system's state. The Actor-Critic PPO model consists of an Actor Network, which outputs a policy $\pi_\theta(a_t \mid s_t)$ defining the probability of selecting action a_t in state s_t, and a Critic Network that evaluates this state through a value function $V_\mu(s_t)$. The model leverages Sample Memory to store between the new and old policies to maintain stable updates. The Advantage Function $\widehat{A_t}$ uses the Temporal Difference error δ_t to indicate how much better or worse the taken action was compared to the Critic's baseline value. Value Loss $\left(L^V(\mu)\right)$ is the loss function for the critic network, minimizing the difference between the predicted value $V_\mu(s_t)$ and the actual return; PPO Clipped Loss $\left(L^{CLIP}(\theta)\right)$ is the clipped objective function for the actor network in PPO, designed to avoid excessive updates to the policy. To enhance policy stability, we use $L^{CLIP}(\theta)$ to clip the ratio within a predefined range, constraining updates

to the policy π_θ and Value Loss $L^V(\mu)$ for the critic. An LSTM layer with 256 hidden units captures temporal dependencies by using recurrent feedback, enabling the policy to account for patterns over time. The model is trained by optimizing $L^{CLIP}(\theta)$ and $L^V(\mu)$ using gradient descent, resulting in adaptive bus management decisions that consider both the current state and sequential patterns in operational data.

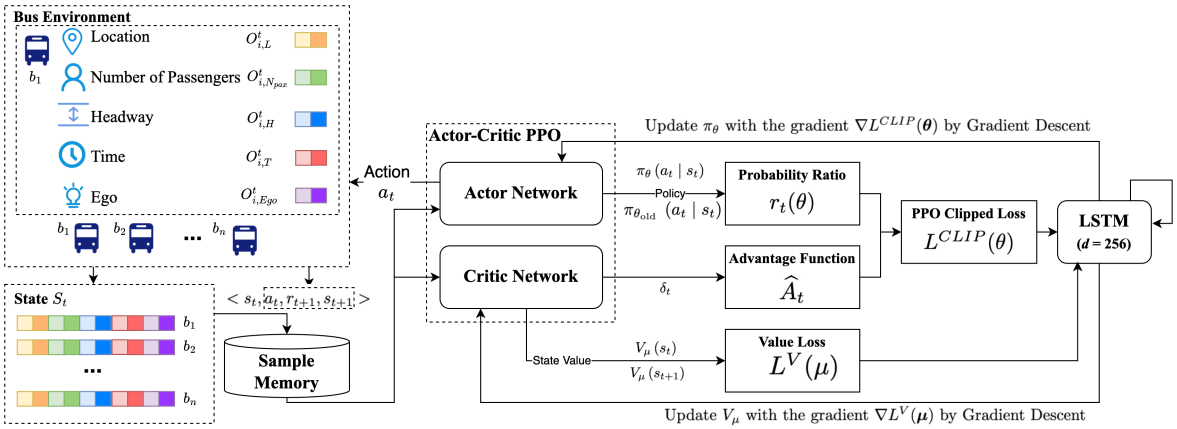

Figure 4. The Actor-Critic Framework for PPO Algorithm.

4.1.5. Action Space

Both discrete and combined action spaces are feasible for bus operation. On continuous action spaces, standard PPO is unstable when rewards vanish outside bounded support [22]. Since combined action spaces include continuous action spaces, we choose discrete action spaces for RL in our model. The action value and action spaces for an environment developed using the Gym library are shown in Tables 2 and 3. Gym is a library in python to support RL. It provides a standardized interface for creating and managing RL environments, including support for discrete and continuous action spaces. In Table 3, the High-level Action column represents discrete options that guide broader strategies (e.g., "Holding", "Skipping", or "Turning Around"), while the Low-level Action column specifies finer control within these strategies. The Box configuration allows for a continuous range of actions within specified bounds, supporting more nuanced decisions. For instance, "Holding" includes both a discrete high-level decision and a continuous low-level parameter within the 0 to 1 range. On the other hand, the Discrete Action Space column provides a simplified, finite set of actions represented as Discrete (N), where N is the total number of possible actions. Each strategy has corresponding discrete action representations.

The action space for holding is discretized into an interval of 10 s (For example, an action value of 5 means an action of holding for 50 s.) Discrete action space introduces simplicity to the problem-solving process by transforming a combined action space into a manageable set of discrete options for the agent to select from, thereby making the learning algorithm more tractable. Furthermore, combined action spaces are susceptible to stability challenges during training, since minimal alterations in policy could precipitate substantial changes in action outcomes; discretization mitigates this issue, enhancing stability. Moreover, discretization of the action space can significantly reduce the computational resources required for policy evaluation and improvement, which is beneficial for real-world applications where quick decisions may be necessary.

Table 2. Action Value

	Combined Action Space		Discrete Action Space
	High-Level Action	**Low-Level Action**	**Discrete Action**
Holding	0	Holding time	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
Skipping	1	/	12
Turning Around	2	/	13

Table 3. Action Space

	Combined Action Space		Discrete Action Space
	High-Level Action	**Low-Level Action**	**Discrete Action**
Holding	Discrete (2)	Box (low = 0, high = 1, shape = (1,))	Discrete (12)
Skipping	Discrete (2)	/	Discrete (2)
Turning Around	Discrete (2)	/	Discrete (2)
Combined Strategies	Discrete (3)	Box (low = 0, high = 1, shape = (1,))	Discrete (14)

Note: - 'Box' specifies the continuous numeric range of valid action values for each dimension of the action space in gym library. - 'Discrete' a finite set of discrete actions.

4.2. LSTM

The bus's previous action should affect its next action. Whether the bus ahead is holding might also affect this bus's action. Therefore, agents need to memorize and analyze the actions taken. The inherent unpredictability of bus schedules, characterized by variations in passenger arrivals and unforeseen delays, requires an approach that can effectively capture and leverage sequential dependencies. Long Short-Term Memory (LSTM) networks, renowned for their ability to model and process sequential data [23], serve as a fundamental cornerstone in achieving this goal.

Mathematically, the LSTM cell employs a set of gating mechanisms to manage the flow of information across time steps.x_t represents the input embedding for the current time step t, generated by the Transformer module and fed into the LSTM. Each x_t is derived from the Transformer's output, allowing the LSTM to leverage the Transformer's encoded sequence representations. This LSTM cell is used to update the hidden state h_t and cell state c_t in a way that accounts for the current input data x_t, the previous hidden state h_{t-1}, and the previous cell state c_{t-1}. The gating mechanisms, which include the forget gate f_t, input gate i_t, and output gate o_t, govern the retention, updating, and discard of information at each time step to address the intricacies of bus operations. The LSTM cell's internal computations are represented by the following equations:

$$f_t = \sigma\Big(W_f \cdot [h_{t-1}, x_t] + b_f\Big) \tag{6}$$

$$i_t = \sigma\big(W_i \cdot [h_{t-1}, x_t] + b_i\big) \tag{7}$$

$$\tilde{c}_t = \tanh\big(W_c \cdot [h_{t-1}x_t] + b_c\big) \tag{8}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{9}$$

$$o_t = \sigma\big(W_o \cdot [h_{t-1}x_t] + b_o\big) \tag{10}$$

$$h_t = o_t \odot \tanh(c_t) \tag{11}$$

In these equations, σ represents the sigmoid activation function, $\odot$ denotes element-wise multiplication, and tanh signifies the hyperbolic tangent function. The weight matrices W_f, W_i, W_c, and W_o along with bias vectors b_f, b_i, b_c, and b_o are learned during training. By updating the hidden state and cell state using this LSTM architecture, the RL agent can effectively capture the sequential patterns and dependencies present in the bus operation environment, ultimately aiding in making informed decisions for optimized management.

The process of the whole implementation is shown in Algorithm 1. The algorithm first initializes variables, including the Bus Environment e, the RL policy π, an experience replay memory D, and the LSTM hidden and cell states h and c. The LSTM function handles initializing LSTM units, training episodes, and updates to the agent's policy π. The inner loop runs for each time step within an episode, where the environment state s_t is observed, an action a_t is selected based on the current policy and LSTM states, and a transition (s_t, a_t, r_t, s_{t+1}) is stored in the episode memory M. Each transition includes the current state, action, reward r_t, and the next state s_{t+1}, all of which crucial for updating the agent. The LSTM state (h, c) is also updated at each time step, enabling the network to incorporate sequential information. After each episode, the episode memory M is added to the replay memory D, which is then sampled to update the policy using PPO. This structured approach allows the RL agent to learn a time-dependent policy for improved bus operation, leveraging both episodic experience and sequential data handling through LSTM. When implementing the LSTM algorithm, we set $LSTM\_Units$ to 256 and get π, D from RL agent.

Algorithm 1 Bus Operation RL Training with LSTM

Input: Bus Environment e, Hyperparameters, LSTM Units
Output: Trained RL Policy π

Initialize LSTM network with $LSTM\_Units$ units.
Initialize RL agent with policy π.
Initialize experience replay memory D.
Initialize LSTM hidden state h and cell state c.
function LSTM($LSTM\_Units$, π, D, $batch\_size$)
    for episode = 1 to max_episodes **do**
        Reset bus environment e to initial state.
        Reset LSTM state: $(h, c) =$ LSTM_Reset().
        Initialize episode memory M.
        for time step $t = 1$ to max_steps **do**
            Get bus state s_t from environment e.
            Select action a_t using policy π and LSTM state (h, c).
            Execute action a_t in environment e, observe next state s_{t+1} and reward r_t.
            Store transition (s_t, a_t, r_t, s_{t+1}) in episode memory M.
            Update LSTM state (h, c) based on (h, c, a_t).
            if episode is done **then**
                Add episode memory M to experience replay memory D.
                Break from time step loop.
        if len$(D) \geq$ batch_size **then**
            Sample a batch of transitions from D.
            Update the policy π using PPO with LSTM states.
    return trained policy π
LSTM($LSTM\_Units$, π, D, $batch\_size$)

4.3. Domain Randomization

In the context of deploying bus operations, it is imperative for the RL algorithm to be resilient to the unpredictable nature of real-world bus schedules. Variabilities such as unexpected delays or sudden surges in passenger arrivals can disrupt the optimal

functioning of bus operations, leading to issues like bus bunching. Therefore, the RL algorithm must be robust to the variability in delays. To close the "Sim to Real" gap, we implemented Domain Randomization (DR) in our training regimen.

The primary motivation behind DR is to introduce variability in the training environment, enabling the RL agent to generalize better when faced with unforeseen circumstances in real-world scenarios. Using DR, a range of simulated environments can be generated with diverse randomized attributes and subsequently train a model that is effective in all of these environments. This model is likely to possess adaptability to the real-world setting, given that the actual system is anticipated to be just one realization of the distribution of training variations.

In DR, the training environment is the source domain that we can have full access to, while the real-world bus operation is the target domain that we want to transfer to. The model is trained in the source domain. A set of N randomization parameters can be controlled in the source domain e_{ξ}, where the configuration ξ is sampled from a randomization space.

With DR, discrepancies between the source and target domains are modeled as variability in the source domain [24]. Throughout the process of policy training, episodes are gathered from the source domain e_{ξ} utilizing randomization. As a result, the policy becomes acquainted with a diverse array of environments and acquires the ability to generalize effectively. R is the reward defined in Section 4.1.3. The policy parameter θ is refined to maximize the average expected reward R across a distribution of configurations:

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\xi \sim \Xi}\Big[\mathbb{E}_{\pi_\theta, \tau \sim e_{\xi}}[R(\tau)]\Big] \tag{12}$$

In (12), we model discrepancies between the source and target domains by introducing variability in the source domain, denoted as e_{ξ}. The symbol e_{ξ} represents the source environment with randomized parameters ξ, which introduce diverse configurations to simulate domain discrepancies. The variable ξ is drawn from a distribution Ξ, capturing different configurations of the source environment. Through policy training, episodes are gathered from the source domain e_{ξ}, where π_θ denotes the policy with parameters θ. The parameter θ is optimized to maximize the policy's performance across various configurations by maximizing the expected reward R, which is defined in Section 4.1.3. The symbol $R(\tau)$ represents the reward for a trajectory τ, which is a sequence of states and actions collected in the source environment e_{ξ} under the policy π_θ. Finally, θ^* denotes the optimal policy parameters that maximize the average expected reward across the distribution of source domain configurations. We then implement DR on passenger arrival rate and bus schedule.

Sánchez-Martínez, et al. set the domain of the adjustment factor from 0 to 2 [5]. They chose arrival rates that yield a peak bus load of 75% of capacity in high-crowding cases and 25% in low-crowding. Similarly, we introduce three distinct levels of passenger arrival rates. Without DR, the passenger arrival rate is set as λ. The demand levels are shown by the parameters l_1, l_2, l_3. Let $L = \{l_1, l_2, l_3\}$. We let $l_1 = 1.25$ in high-crowding cases, $l_2 = 1.0$ in ordinary cases, and $l_3 = 0.75$ in low-crowding cases. Let passenger arrival rate with DR be λ^*. Demand levels l_1, l_2, l_3 are randomly selected, so $\lambda^* = rand(L)\lambda$. This variability ensures that the RL agent is well-equipped to handle both peak and off-peak scenarios, as well as any unexpected surges in passenger demand.

To further introduce randomness, Gaussian noise is added to the demand levels l_1, l_2, l_3. To ensure that these randomized demand levels remain realistic and within predefined bounds, they are clipped to lie within the minimum and maximum values of the initial demand levels array L. The resulting randomized demand levels are then used to compute the passenger arrival rate, which subsequently influences the generation of the passenger arrival times. This DR approach ensures that the generated passenger arrival

tables encompass a broad range of possible scenarios, aiding in creating a more generalized and adaptable model. The process of generating passenger arrival is shown in Algorithm 2.

Algorithm 2 DR

Input: $N_{Station}$, HORIZON, N_{Pax}
Parameter: Demand Level, Seed
Output: Passenger Arrival Table

Let $DemandLevel\Theta = [0.5, 0.75, 1.0]$.
Initialize the random seed.
Generate a random array $RandomDemandLevels$, denoted as $\hat{\Theta} = [X_1, X_2, \ldots, X_{N_{Station}}] \in \mathbb{R}^{N_{Station}}$, with X_i randomly chosen from Θ for $i = 1, 2, \ldots, N_{Station}$.
Add Gaussian noise $\epsilon \sim N(0, 1)$, to $\hat{\Theta}$.
Clip $\hat{\Theta}$ to ensure that its values remain within the range specified by Θ.
function GENERATETABLE(*Seed*)
    Initialize the random seed.
    Create a zero array $PaxArriveTable$ of shape $(N_{station}, int(N_{passenger}))$.
    For each row of $PaxArriveTable$, generate passenger arrival times following an exponential distribution.
    Replace zeros in $PaxArriveTable$ with infinity.
    return $PaxArriveTable$
function GENERATEPAX($N_{Station}$, N_{Pax}, $\hat{\Theta}$, $PaxArriveTable1$,$PaxArriveTable2$)
    Compute $N_{Station}$ As $N_{Station} = N_{Station}//2$.
    Reverse the order of rows in $PaxArriveTable2$, then concatenate it with $PaxArriveTable1$.
    For each row of the concatenated table, add a time interval proportionally.
    return The Concatenated Table
GeneratePax(GenerateTable(Seed1), GenerateTable(Seed2))

Given that delays are a common occurrence in bus operations, we incorporated random delays in our training environment. By training the agent with random delays, we aim to make it adaptive to various scenarios and ensure smooth operations. The random delay rd is added every time a bus leaves a station and obeys uniform distribution, $rd \sim U[min\_delay, max\_delay]$.

View the learning randomization parameters in DR as a bilevel optimization problem. Assume that: (1) we have access to bus operation in real environment e_{real} (2) the randomization settings (demand level and random delay) are sampled from a distribution parameterized by $\phi, \xi \sim P_\phi(\xi)$. Our goal is to acquire knowledge of a distribution upon which a policy π_θ can be trained to attain peak performance within the context of e_{real}:

$$\phi^* = \arg\min_{\phi} \mathcal{L}\Big(\pi_{\theta^*(\phi)}; e_{real}\Big) \tag{13}$$

$$\theta^*(\phi) = \arg\min_{\theta} \mathbb{E}_{\xi \sim P_\phi(\xi)}\Big[\mathcal{L}\Big(\pi_\theta; e_\xi\Big)\Big] \tag{14}$$

where $\mathcal{L}(\pi; e)$ is the loss function of policy π evaluated in the environment e.

In the bilevel optimization framework defined by (13) and (14), the goal is to learn the optimal distribution for randomization parameters, denoted by ϕ. The variable ϕ parameterizes the distribution $P_\phi(\xi)$, from which the randomization settings ξ are sampled. This distribution $P_\phi(\xi)$ defines the variability within the source environment e_ξ. The outer optimization objective seeks to adjust ϕ to minimize the policy's performance loss in the real environment e_{real}. $\pi_{\theta^*(\phi)}$ represents the policy trained using optimal parameters $\theta^*(\phi)$, where $\theta^*(\phi)$ minimizes the expected loss in the source environment e_ξ under the distribution $P_\phi(\xi)$. This two-level optimization structure enables us to find the optimal

randomization distribution ϕ^*, facilitating the training of a policy $\pi_{\theta^*(\phi^*)}$ that generalizes effectively and achieves peak performance in the real environment e_{real}.

In DR, even though the ranges for randomization are manually selected, this approach necessitates domain expertise and a series of iterative adjustments through trial and error. In nature, this constitutes a manual optimization procedure for adjusting ϕ to achieve the optimal $\mathcal{L}\left(\pi_{\theta^*(\phi)}; e_{real}\right)$ within the context of e_{real}.

4.4. Curriculum Learning

In improving bus operations with reinforcement learning (RL), managing a discrete action space containing actions such as holding, skipping stations, and turn-arounds presents unique challenges. The action space varies in granularity, from temporal adjustments (holding for specific seconds) to substantial route alterations (turn-arounds), complicating policy formulation due to their non-uniform impact on state transitions. When actions have significantly different impacts on the system's dynamics, the agent might struggle to adequately explore and evaluate all actions. This heterogeneity in action types mandates a curriculum learning approach, where the agent sequentially masters each action, developing a robust understanding of when and how to deploy each action optimally. In our implementation, we built an amount of Φ difficulty levels, each difficulty level is numbered by $\varphi, \varphi = 1, 2, ...$ Within one difficulty level, only certain actions can be taken. Actions that are allowed to be used at difficulty level φ are stored in array a_φ. The difficulty level will increase once the current reward r reaches the threshold of a certain reward at that difficulty level r_φ or the reward does not increase after many iterations (it is set to 20,000 iterations in our paper, which means if the reward at difficulty level φ remains the same over 20,000 iterations, the difficulty level will be increased to $\varphi + 1$).

We test different settings of curriculum, as shown in Table 4. In test 1, we first train the model with only the holding method, then add skipping and turning around. In test 2, since the action space of holding is bigger than skipping and turning around, we split the holding action into 2 difficulty levels. In tests 3 and 4, we switch the order of the curriculum. Also, more actions do not necessarily mean a higher difficulty level. Therefore, we add test 5 to first implement all the actions, then gradually decrease and increase the actions. It turns out that only test 3 shows a better result than no curriculum learning. This means learning skipping first and then adding turning around and holding can help the agents learn a better strategy. Intuitively, this also makes sense because skipping and turning around has fewer action spaces than holding.

Table 4. Different Settings of Curriculum and the Final Reward.

Test	Φ	a_1	a_2	a_3	a_4	a_5	Final Reward	Better Than No CL
1	3	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	/	/	−1.04	
2	4	0, 1, 2, 3, 4, 5, 6	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	/	−1.04	
3	3	12	12,13	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	/	/	−1	✓
4	3	13	12,13	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	/	/	−1.05	
5	5	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	−1.03	

5. Experiments

5.1. Setup

5.1.1. Environment Parameters

The simulation framework is constructed based on the parameters outlined in Section III and illustrated in Figure 3. The model includes a corridor with $m = 20$ uniformly distributed bus stations and a system comprising $n = 14$ buses. The capacity of each bus $C = 60$. The journey time between any two adjacent stops, excluding time spent at stops, is 3 min. Passenger boarding and alighting times adhere to the standards specified by [12], which are 1.8 s per passenger for alighting and 3.0 s per passenger for boarding. The upper limit of holding time at stops is set to 120 s.

5.1.2. Simulation Settings

At the initialization of the simulation, buses depart from the first stop s_1 according to the headway of a bus line at Chicago. The departure data is gathered from Chicago Transit Authority (CTA). The rate of passenger arrivals at each stop follows a Poisson distribution fitted from CTA data. The simulation advances in one-second increments and the time horizon is set to 3 h. The granularity of time in the simulation (how often actions can be taken) is set as 1s.

5.1.3. RL Algorithm Parameters

In the simulation setup for the reinforcement learning model, several key parameters define the learning dynamics and optimization strategy for the PPO agent operating within a bus operation environment. The discount factor is set at 0.99, indicating that future rewards are almost as significant as immediate ones, but slightly less so, which helps the agent value long-term outcomes effectively. The learning rate is configured at 0.01, providing a moderate pace for updating the agent's knowledge based on new data, which balances the trade-off between learning speed and stability. Batch size is specified at 128, determining the number of experiences the model processes before performing an update, thus impacting the granularity of learning and the smoothness of the update trajectory. The training process set a total of 70,000 steps.

5.2. Ablation Study and The Effectiveness of Three Strategies

The process of systematically modifying aspects of an algorithm to assess the contribution of each component is known as an ablation study. By removing or altering certain parts of the RL algorithm, we can identify which elements are crucial to performance and which are ancillary. We design an ablation study focused on the PPO algorithm, where we incrementally integrate Long Short-Term Memory (LSTM) networks and DR to evaluate their impact on the model's learning process.

Our study begins by establishing a baseline using the Simple PPO algorithm. As described in Section 4.1, PPO utilizes a clipped objective function to prevent large policy updates, which could lead to performance collapse. This baseline serves as a control for subsequent experiments, ensuring that any observed improvements can be attributed to the newly added components.

After establishing the baseline performance with Simple PPO, we introduce an LSTM network with hidden unit dimension of 256 into the PPO framework. By incorporating LSTM, we aim to determine whether the agent's ability to remember and utilize past information enhances its decision-making process, leading to improvements in both performance and efficiency.

The final component added to our RL model is DR, a technique designed to improve the robustness of the learned policy by training the agent across a variety of randomized environments. As described in Section 4.3, this approach can be particularly beneficial in scenarios where the trained agent is expected to be deployed in real-world conditions

that differ from the simulated training environment. By exposing the agent to a range of variations during training, we expect it to generalize better to unseen situations.

The ablation study is structured into four distinct experiments, as outlined as follows:

- Senario1: Simple PPO
- Senario2: Simple PPO + LSTM
- Senario3: Simple PPO + DR
- Senario4: Simple PPO + LSTM + DR
- Senario5: Simple PPO + LSTM + DR + Curriculum Learning

The reward plots of different scenarios is shown in Figure 5. From the ablation study, we can observe improvements in performance and efficiency when adding new models.

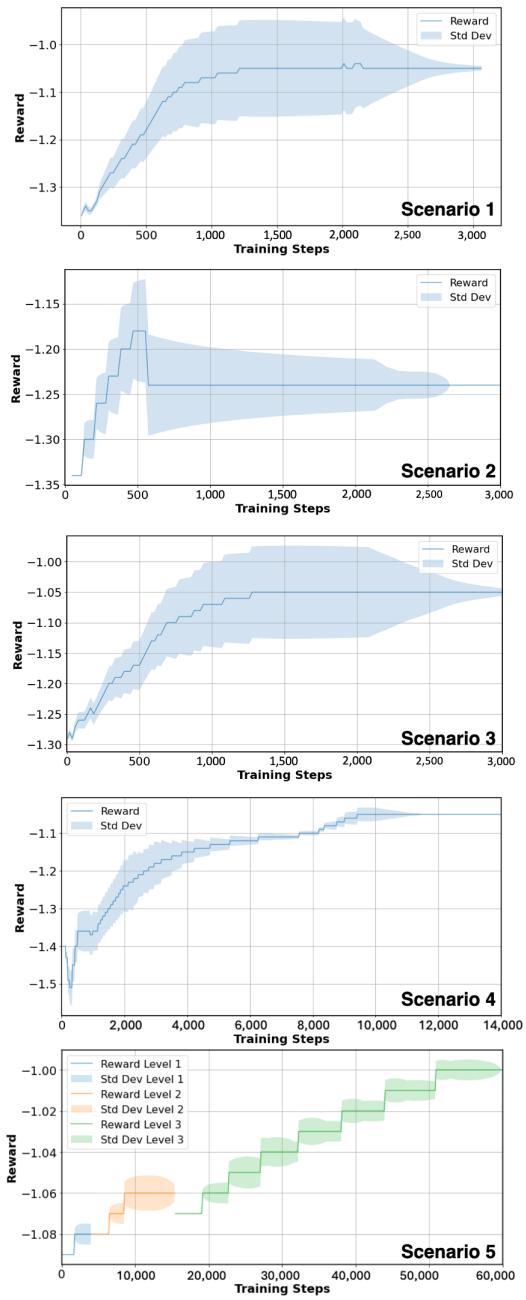

Figure 5. Thereward plots of different scenarios with three strategies implemented

As is shown in Section 4.4, the reward does not increase if the holding strategy is trained through curriculum learning. Therefore, we let $\Phi = 1$, and $a_1 = [0, 1, 2, 3, 4, 5, 6, 7, 8,$

9, 10, 11] in order to implement the holding strategy. The comparison of the final reward of holding only and three strategies is shown in Table 5. Figure 6 illustrates that the bus schedule becomes more evenly distributed after training. The reward represents the total time spent in the system, as stated in Section 4.1. The total time spent in the system is decreased by 6% and the average passenger waiting time is reduced by 16%. For example, in a bus system with 14 buses and 20 stations, when the headway is set at 5 min, it results in an average reduction of about 50 s in the waiting time per passenger. Therefore, with suitable algorithm implemented, a combination of three strategies yields more effective results than solely relying on the holding strategy.

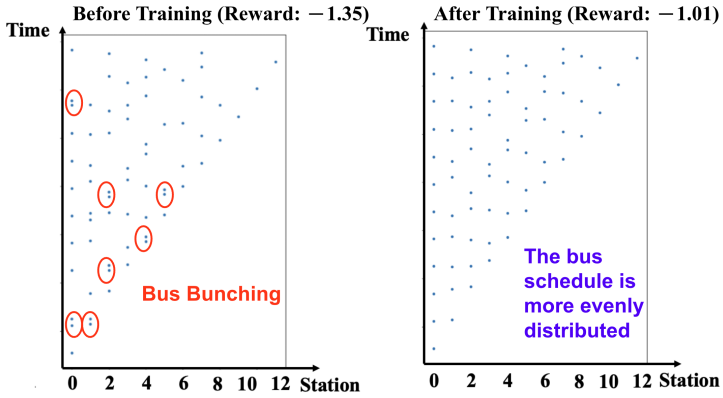

Figure 6. Bus Arrival Time Before and After Training.

Table 5. A Comparison between Holding Only and Three Strategies.

Scenario	Algorithm				Reward		
	PPO	LSTM	Domain Randomization	Curriculum Learning	Holding Only	Three Strategies	Better Use Three Strategies
1	✓				−1.06	−1.05	✓
2	✓	✓			−1.05	−1.24	
3	✓		✓		−1.05	−1.05	
4	✓	✓	✓		−1.04	−1.01	✓
5	✓	✓	✓	✓	−1.04	−1	✓

5.3. The Robustness in Different Scenarios

To evaluate the result, we test the performance of DR model and compare it with the model without DR. Compared to the basic scenario, we change three parameters (number of buses, number of stations, and headway). In order to see the robustness of our model, we also tested the performance of DR under different levels of noise. The scenarios and the corresponding results are listed in Figure 7. The model employing DR achieves superior outcomes in different scenarios in terms of its final reward, average reward, and reduced average waiting time. Considering that the model integrating DR incorporates stochastic noise, it follows that the standard deviation would exhibit an elevation.

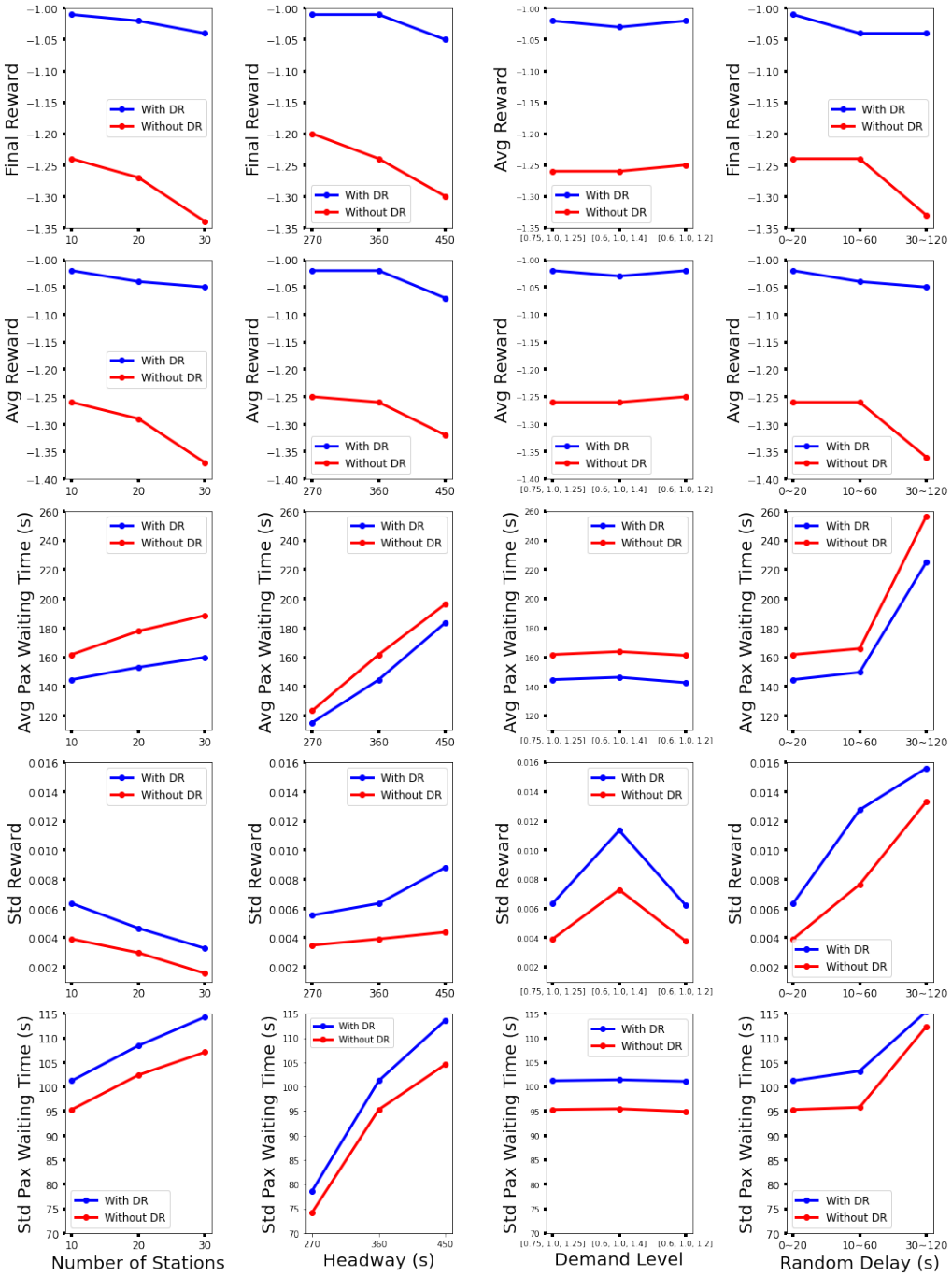

Figure 7. The evaluation of DR.

6. Conclusions and Future Work

In this paper, we propose a novel RL framework to improve bus operations. The model contain three high-level actions: holding, skipping, and turning around. These actions aim to reduce passenger waiting times. In order to let the model memorize the previous action and affect the current action, LSTM is implemented to capture both Markov and Non-Markov processes. To accommodate the unpredictability in bus operations and enable the agent to gradually expand its knowledge across the entire action space, we developed a curriculum and applied domain randomization during training. Our approach has demonstrated its effectiveness in adapting to the proposed action space, outperforming agents that rely solely on holding strategies. This shows the potential for efficiently

learning from sparser reward signals and navigating more complex action spaces with our new method.

This study demonstrates that curriculum learning effectively enables the agent to navigate complex and varied action spaces. Additionally, incorporating domain randomization in the training process enhances the robustness of the resultant model. Moreover, the introduced action space, which includes holding, skipping, and turning around, facilitates greater efficiency in bus systems compared to strategies that solely rely on holding.

In order to implement our model in reality, it would require clear communication systems between buses, central control centers, and passengers. Investment in real-time tracking and communication infrastructure would be necessary to coordinate actions like skipping stops or turning around without causing confusion or inconvenience to passengers. Operators on board and in control centers would implement the model's real-time prescriptions by utilizing enhanced communication systems that provide dynamic instructions based on the RL model's outputs. Bus drivers would adjust their actions according to the control center as well as inform the passengers via display screen and broadcast, while control center operators would monitor system performance, oversee the RL algorithms, and coordinate with drivers to ensure effective execution of the strategies. We also hope to mitigate the impact of the disruption to passengers using our model. For instance, an incentive mechanism could be designed to encourage passengers to cooperate, or certain actions could be restricted to ensure passenger convenience, thereby improving service reliability and passenger satisfaction.

In the future, we aim to explore the development of improved curricula using algorithmic strategies tailored for bus operation scenarios. Another future goal is to assess the effectiveness of domain-randomized training for sim-to-real transfer by field-testing the algorithm in real-world, variable traffic conditions. For the potential field test, we will collaborate with a transit agency to select a controlled route, implement tracking systems, and collect passenger feedback to evaluate the effectiveness of the proposed rewards and assess any unmodeled factors affecting bus performance. While theoretical, this outline would guide gradual implementation and offer a structured approach to future testing.

We believe this work opens new avenues in the extensively researched field of transportation operations, introducing innovative problem formulations and methods with lower operational costs, leading to a smarter, more sustainable urban transit system.

Author Contributions: Conceptualization, A.Q. and Y.T.; methodology, A.Q. and Y.T.; validation, A.Q., Y.T. and X.J.; formal analysis, A.Q. and Y.T.; investigation, A.Q. and Y.T.; resources, A.Q. and Y.T.; writing—original draft preparation, Y.T., A.Q., X.J., B.M. and S.C.; writing—review and editing, Y.T., A.Q., X.J., B.M., S.C., J.R., H.N.K., C.W. and J.Z.; visualization, Y.T., A.Q. and X.J.; supervision, H.N.K., C.W. and J.Z.; project administration, J.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The real-world data used to determine some of the parameters was collected in partnership with the Chicago Transit Agency and cannot be released due to privacy restrictions.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Higashide, S. *Better Buses, Better Cities: How to Plan, Run, and Win the Fight for Effective Transit*; Island Press: Washington, DC, USA, 2019.
2. Hu, W. Adaptive Transit Signal Priority Algorithms for Optimizing Bus Reliability and Travel Time Using Deep Reinforcement Learning. Ph.D. Thesis, University of Toronto: Toronto, ON, Canada, 2022.
3. Daganzo, C.F. A headway-based approach to eliminate bus bunching: Systematic analysis and comparisons. *Transp. Res. Part B Methodol.* **2009**, *43*, 913–921. [CrossRef]

4. Andres, M.; Nair, R. A predictive-control framework to address bus bunching. *Transp. Res. Part B Methodol.* **2017**, *104*, 123–148. [CrossRef]
5. Sánchez-Martínez, G.E.; Koutsopoulos, H.N.; Wilson, N.H.M. Real-time holding control for high-frequency transit with dynamics. *Transp. Res. Part B Methodol.* **2016**, *83*, 1–19. [CrossRef]
6. Daganzo, C.F.; Pilachowski, J. Reducing bunching with bus-to-bus cooperation. *Transp. Res. Part B Methodol.* **2011**, *45*, 267–277. [CrossRef]
7. Bartholdi, J.J.; Eisenstein, D.D. A self-coördinating bus route to resist bus bunching. *Transp. Res. Part B Methodol.* **2012**, *46*, 481–491. [CrossRef]
8. Estrada, M.; Mensión, J.; Aymamí, J.M.; Torres, L. Bus control strategies in corridors with signalized intersections. *Transp. Res. Part C Emerg. Technol.* **2016**, *71*, 500–520. [CrossRef]
9. Wu, W.; Liu, R.; Jin, W. Modelling bus bunching and holding control with vehicle overtaking and distributed passenger boarding behaviour. *TRansportation Res. Part B Methodol.* **2017**, *104*, 175–197. [CrossRef]
10. Daganzo, C.F.; Ouyang, Y. *Public Transportation Systems: Principles of System Design, Operations Planning and Real-Time Control*; World Scientific Publishing Company: Singapore, 2019.
11. Xiao, M.; Xiahou, J.; Ge, M. A Reinforcement-Learning-Based Bus Scheduling Model. In Proceedings of the 2022 IEEE 10th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 17–19 June 2022; pp. 923–927. [CrossRef]
12. Chen, W.; Zhou, K.; Chen, C. Real-time bus holding control on a transit corridor based on multi-agent reinforcement learning. In Proceedings of the 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), IEEE, Rio de Janeiro, Brazil, 1–4 November 2016; pp. 100–106. [CrossRef]
13. Wang, J.; Sun, L. Dynamic holding control to avoid bus bunching: A multi-agent deep reinforcement learning framework. *Transp. Res. Part C Emerg. Technol.* **2020**, *116*, 102661. [CrossRef]
14. Alesiani, F.; Gkiotsalitis, K. Reinforcement Learning-Based Bus Holding for High-Frequency Services. In Proceedings of the 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, 4–7 November 2018; pp. 3162–3168. [CrossRef]
15. Tham, M.L.; Tay, B.S.; Khor, K.C.; Phon-Amnuaisuk, S. Deep Reinforcement Learning Based Bus Stop-Skipping Strategy. In Proceedings of the 2023 International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC), Jeju, Republic of Korea, 25–28 June 2023; pp. 1–6. [CrossRef]
16. Poliziani, C.; Needell, Z.; Laarabi, H.; Waraich, R.; Blick, A.; Fujita, K.S.; Rezaei, N.; Caicedo, J.; Guirado, C.; Spurlock, C.; et al. Simulating Impacts from Transit Service Enhancements in the San Francisco Bay Area *Transp. Res. Rec.* **2024**. [CrossRef]
17. Rodriguez, J.; Koutsopoulos, H.N.; Wang, S.; Zhao, J. Cooperative bus holding and stop-skipping: A deep reinforcement learning framework. *Transp. Res. Part C Emerg. Technol.* **2023**, *155*, 104308. [CrossRef]
18. Wang, X.; Chen, Y.; Zhu, W. A survey on curriculum learning. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**, *44*, 4555–4576. [CrossRef] [PubMed]
19. Narvekar, S.; Peng, B.; Leonetti, M.; Sinapov, J.; Taylor, M.E.; Stone, P. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *J. Mach. Learn. Res.* **2020**, *21*, 1–50.
20. Racanière, S.; Lampinen, A.K.; Santoro, A.; Reichert, D.P.; Firoiu, V.; Lillicrap, T.P. Automated curricula through setter-solver interactions. *arXiv* **2019**, arXiv:1909.12892.
21. Matloff, N. Introduction to Discrete-Event Simulation and the Simpy Language. Ph.D. Thesis, Deptment of Computer Science, University of California, Davis, CA, USA, August 2008; Volume 2, pp. 1–33.
22. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
23. Graves, A.; Graves, A. Long short-term memory. In *Supervised Sequence Labelling with Recurrent Neural Networks*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 37–45.
24. Peng, X.B.; Andrychowicz, M.; Zaremba, W.; Abbeel, P. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018; pp. 3803–3810. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.