Chapter 1: Introduction

## 1.1 Research Background and Motivation

The global retail industry is undergoing a structural transformation driven by e-commerce. Projections indicate that the annual value of global retail e-commerce will grow to $6.42 trillion by 2025, accounting for over 20% of total retail sales [1]. To cope with the new normal of e-commerce orders—characterized by “small batches, high frequency, and time sensitivity”—traditional warehousing models are no longer adequate. This has prompted businesses to adopt **Robotic Mobile Fulfillment Systems (RMFS)** on a large scale. In an RMFS, hundreds of Autonomous Mobile Robots (AMRs) work collaboratively within a grid-based warehouse, capable of increasing order picking efficiency by several folds.

However, the high-density operation of AMRs also introduces new operational bottlenecks. **Traffic congestion at intersections, waiting delays between robots, and frequent acceleration and deceleration behaviors** not only directly limit the overall throughput of the system but also significantly increase energy consumption and carbon emissions. This issue has become critically important under the increasingly stringent global policies on sustainable development and carbon neutrality:

* **High Carbon Footprint**: Carbon emissions from transportation and warehousing activities account for approximately 24% of total global greenhouse gas emissions and have been identified as a priority area for carbon reduction by international logistics leaders like DHL [2].
* **Urgent Policy Pressure**: The European Union’s **Carbon Border Adjustment Mechanism (CBAM)** was initiated in 2023 and is scheduled to be fully implemented in 2026. This mechanism will impose additional costs on imported products with high carbon intensity, compelling companies throughout the supply chain to rigorously inspect and manage carbon emissions in their warehousing and transportation links [3].

Against this backdrop, academia has begun to apply intelligent methods such as Reinforcement Learning (RL) to optimize RMFS operations. Some studies have used deep reinforcement learning to schedule orders to minimize costs [4], while others have reduced total energy consumption by about 3–5% by adjusting traffic strategies and robot speeds [5]. However, these studies either focus on order-level scheduling or still have room for exploration in the trade-off between energy consumption and efficiency. **Currently, a control framework that focuses on the intersection level and aims for the dual optimization of “energy-efficiency” is still lacking.**

To fill this research gap, this study proposes an intersection controller based on **Neuroevolution Reinforcement Learning (NERL)**. This method combines the global search capabilities of evolutionary algorithms with the real-time decision-making advantages of deep reinforcement learning. It aims to provide an “adaptive, energy-aware” right-of-way allocation mechanism for RMFS intersections, with the expectation of effectively reducing the energy consumption per task for AMRs while maintaining order processing efficiency.

## 1.2 Research Objectives

To validate the aforementioned concept and address the core research questions, this study formulates the following four specific objectives. The results will be elaborated in subsequent chapters:

1. **Construct a High-Fidelity RMFS Simulation and Control Platform**
   * Establish a physical warehouse environment that includes a central storage area, one-way aisles, workstations, and charging stations.
   * Design a modular traffic control system architecture based on the Strategy and Factory design patterns to support the flexible integration and fair comparison of different algorithms (see Section 3.2 for details).
2. **Design and Implement Multiple Traffic Control Strategies**
   * Develop two heuristic baseline controllers: a **Time-Based Controller** based on fixed time cycles, and a **Queue-Based Controller** based on real-time waiting queues (see Section 3.3 for details).
   * Implement a value-based deep reinforcement learning method, the **Deep Q-Network (DQN)**, to serve as a benchmark for DRL comparison (see Section 3.4.1 for details).
   * Develop the core contribution of this research, the **Neuroevolution Reinforcement Learning (NERL) Controller**, and investigate the impact of different evolutionary hyperparameters (exploratory vs. exploitative) (see Section 3.4.2 for details).
3. **Design and Execute Rigorous Controlled Experiments**
   * Define a comprehensive set of performance evaluation metrics covering efficiency, throughput, and stability (see Section 3.2.4 for details).
   * Design a comparative matrix with twelve independent experimental groups to systematically compare the performance of different controllers under two reward modes (step-based reward vs. global reward) and different evaluation durations (see Section 3.6.1 for details).
4. **Quantitative Analysis and Statistical Validation**
   * Conduct an in-depth quantitative analysis and comparison of the performance of all control strategies based on experimental data.
   * Employ statistical methods, such as the independent samples t-test, to scientifically verify whether the performance improvement of the proposed NERL method is statistically significant compared to the baseline controllers (see Section 3.5.4 for details).

## 1.3 Research Scope and Limitations

To focus on the core problem and ensure the depth of the research, this study explicitly defines its scope. The following sections list the core areas covered by this research and the aspects excluded to simplify the model.

### 1.3.1 Scope of Research

| Item | Content | Description |
| --- | --- | --- |
| **Core Problem** | Dynamic allocation of right-of-way at intersections. | The research focuses on resolving congestion and energy waste caused by improper traffic coordination at intersections. |
| **Control Strategies** | Time-Based, Queue-Based, DQN, NERL. | Encompasses a comprehensive comparison from static rules and dynamic heuristics to two different DRL frameworks. |
| **Optimization Goals** | Minimize total energy consumption, maximize order completion, minimize average waiting time. | Adopts a multi-objective optimization perspective to evaluate the overall performance of strategies in terms of efficiency and energy consumption. |
| **Evaluation Metrics** | Efficiency, throughput, and stability metrics defined in Section 3.2.4. | Utilizes a complete set of quantitative metrics to thoroughly evaluate the pros and cons of each controller. |

### 1.3.2 Research Limitations

This study intentionally excludes the following issues from its scope to maintain focus on the core problem. These issues can serve as directions for future research.

* **Path Planning**: This research does not involve global path planning algorithms for robots from start to destination. It is assumed that all robots follow a predefined shortest path.
* **Slotting Assignment**: This research does not optimize the storage location of stock-keeping units (SKUs) on shelves or the placement of shelves within the warehouse. It is assumed that these are randomly or pre-configured.
* **Charging Strategy Management**: Although charging stations exist in the simulation environment, this study does not involve monitoring robot battery levels and proactively dispatching them for charging.

# Chapter 2: Literature Review

## 2.1 Chapter Overview

This chapter aims to systematically review the research related to traffic management in Robotic Mobile Fulfillment Systems (RMFS). The chapter will unfold progressively: first, it will elaborate on the root causes of traffic congestion in RMFS and its profound impact on system performance. Subsequently, it will sequentially evaluate three categories of control strategies relevant to this study—traditional heuristic methods, mainstream Deep Reinforcement Learning (DRL) approaches, and emerging Neuroevolution methods—assessing their applications, achievements, and limitations in addressing this problem. Through this structured analysis, the chapter aims to precisely identify the existing research gap and ultimately articulate the necessity and novelty of proposing a traffic control strategy that integrates energy considerations and is based on Neuroevolution Reinforcement Learning (NERL).

## 2.2 Robotic Mobile Fulfillment Systems: Operational Efficiency and Traffic Bottlenecks

As a “Goods-to-Person” paradigm in warehouse automation, the core of RMFS involves using a large number of Autonomous Mobile Robots (AMRs) to transport mobile shelves (Pods), thereby replacing traditional manual picking routes and significantly enhancing order fulfillment efficiency [6]. Since the acquisition and large-scale deployment of Kiva Systems by Amazon [7], RMFS has become a fundamental infrastructure for modern e-commerce logistics. Studies indicate that its throughput can be two to four times that of traditional warehouses. As companies like Amazon deploy over 750,000 robots, the operational density and complexity of these systems have reached unprecedented levels [8].

However, high-density deployment brings a new and severe bottleneck: **Traffic Congestion**. As defined in Chapter 3 of this study, when a large number of robots execute tasks within a limited network of aisles, especially at intersections, conflicts and waiting periods are highly likely to occur due to improper right-of-way coordination [9]. This triggers a series of cascading negative effects, not only causing delays in order fulfillment and reducing overall system throughput, but recent research also indicates that frequent stop-and-go behaviors significantly increase overall energy consumption [10] and may accelerate the aging of robot batteries, thereby increasing operational costs. Therefore, designing an **intersection management strategy** that is adaptive and balances both efficiency and energy sustainability has become a core challenge in enhancing the overall performance of RMFS.

Within the complex decision-making framework of RMFS, traffic control and path planning are the elements most directly related to the system’s real-time performance. In recent years, many studies have focused on solving this problem. For example, **Luo et al. (2023) [11]** proposed a path planning method combining the A\* algorithm and DQN, while **Zhou et al. (2024) [12]** used an attention-based DRL model to collaboratively handle multi-AGV scheduling and pod reconfiguration problems. These studies highlight the immense potential of intelligent algorithms in RMFS operational optimization. Although this research focuses on intersection traffic control, its optimization goal—enhancing system fluidity and efficiency—is fundamentally aligned with these studies.

## 2.3 Review and Analysis of Intersection Traffic Control Methods

To address the intersection traffic problem in RMFS, various control strategies have been proposed in academia and industry. Their development can be broadly categorized into heuristic methods and learning-based methods, reflecting an evolutionary trend from static rules to dynamic, adaptive control.

### 2.3.1 Heuristic and Rule-based Controllers

Early solutions often relied on manually designed heuristic rules. Their advantages include simplicity of implementation and low computational overhead, making them common baselines for evaluating the performance of more complex algorithms.

One category is **Static Time-based Control**, which borrows from traditional urban traffic light systems to set fixed right-of-way switching cycles for intersections. This concept aligns with early fixed-rule urban traffic solutions [13] and corresponds to the **TimeBasedController** designed in Section 3.3.1 of this study. However, its fundamental drawback is its complete inability to perceive real-time traffic flow changes. When traffic fluctuates dramatically, this static strategy is prone to wasting green light time on idle directions or causing unnecessary queue buildup in busy directions.

To address the shortcomings of static methods, another category, **Dynamic Reactive Control**, emerged. These methods react to the real-time state of an intersection, such as the length of waiting queues. For instance, the study by **Teja (2009) [14]** assigned agents to intersections to manage traffic flow based on the priority of incoming robots. This approach forms the conceptual basis for the **QueueBasedController** designed in Section 3.3.2 of this study. Although more flexible, the design of these heuristic rules heavily relies on expert experience and prior knowledge. Their predefined weights and logic struggle to capture the optimal coordination strategy in complex traffic patterns, often leading to suboptimal performance in highly dynamic and non-linear traffic scenarios [15].

### 2.3.2 Deep Reinforcement Learning (DRL) Based Controllers

To overcome the limitations of manual rules, Deep Reinforcement Learning (DRL) has been widely applied to traffic control problems. DRL enables an agent (the controller) to autonomously learn complex decision-making policies from interactions with the environment to maximize long-term cumulative rewards [16].

This paradigm has achieved significant success in the macroscopic field of **urban traffic signal control**. Numerous studies have demonstrated that methods based on Deep Q-Networks (DQN) and their variants significantly outperform traditional methods in reducing vehicle delays and increasing network throughput [17]. For example, **Cao et al. (2024) [18]** proposed G-DQN to improve convergence speed; **Zhu et al. (2021) [19]** compared the performance of PPO and DQN in single-intersection control; and **Angela et al. (2024) [20]** designed a reward function that incorporates multi-dimensional information such as pressure, queue length, and speed to train DQN agents. These studies fully showcase the powerful capabilities of DRL in handling sequential decision-making problems.

In recent years, researchers have begun to transfer this paradigm to RMFS scenarios. They typically use local traffic information at intersections (e.g., queue length, waiting time) as the **State**, switching traffic phases as the **Action**, and maximizing throughput or minimizing waiting time as the **Reward** to train the controller [21] [22]. Although DQN-based methods show great potential, they also face inherent challenges: 1. **Reward Design Sensitivity**: The performance of DQN is highly dependent on dense and well-designed real-time reward signals. As analyzed in Section 3.4.3 of this study, poor or overly simplistic reward designs can lead the agent to learn unintended, suboptimal behaviors (e.g., frequently oscillating signals just to receive switching rewards). 2. **Risk of Local Optima**: In complex, high-dimensional state spaces, gradient-based optimization methods can sometimes converge to local optima, making it difficult to discover globally optimal coordination strategies. 3. **Sample Efficiency Issues**: DQN typically requires a large number of environmental interaction samples to learn an effective policy, and the training process can be very time-consuming.

Given the representativeness and maturity of DQN in this field, this study uses it as a key **baseline** for objectively evaluating the potential improvements offered by the proposed new method.

## 2.4 The Potential of Neuroevolution Reinforcement Learning (NERL)

To address the challenges faced by traditional DRL methods, this study introduces another optimization paradigm: Neuroevolution Reinforcement Learning (NERL). Unlike DQN, which attempts to learn a value function, NERL searches directly in the **parameter space** of the policy network. It treats the weights of a neural network as an individual’s “genes” and uses evolutionary algorithms (such as elitism, crossover, and mutation) to optimize an entire population of agents, ultimately yielding high-performance control policies [23].

NERL exhibits several unique advantages in solving complex control problems, making it particularly suitable for the objectives of this research: 1. **Robustness to Reward Functions**: Evolutionary algorithms are gradient-free, black-box optimization methods that do not require the reward function to be continuous or differentiable. This makes them naturally suited for handling **sparse, delayed, or globally-defined reward signals**. For example, the “total number of completed orders” or “total energy consumption” over an entire simulation period can be directly used as a fitness function. This aligns perfectly with the **Global Reward** concept designed in Section 3.4.3 of this study, thereby avoiding tedious reward engineering. 2. **Stronger Global Exploration**: The crossover and mutation operations in the evolutionary process facilitate broader exploration of the parameter space. By maintaining population diversity, this reduces the risk of getting trapped in local optima and increases the likelihood of discovering innovative and efficient control strategies. 3. **Inherent Parallelism**: The evaluation process for each individual (policy network) in the population is independent and can be deployed in a massively parallel fashion on multi-core CPUs or computing clusters. This allows for the efficient use of modern computational resources, significantly reducing the wall-clock time for training [24].

Although neuroevolution has achieved remarkable success in fields like robotics control and game AI [25], **its specific application to RMFS intersection traffic control, and its systematic comparison against traditional heuristic methods and mainstream DRL methods (like DQN) on a unified platform that considers both time efficiency and energy consumption, remains a largely unexplored area.**

## 2.5 Chapter Summary and Research Gap

Based on the analysis above, the development trajectory of existing RMFS traffic control research clearly points to several key unresolved issues. Control strategies are evolving from simple static rules to dynamic heuristics, and further to adaptive methods based on DRL. However, existing DRL methods (mainly DQN) still face challenges in reward design, avoidance of local optima, and optimization for long-term global objectives. Meanwhile, most traffic control research focuses on throughput and time delay, while studies that treat **energy consumption** as a core optimization objective of equal importance to efficiency are relatively scarce.

More importantly, the potential of neuroevolution—as a method with stronger global exploration capabilities and greater robustness to sparse, global rewards—has not been systematically explored or validated for the RMFS traffic control problem.

Therefore, the core **research gap** of this study can be summarized as: **the lack of an intelligent control strategy for RMFS that can effectively balance and optimize both traffic efficiency and energy consumption, while overcoming the limitations of traditional DRL in areas such as reward design and local optima.**

To fill this gap, this thesis proposes an intersection controller based on Neuroevolution Reinforcement Learning (NERL). We will conduct a comprehensive comparison of NERL with traditional heuristic methods and a standard DQN method on a high-fidelity simulation platform that incorporates a detailed energy consumption model. This research not only aims to propose a superior control strategy but also provides a solid empirical basis for evaluating the strengths and weaknesses of different learning paradigms in solving complex multi-agent coordination problems.

To more intuitively compare this study with related literature, Table 2.1 summarizes the similarities and differences between selected representative studies and this research in terms of problem definition, decision criteria, and algorithmic approaches.

**Table 2.1: Comparison with Related Research**

| **Reference** | **Performance** | **Energy** | **Dynamic Routing** | **Traffic Strategy** | **Collision Handling** | **Multi-Agent** | **Analysis** | **Heuristic** | **Simulation** | **Reinforcement Learning** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Li et al. (2020) [10]** | **✓** | **✓** |  |  |  | **✓** | **✓** |  | **✓** |  |
| **Luo et al. (2023) [11]** | **✓** |  | **✓** |  | **✓** |  |  | **✓** | **✓** | **✓** |
| **Yuan et al. (2023) [26]** | **✓** |  |  |  | **✓** | **✓** |  |  | **✓** | **✓** |
| **Zhou et al. (2024) [27]** | **✓** |  | **✓** | **✓** |  | **✓** |  |  | **✓** | **✓** |
| **Cao et al. (2024) [18]** | **✓** |  |  | **✓** |  |  |  |  | **✓** | **✓** |
| **This Study (Ours)** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** | **✓** |

# Chapter 3: Methodology

## 3.1 Problem Definition

In modern automated warehouse Robotic Mobile Fulfillment Systems (RMFS), as order volumes grow and robot deployment density increases, traffic congestion has become a core bottleneck constraining overall operational efficiency. A large number of robots executing tasks within limited aisle space, especially at network intersections, are highly susceptible to conflicts and waiting due to improper right-of-way coordination. This triggers a series of cascading negative effects, including order fulfillment time delays, energy waste from futile robot waiting, and a decline in the system’s overall throughput.

Therefore, the central research question of this study is: **How to design an effective and adaptive traffic control strategy to dynamically regulate intersection right-of-way, thereby minimizing robot waiting time and overall energy consumption, ultimately enhancing the operational efficiency of the entire warehouse system?**

To answer this question, it is imperative to first construct an experimental platform capable of accurately simulating the aforementioned challenges. The experimental environment and system architecture detailed in this chapter constitute a high-fidelity testbed tailored for this core problem. Its purpose is to provide a stable, controllable, and quantifiable foundation for the implementation, training, and comparison of different control algorithms in subsequent chapters.

## 3.1.1 Chapter Structure

This section aims to provide a comprehensive overview of the experimental platform supporting this research. The content will cover four core parts: first, a detailed elaboration of the constructed warehouse simulation environment, including its physical layout and core entities (Section 3.2.1); second, a description of the traffic control system architecture designed for algorithm evaluation (Section 3.2.2); third, a list of the specific hardware and software configurations used for the experiments (Section 3.2.3); and finally, the definition of the quantitative evaluation metrics used to measure the effectiveness of various control strategies (Section 3.2.4). Through this section, readers will gain a complete understanding of the fundamental environment and evaluation criteria for the experiments conducted.

## 3.2 Experimental Environment and System Architecture

To effectively validate and compare the Neuroevolution Reinforcement Learning (NERL) traffic control strategy proposed in this study with other baseline methods, it is essential to construct a high-fidelity simulation platform that accurately reflects the challenges of real-world warehouse operations. This platform serves not only as a testbed for the algorithms but also as a prerequisite for ensuring that all comparisons are conducted on a fair, controllable, and quantifiable basis.

This section will comprehensively describe the composition of this experimental platform. The content is sequentially divided into five core parts: - **3.2.1 Warehouse Simulation Environment Design**: Details the physical layout of the simulated world, the road network structure, and the core entity components, including robots, pods, and workstations. - **3.2.2 Traffic Control System Architecture**: Provides an in-depth analysis of the software architecture designed for algorithm integration and evaluation, explaining how the system drives decisions, executes control, and triggers learning. - **3.2.3 Experimental Hardware and Software Configuration**: Lists the specific hardware specifications and software libraries used for all simulation and training experiments. - **3.2.4 Performance Evaluation Metrics Definition**: Clearly defines the Key Performance Indicators (KPIs) used to measure the effectiveness of different control strategies, serving as the basis for the experimental results analysis in subsequent chapters. - **3.2.5 Robot Physical Model and Energy Consumption**: Details the robot physical model and its energy consumption calculation method used in the simulation, providing a physical basis for energy efficiency evaluation.

Through the exposition in this section, readers will gain a complete understanding of the fundamental environment and evaluation criteria for the experiments conducted in this research.

## 3.2.1 Warehouse Simulation Environment Design

To conduct an effective evaluation of traffic control strategies, this study first constructs a high-fidelity warehouse simulation environment. This environment not only defines the physical layout of the space but also includes various dynamic entities and their interaction rules, which together form a complex Robotic Mobile Fulfillment System (RMFS). This section details its design.

### 1. Physical Environment and Layout

The simulated warehouse is built on a two-dimensional discrete grid, where each grid cell has a specific function. The overall layout adopts a functional zoning design to ensure an orderly operational flow.

* **Central Storage Area**: Located in the center of the warehouse, this area consists of a dense arrangement of **Pod Locations**. The aisles in this zone are designed as strictly **One-way Aisles**, with the flow directions of horizontal and vertical aisles arranged in an alternating pattern. This design physically simplifies the complexity of traffic management and aims to reduce potential conflicts when robots travel in opposite directions.
* **Workstation Area**: Distributed on both sides of the warehouse. One side features **Picking Stations**, which are the exit points for order fulfillment. The other side has **Replenishment Stations**, which are the entry points for goods into the system.
* **Charging Stations**: Scattered within the storage area, these are converted from some pod locations and are available for robots to charge autonomously.

**【圖表建議：圖 3.2.1 - 倉儲佈局示意圖】** 為直觀展示佈局，建議此處插入一張示意圖，用不同顏色標示出儲存區、揀貨區、補貨區與充電站，並用箭頭清晰標示出單向通道的流動方向。

### 2. Core Entities and Lifecycles

The dynamic behavior of the system is driven by the interactions among several core entities.

* **Robot**: As the most central active unit in the system, the robot has a complex state machine to manage its workflow, including states such as idle, taking\_pod, delivering\_pod, station\_processing, and returning\_pod. This study establishes a detailed physical and energy model for the robot. Its energy consumption calculation considers not only the load but also startup costs and regenerative braking (recovering energy during braking), providing a solid basis for energy efficiency evaluation. Furthermore, robots are equipped with priority-based autonomous collision avoidance logic, enabling them to resolve local conflicts to some extent.

**【圖表建議：圖 3.2.2 - 機器人狀態轉換圖】** 為清晰展示機器人的工作流程，建議此處插入一張 UML 狀態機圖，描繪其核心狀態以及觸發狀態轉換的事件（如「分配新任務」、「到達工作站」等）。

* **Pod**: A mobile carrier for storing goods (SKUs). Each pod can hold multiple types of SKUs and records the current quantity and replenishment threshold for each SKU. When the stock level falls below the threshold, the system automatically triggers a corresponding replenishment task.
* **Station**: The node for human-robot collaboration. When a robot delivers a pod to a workstation, the system simulates the picking or replenishment delay caused by a human worker. To handle high traffic, workstations are also designed with a dynamic path adjustment mechanism, which activates an alternative long path to alleviate congestion when there are too many robots at the station.

### 3. Order and Task Flow

The driving force of the simulation comes from orders. An **Order** represents a customer request, containing multiple SKUs to be picked. The system breaks down an order into one or more **Jobs**. The core of a job is “to transport a specified pod to a designated workstation,” which is the smallest work unit that can be directly assigned to a robot. The entire process is as follows: 1. The system receives an order. 2. The SKUs required for the order are located on specific pods. 3. The system generates one or more jobs and places them in a job queue. 4. An idle robot retrieves a job from the queue and begins its work cycle of picking up, delivering, and returning the pod. 5. When all SKUs required for an order have been successfully delivered to the workstation, the order is marked as complete.

## 3.2.2 Traffic Control System Architecture

To enable flexible integration and fair comparison of different traffic control algorithms, this study designs a modular software architecture based on the **Strategy Pattern** and **Factory Pattern**. The core of this architecture is to decouple the “decision-making algorithm” from the “system execution framework,” ensuring that whether it is a simple rule-based logic or a complex deep reinforcement learning model, it can operate and be evaluated on the same foundation. The system is primarily composed of the following three components:

### 1. TrafficController (Abstract Base Class for Traffic Controllers)

This Abstract Base Class (ABC) defines a unified interface that all traffic control strategies must adhere to. Its most critical method is get\_direction(intersection, tick, warehouse), which receives the detailed current state of an intersection (local information) and the state of the entire warehouse system (global information), and returns a traffic decision for that intersection (e.g., "Horizontal" or "Vertical"). By forcing all controllers to implement this interface, the system ensures consistency in how different algorithms are called. Additionally, the base class integrates standardized statistical data collection functions to record various performance metrics.

### 2. TrafficControllerFactory (Controller Factory)

This class adopts the Factory design pattern and is responsible for dynamically creating instances of TrafficController subclasses based on external configurations (such as the controller type specified in an experiment configuration file). When the simulation core requires a controller, it only needs to provide a string identifier like "dqn", "nerl", or "time\_based", and the factory will return a corresponding, initialized controller object. This design completely decouples the “creation logic” of the controllers from their “usage logic,” significantly enhancing the flexibility and scalability of the experimental workflow, allowing different control strategies to be switched without modifying any core simulation code.

### 3. IntersectionManager (Intersection Manager)

The Intersection Manager is the central coordinator and execution engine of the entire traffic control system. Its operational flow constitutes a complete closed-loop control system: 1. **Holds a Controller Instance**: During the simulation initialization phase, the IntersectionManager obtains the controller instance required for the current experiment through the controller factory. 2. **Drives the Decision Loop**: In each time unit (tick) of the simulation, the manager iterates through all intersections in the warehouse. 3. **Gets a Decision**: For each intersection, it calls the get\_direction() method of the TrafficController instance to obtain a traffic command for that intersection. 4. **Executes the Decision**: Based on the command returned by the controller, the IntersectionManager updates the internal state of the intersection, such as changing the allowed direction of travel. 5. **Triggers Model Training**: Specifically for reinforcement learning-based controllers (DQN/NERL), after the decision and execution steps are completed, the IntersectionManager then calls their train() method, providing the model with the state transition that just occurred (State-Action-Reward-NextState), enabling it to learn and optimize from experience.

**【圖表建議：圖 3.2.1 - 交通控制系統運作序列圖】**

為使讀者能更直觀地理解此運作流程，強烈建議在此處插入一張 **UML 序列圖 (Sequence Diagram)**。該圖應清晰地展示從模擬器主迴圈 (Simulation Loop) 觸發，到 IntersectionManager 遍歷路口，再到 TrafficController 進行決策 (get\_direction)，最後由 IntersectionManager 更新路口狀態 (updateAllowedDirection) 並觸發學習 (train) 的完整訊息傳遞順序。

## 3.2.3 Experimental Hardware and Software Configuration

To ensure the reproducibility and validity of the research results, all simulation, training, and evaluation experiments were conducted on a clearly defined hardware platform and a standardized software environment. This section details the relevant configurations.

### Hardware Configuration

The computational tasks of this research are primarily divided into two parts: model training on a high-performance cloud platform, and development, debugging, and results analysis on a local machine.

#### Training Environment (Runpod Secure Cloud)

All computationally intensive model training tasks were performed on the Runpod cloud platform to leverage its powerful computing resources and accelerate the learning process. - **CPU**: 42 vCPUs - **GPU**: 1 x NVIDIA GeForce RTX 4090 - **Memory (RAM)**: 83 GB

#### Development and Analysis Environment (Laptop)

Program development, initial testing, parameter tuning, and final data analysis and visualization were completed on a local computer. - **CPU**: AMD Ryzen 9 6900HX - **GPU**: NVIDIA GeForce RTX 3080 Ti - **Memory (RAM)**: 16.0 GB

### Software Configuration

The choice of software environment aims to balance development efficiency, computational performance, and broad community support. - **Operating System**: - **Training Environment**: Ubuntu 22.04 (in a runpod/pytorch:2.1.0-py3.10-cuda11.8.0-devel Docker container) - **Development Environment**: Microsoft Windows 11 - **Programming Language**: Python 3.10 - **Core Computing Libraries**: - PyTorch (version 2.1.0): As the core deep learning framework, used to build and train the DQN and NERL neural network models. - numpy (version 1.24.4): Provides the foundation for all numerical computations, widely used in state representation, reward calculation, etc. - pandas (version 2.0.3): Primarily used for processing, storing, and analyzing experimental data. - **Simulation and Analysis Tools**: - networkx (version 3.2.1): Used to construct and analyze the graph structure of the warehouse’s road network. - scikit-learn (version 1.5.2): In this study, mainly used for data preprocessing steps such as data normalization. - **Visualization Libraries**: - matplotlib (version 3.7.2): Used to generate static 2D charts, such as line graphs, bar charts, etc. - seaborn (version 0.13.2): Based on matplotlib, provides more aesthetically pleasing and statistically-oriented visualizations, such as heatmaps. - **System Tools**: - psutil (version 5.9.8): Used to monitor system resource usage.

## 3.2.4 Performance Evaluation Metrics Definition

To objectively and quantitatively evaluate the merits of different traffic control strategies, this study establishes a comprehensive set of Key Performance Indicators (KPIs). For clarity in definition, we first agree on the following mathematical notations: - : The set of all robots in the warehouse. - : The set of all completed orders within the simulation period. - : The set of all events of robots passing through intersections. - : The total simulation duration, in units of .

### 1. Efficiency Metrics

**Total Energy Consumption** This metric measures the overall energy efficiency of the system and is one of the core optimization objectives of this study. It is calculated as the sum of the energy consumed by all robot activities during the simulation period (see Section 3.2.5 for the detailed calculation model).

where represents the total energy consumption of a single robot throughout the simulation, in energy units (EU).

**Average Robot Utilization** This metric reflects the overall business level of the robot fleet and is a dimensionless scalar. It is defined as the average percentage of time all robots are in a non-idle state relative to the total simulation time.

where is the total number of robots, and is the total active time of robot , in .

### 2. Throughput Metrics

**Completed Orders Count** This metric directly measures the total output of the system within a fixed time, reflecting the overall operational efficiency.

**Average Order Processing Time** This metric measures the system’s response speed in processing a single order. It is defined as the average time taken for all completed orders from the start of processing to completion.

where and are the completion and start times of order , respectively, in .

**Average Intersection Waiting Time** This metric directly reflects the coordination efficiency of the traffic control strategy. It calculates the average waiting time for each event of a robot passing through an intersection.

where is the total number of times robots pass through intersections, and is the waiting time for a single passing event , in .

### 3. Stability Metrics

**Total Stop-and-Go Count** This metric reflects the smoothness of the traffic flow. Frequent starting and stopping not only consume extra energy but also indicate unstable traffic flow.

where is the number of stop-and-go events for robot .

## 3.2.5 Robot Physical Model and Energy Consumption

To ensure the realism and credibility of the simulation, this study establishes a robot model based on physical principles. This section details the physical unit system used in the simulation, the core parameters of the robot, and the energy consumption calculation model.

### 1. Unit System Definition

To unify the physical calculations in the simulation, this study adopts the following consistent unit system:

* **Time**: The basic unit of time in the simulation is the . According to the system settings, each corresponds to seconds in the real world.
* **Distance**: One unit of length in the simulation corresponds to meter in the real world.
* **Mass**: The unit of mass in the simulation is kilograms (kg).
* **Velocity**: The unit is meters/second (m/s).
* **Acceleration**: The unit is meters/second squared (m/s²).
* **Energy**: The calculation of energy is based on standard physical formulas. To maintain internal consistency in the simulation, we define its unit as the “Energy Unit (EU),” which is proportional to the Joule.

### 2. Robot Physical Parameters

The following table details the physical parameters applied to all robot entities. These parameters are defined in world/entities/robot.py and remain constant throughout the simulation.

**【Table Suggestion: Table 3.2.1 - Robot Model Physical Parameters】**

| Parameter | Symbol | Value | Unit | Physical Meaning |
| --- | --- | --- | --- | --- |
| Robot Mass |  | 1 | kg | The mass of the robot itself |
| Load Mass |  | 0 | kg | The additional load when the robot is carrying goods |
| Gravity |  | 10 | m/s² | Approximate value of Earth’s gravitational acceleration |
| Friction Coeff. |  | 0.3 | - | Rolling friction between the ground and wheels |
| Inertia Coeff. |  | 0.4 | - | Inertial resistance experienced by the robot during acceleration |
| Startup Cost |  | 0.5 | EU | Extra energy consumed to overcome static friction when starting from rest |
| Regen. Braking Eff. |  | 0.3 | - | The proportion of kinetic energy that can be recovered during braking (30%) |
| Maximum Speed |  | 1.5 | m/s | The maximum speed of the robot under no restrictions |

### 3. Energy Consumption Calculation Model

The energy consumption model in this study not only considers basic movement energy but also integrates startup costs and a regenerative braking mechanism to more realistically reflect the energy characteristics of electric robots. The total energy consumption is calculated by accumulating the energy used in different robot motion states.

**A. Constant Velocity Energy Consumption** When a robot moves at a constant velocity , its energy consumption is mainly from overcoming friction.

where is one time step ( seconds).

**B. Acceleration Energy Consumption** When a robot accelerates with an acceleration , it needs to overcome inertia in addition to friction.

where is the average velocity within that time step.

**C. Startup Cost** When a robot starts moving () from a standstill (), a one-time, fixed startup cost is incurred.

**D. Regenerative Braking (Energy Recovery)** When a robot brakes to a complete stop () from a moving state (), the system recovers a portion of the energy based on its change in kinetic energy.

**E. Total Energy Change** Combining the above items, the total energy change for a robot in a single time step can be expressed as:

where corresponds to or depending on whether the robot is moving at a constant velocity or accelerating. The total energy consumption of the system is the cumulative sum of for all robots over all time steps.

## 3.3 Baseline Controller Design

To objectively and rigorously evaluate the performance of the deep reinforcement learning traffic control method proposed in this study (detailed in Section 3.4), it must be compared against a set of representative and easily understandable baseline controllers. Baseline controllers provide a performance reference point, enabling us to quantify the actual degree of improvement brought by complex algorithms. An ideal baseline should reflect current industry practices or intuitive solutions.

This study selects two different but representative logics to design the baseline controllers: one is a fixed-time controller that completely ignores real-time traffic conditions, and the other is a dynamic controller that reacts to the immediate demands at the intersection. These two represent the fundamental forms of static and dynamic control strategies, respectively, and can comprehensively measure the adaptability and superiority of the reinforcement learning model in different traffic scenarios.

This section will sequentially detail the internal design principles, decision logic, and key parameters of the following two baseline controllers:

1. **Time-Based Controller**: A static controller based on switching right-of-way at fixed time cycles.
2. **Queue-Based Controller**: A dynamic, reactive controller that makes decisions based on the length of aTaiting queues at the intersection and the priority of tasks.

## 3.3.1 Time-Based Controller

The TimeBasedController is the most fundamental static traffic control strategy. Its core idea originates from traditional urban traffic signal systems, completely disregarding real-time traffic flow or any dynamic changes at the intersection. It relies solely on a pre-set, fixed time cycle to alternate the right-of-way between horizontal and vertical directions. The advantage of this method lies in its extreme simplicity and predictability, but its disadvantage is equally apparent—it cannot adapt to fluctuations in traffic demand, easily causing unnecessary waiting during busy periods or wasting green time during light traffic.

### Design Principle and Decision Logic

The operation of this controller is determined entirely by three parameters: the green light time for the horizontal direction (), the green light time for the vertical direction (), and the total signal cycle length (), which is the sum of the two.

At any given time tick in the simulation, the controller determines the current position in the signal cycle () using a modulo operation:

Based on the value of , the controller makes a decision on the direction of passage. If is less than the horizontal green time , the right-of-way is given to the horizontal direction; otherwise, it is given to the vertical direction. The decision rule can be expressed as:

In our warehouse environment, since pods are primarily arranged along the horizontal direction, the frequency and volume of robot movement are much higher horizontally than vertically. To accommodate this characteristic, we assign a longer green time to the horizontal direction in our parameter settings (e.g., ticks), while the vertical green time is relatively shorter (e.g., ticks), aiming to achieve a preliminary traffic balance without considering real-time status.

**【圖表建議：圖 3.3.1 - 時間基礎控制器訊號週期示意圖】**

建議在此處插入一張時間軸圖，清晰地展示 的構成，並標示出 和 的區間，以及在不同區間內對應的通行方向決策（Horizontal/Vertical）。

## 3.3.2 Queue-Based Controller

The QueueBasedController is a dynamic, reactive control strategy designed to address the fundamental shortcoming of the time-based controller, which is its inability to perceive real-time traffic demand. This controller continuously monitors the waiting queues in both directions of an intersection and dynamically calculates the right-of-way based on the urgency of the tasks being performed by the robots. Compared to the static time-based controller, the queue-based controller can more flexibly adapt to changes in traffic flow, prioritizing the allocation of passage resources to the direction with the more urgent demand.

### Design Principle and Decision Logic

The core of this controller lies in quantifying and combining two key factors for decision-making: **the number of robots** and **task priority**.

#### 1. Task Priority Weighting System

In warehouse operations, not all robot tasks are of equal importance. For example, a robot delivering a pod to a picking station, if delayed, will directly impact order fulfillment efficiency. In contrast, an empty robot heading to the storage area has a relatively lower task urgency. To reflect this difference, we have defined a set of priority weights () for different robot states (robot.current\_state):

| Robot State (current\_state) | Task Description | Priority Weight () |
| --- | --- | --- |
| delivering\_pod | Delivering a pod to a picking station | 3.0 |
| returning\_pod | Returning a pod to the storage area | 2.0 |
| taking\_pod | Going to the storage area to pick up a pod | 1.0 |
| idle | Idle or on standby | 0.5 |
| station\_processing | Processing at a picking station | 0.0 |

#### 2. Direction Priority Calculation

For each direction at an intersection (Horizontal H or Vertical V), the controller calculates a weighted priority sum ( or ). This value is the sum of the task priority weights of all waiting robots in that direction (). The formula is as follows:

where represents the priority weight corresponding to the current state of robot .

Furthermore, considering the inherent higher traffic flow in the horizontal direction due to the warehouse layout, we introduce a bias factor (), to weight the priority sum of the horizontal direction, giving it an additional competitive advantage. Therefore, the final horizontal priority used for comparison is:

#### 3. Decision Process

The controller’s decision process is as follows: 1. **Minimum Green Time Check**: To prevent frequent signal changes at the intersection due to rapid changes in traffic conditions (which would cause robots to repeatedly accelerate and decelerate, wasting energy), the controller first checks if a minimum green time () has passed since the last direction change. If not, the current direction is maintained. 2. **Priority Comparison**: If the minimum green time has been met, the controller calculates the weighted horizontal priority and the vertical priority . 3. **Special Case Handling**: - If there are no waiting robots in one direction, the right-of-way is immediately given to the direction with robots. - If there are no robots in either direction, the current state is maintained. 4. **Final Decision**: In general cases, the controller compares and and assigns the right-of-way to the direction with the higher priority sum.

Through this mechanism, the queue-based controller can make reasonable and efficient responses to real-time traffic demands while considering stability (minimum green time) and layout characteristics (bias factor).

## 3.3.3 Baseline Controller Parameter Settings

To ensure the validity and reproducibility of the experiments, this study clearly defines and standardizes the parameters used by the two baseline controllers. These parameters were selected based on preliminary experiments and empirical experience, aiming to enable the controllers to exhibit reasonable and stable performance in general scenarios.

### 1. Time-Based Controller (TimeBasedController)

The logic of this controller is entirely driven by a fixed time cycle, and its parameter settings are as follows:

| Parameter Name | Default Value | Unit | Description |
| --- | --- | --- | --- |
| horizontal\_green\_time | 70 | ticks | The duration of the green light for the horizontal direction. Due to the warehouse layout, the horizontal main roads carry heavier east-west traffic flow, thus are given a longer passage time. |
| vertical\_green\_time | 30 | ticks | The duration of the green light for the vertical direction. |
| **Total Cycle Length** | **100** | **ticks** | **A complete signal cycle (70 + 30).** |

### 2. Queue-Based Controller (QueueBasedController)

This controller makes decisions based on real-time traffic conditions, and its parameters involve the sensitivity of its decisions and its preference for different tasks.

| Parameter Name | Default Value | Unit/Type | Description |
| --- | --- | --- | --- |
| min\_green\_time | 1 | ticks | Minimum green time. A very small value is set to prevent the signal from “oscillating” too frequently between two conflicting requests, while still maintaining the controller’s rapid response to traffic changes. |
| bias\_factor | 1.5 | float (multiplier) | Horizontal direction preference factor. This multiplier is applied to the calculated weighted queue result for the horizontal direction to compensate for the naturally higher traffic flow on the horizontal main roads, preventing the vertical direction from too frequently preempting the right-of-way due to a small number of high-priority robots. |
| priority\_weights | {"delivering\_pod": 3.0, "returning\_pod": 2.0, "taking\_pod": 1.0, "idle": 0.5} | Dictionary | Task priority weights. This dictionary defines the importance of robots in different task states. For example, a robot delivering goods to complete an order (delivering\_pod) has a weight 6 times that of an idle (idle) robot. |

## 3.4 Deep Reinforcement Learning Controller Design

To address the highly dynamic and complex nature of traffic within an RMFS, this study employs Deep Reinforcement Learning (DRL) to develop intelligent traffic controllers. DRL combines the powerful feature extraction capabilities of deep learning with the decision-making optimization framework of reinforcement learning, enabling an agent to learn effective control policies directly from high-dimensional raw sensory data.

Traditional traffic control methods, whether fixed-phase or rule-based heuristic algorithms, often struggle to achieve global optimality when faced with complex and non-linear traffic patterns. The DRL approach offers a more adaptive solution, where the controller (agent) can autonomously learn a policy that maximizes long-term cumulative rewards through continuous trial-and-error interactions with the simulation environment, without relying on hand-crafted complex rules.

This section will detail the design of two DRL controller architectures for this study:

1. **Deep Q-Network (DQN)**: As a mature and widely used DRL algorithm, DQN serves as a strong **baseline for comparison** to measure the relative performance of the new method proposed in this study.
2. **Neuroevolution Reinforcement Learning (NERL)**: This is the **core contribution** of this research. This method combines the policy representation of neural networks with the global search capabilities of evolutionary algorithms, aiming to overcome challenges that traditional DRL methods may face, such as training instability and sample inefficiency.

The following subsections will first describe the model architecture and operational principles of DQN and NERL respectively, followed by a detailed definition of the common state space, action space, and reward function design, which are the core elements constituting a DRL problem.

## 3.4.1 Deep Q-Network (DQN) Controller Design

The Deep Q-Network (DQN) is the **baseline for comparison** method used in this study to establish a traffic control policy. As a foundational algorithm in the field of deep reinforcement learning, DQN combines deep neural networks with classic Q-Learning, enabling it to handle high-dimensional state spaces. DQN was chosen as a baseline to evaluate the improvements offered by the proposed NERL method within a recognized and stable DRL framework.

The core idea of DQN is to learn an action-value function, the Q-function. This function, , is approximated using a neural network parameterized by . Its goal is to predict the expected cumulative reward for taking action in a given state . The optimal Q-function, , follows the Bellman Optimality Equation:

where is the immediate reward, is the discount factor representing the importance of future rewards, and is the successor state. Once an accurate Q-function is learned, the agent can execute an optimal policy in any state by selecting the action that maximizes .

### 1. Core Stability Mechanisms

To address the training instability that can arise when using non-linear function approximators like neural networks, the DQN architecture adopted in this study integrates two key techniques:

* **Experience Replay**: Transition samples generated from the controller’s interaction with the environment are stored in a fixed-size memory buffer . During the training phase, the algorithm randomly samples a minibatch of these experiences for learning, rather than using sequential time-series samples. This practice breaks the temporal correlations between samples, making the training data more closely resemble an independent and identically distributed (i.i.d.) assumption, which significantly improves training stability.
* **Target Network**: The algorithm maintains two separate neural networks. One is the **Policy Network**, with parameters , used to select actions at each time step. The other is the **Target Network**, with parameters . When calculating the Temporal Difference (TD) target, the target Q-value is computed by the target network, i.e., . The parameters of the target network are periodically copied from the policy network’s parameters (), rather than every step. This delayed update mechanism decouples the dependency between the target Q-value and the current Q-value, effectively suppressing oscillations and divergence that can occur during bootstrapping.

### 2. Learning Process

DQN is trained by minimizing the loss function of a randomly sampled batch of transition experiences. The loss function is defined as the Mean Squared Error (MSE) between the TD target and the output of the policy network:

The gradient of this loss function is used to update the weights of the policy network via Stochastic Gradient Descent (SGD) or its variants (like Adam).

### 3. Neural Network Architecture

The policy and target networks used in the DQN controller of this study are both Feed-Forward Neural Networks. The input layer dimension corresponds to the state space defined in **Section 3.4.3**, and the output layer dimension corresponds to the action space defined in **Section 3.4.4**. The network includes two hidden layers and uses ReLU as the activation function, with its specific architecture more clearly defined in **Section 3.5.3**.

**【圖表建議：圖 3.4.1 - DQN 訓練與決策流程圖】**

建議在此處繪製一張圖，清晰地展示 DQN 的完整運作流程。圖中應包含以下兩個並行的循環： 1. **決策循環 (Agent-Environment Interaction)**: \* 從環境接收狀態 。 \* 策略網路 接收 並輸出所有動作的 Q 值。 \* 使用 -greedy 策略選擇動作 。 \* 在環境中執行 ，獲得獎勵 和新狀態 。 \* 將轉換樣本 存入經驗回放記憶體 。 2. **訓練循環 (Network Update)**: \* 從 中隨機採樣一批轉換樣本。 \* 使用目標網路 計算 TD 目標。 \* 計算策略網路 的損失函數。 \* 執行梯度下降以更新策略網路的權重 。 \* 定期將策略網路的權重複製到目標網路 ()。

## 3.4.2 Neuroevolution Reinforcement Learning (NERL) Controller Design

Neuroevolution Reinforcement Learning (NERL) is the core innovative method proposed in this study. This approach combines the global search capabilities of Evolutionary Algorithms (EAs) with the non-linear function approximation abilities of neural networks, aiming to overcome challenges that traditional gradient-based DRL methods (like DQN) may encounter when dealing with sparse rewards and complex parameter spaces, such as unstable convergence or getting stuck in local optima.

Unlike DQN, which seeks to approximate a value function, the goal of neuroevolution is to directly optimize in the **parameter space** of a policy. In the NERL framework, the weights and biases of a neural network controller (i.e., a policy ) are treated as an individual’s **genotype**. The algorithm directly searches for the optimal policy parameters by applying evolutionary operations to a **population** of many individuals.

### 1. Evolutionary Workflow

The core process of NERL revolves around an “evaluate → select → reproduce” evolutionary cycle, which iterates to improve the overall performance of the population. Assume at generation , there is a population , containing individuals.

1. **Evaluation**: Each individual in the population is deployed in a separate instance of the simulation environment to perform a full evaluation episode. During the episode, the policy makes all decisions independently. After the episode concludes, the individual’s **Fitness Score** is calculated based on the system’s macroscopic performance metrics (i.e., the global reward defined in **Section 3.4.5**). This process is highly parallelizable, which can significantly shorten training time.
2. **Selection**: Based on the fitness scores calculated for all individuals, the algorithm selects high-performing individuals from the current population as **parents** to provide genes for the next generation. This study employs **Tournament Selection**, a mechanism that strikes a good balance between selection pressure and maintaining population diversity.
3. **Reproduction**: A new offspring population is generated by applying genetic operations to the parents. The main operations include:
   * **Elitism**: To prevent the loss of the best solutions found during the evolutionary process, the top elite individuals with the highest fitness in each generation are directly and completely copied to the next generation’s population.
   * **Crossover**: Simulating biological reproduction, two individuals and are selected from the parents, and their parameter vectors are mixed to create new offspring. This study uses **Uniform Crossover**.
   * **Mutation**: After crossover, a small random perturbation is applied to the parameters of the offspring individuals. This operation is key to maintaining population diversity and avoiding premature convergence. This study uses **Gaussian Mutation**.

By iteratively executing the above cycle, the average fitness of the population will steadily increase, eventually converging to a high-performance policy network capable of efficiently solving complex traffic control problems.

### 2. Neural Network Architecture and Parameters

To ensure a fair and consistent comparison with the DQN baseline, the NERL controller adopts the exact same neural network architecture as DQN. Its evolutionary process is governed by a series of key hyperparameters (such as population size, mutation rate, and elitism ratio), the specific values of which for this experiment are detailed in the experimental configuration in **Section 3.6.2**.

**【Graphic Suggestion: Figure 3.4.1 - NERL Evolutionary Cycle Diagram】**

It is recommended to draw a circular diagram here to illustrate the core workflow of NERL. The diagram should include: 1. **Initial Population** : Shows multiple neural network individuals . 2. **Parallel Evaluation**: Each runs in a separate environment, and its fitness is calculated. 3. **Selection**: Parents are selected based on fitness scores through tournament selection. 4. **Reproduction to Generate** : Shows elitism, crossover, and mutation operations to create the new generation population. 5. Arrows should clearly point to the next stage, a complete evolutionary closed loop from to .

## 3.4.3 State Space Design

The definition of the state space is a cornerstone for the success of reinforcement learning. It must provide the agent with sufficient and meaningful information to make effective decisions, while avoiding the “Curse of Dimensionality” caused by excessively high dimensions. In the RMFS traffic control problem of this study, the state of any intersection at a decision moment is defined as a feature vector that comprehensively reflects its local traffic conditions and also considers the potential impact of surrounding and downstream intersections.

Specifically, is a **17-dimensional continuous vector**, composed as follows:

**1. Local State - 8 dimensions**: Describes the real-time traffic information of the intersection where the controller is located. \* **Horizontal Direction**: \* : **Queue Length** - The number of robots waiting to pass through the intersection on the horizontal road. \* : **First Vehicle Waiting Time** - If the queue is not empty, this is the time the first robot in the queue has been waiting; otherwise, it is 0. \* : **Average Waiting Time** - The average waiting time of all waiting robots in the horizontal direction. \* : **Downstream Saturation** - The queue length of the next intersection in the horizontal direction, used to predict the potential risk of spillback. \* **Vertical Direction**: \* - : The same four features for the vertical road.

**2. Neighbor State - 8 dimensions**: Describes key information from the four neighboring intersections directly connected to the current one, helping the agent to understand the broader traffic situation. \* **Each of the four neighbors (Up, Down, Left, Right)** provides 2 dimensions of information: \* : **Upstream Neighbor** - Vertical queue length, horizontal queue length. \* : **Downstream Neighbor** - Vertical queue length, horizontal queue length. \* : **Left Neighbor** - Vertical queue length, horizontal queue length. \* : **Right Neighbor** - Vertical queue length, horizontal queue length.

**3. Global State - 1 dimension**: Introduces a macroscopic indicator to help the agent align local decisions with overall system goals. \* : **Average Picking Station Queue** - The average queue length at the entrance of all picking stations. This is a critical system-level indicator, as picking stations are the ultimate bottleneck for the entire material flow.

### State Normalization

Since the 17 features described above have different physical units and numerical ranges (e.g., counts, time, ratios), feeding them directly into a neural network would cause features of different scales to have varying impacts on the gradient, leading to an unstable training process. Therefore, before feeding the state vector into the DRL model, this study employs an **Adaptive Normalization** technique. This technique dynamically tracks the running mean and standard deviation of each state feature during training and uses them to standardize the feature values to a distribution with a mean close to 0 and a standard deviation close to 1, thereby ensuring the stability and efficiency of the model training.

## 3.4.4 Action Space Design

In contrast to the high-dimensional state space, this study designs a concise and intuitive discrete action space for the DRL agent. At each decision point, the controller can select one action from a set of **6 discrete actions**. These actions not only cover basic traffic signal phase control but also introduce the ability to dynamically adjust local speed limits for more refined traffic flow management.

The action set is defined as , where each integer corresponds to a specific control command:

### 1. Basic Phase Control

This part of the actions is mainly responsible for managing the right-of-way at the intersection.

* **Action 0 (KEEP\_CURRENT\_PHASE)**: **Maintain the current phase**. This action keeps the current signal state of the intersection unchanged. This is the optimal choice when the existing traffic flow is smooth, or when the cost of changing the phase is higher than the potential benefit.
* **Action 1 (SWITCH\_TO\_VERTICAL\_GREEN)**: **Switch to vertical green**. This action forces the signal phase to a vertical green light and a horizontal red light, aiming to relieve traffic pressure in the vertical direction.
* **Action 2 (SWITCH\_TO\_HORIZONTAL\_GREEN)**: **Switch to horizontal green**. This action forces the signal phase to a horizontal green light and a vertical red light, aiming to relieve traffic pressure in the horizontal direction.

### 2. Dynamic Speed Control

To more proactively manage traffic flow and avoid cascading spillbacks at bottleneck intersections, the model can also execute the following actions to dynamically adjust the speed limit of robots leaving the intersection. These actions do not change the signal phase themselves.

* **Action 3 (SET\_SPEED\_NORMAL)**: **Set speed to normal**. This restores the speed limit for robots departing from this intersection to the default value (1.0), typically used after traffic conditions have eased.
* **Action 4 (SET\_SPEED\_SLOW)**: **Set speed to slow**. This reduces the speed limit to a slow speed (0.5), used to proactively smooth traffic fluctuations when potential downstream congestion is anticipated.
* **Action 5 (SET\_SPEED\_VERY\_SLOW)**: **Set speed to very slow**. This reduces the speed limit to a very slow speed (0.2), used to minimize the flow into a congested area when severe downstream congestion has already occurred.

### Decision Interval

All DRL controllers operate at a fixed decision interval of ticks. This means that the controller evaluates and selects a new action based on the current state every 10 simulation time steps. During this interval, the intersection will maintain the signal phase and speed limit set by the previous decision.

## 3.4.5 Reward Function Design

The reward function is the core mechanism in reinforcement learning that guides the agent’s behavior, translating the desired system goals into an observable and quantifiable scalar feedback signal. To explore the effects of learning at different time scales, this study designs two distinct reward modes: “Step Reward” and “Global Reward.”

### 1. Step Reward

The step reward mode aims to provide the agent with a dense, immediate, and local feedback signal. At the end of each decision interval ( ticks), the system independently evaluates the local performance of each intersection and calculates a composite reward value. This high-frequency feedback helps the agent to quickly learn basic traffic control heuristics, such as prioritizing high-priority tasks and reducing waiting vehicles.

For a single intersection , its step reward within a decision interval is composed of the following weighted components:

where the components are defined as follows:

* **Flow Reward ()**: A positive reward given based on the number of robots that successfully pass through the intersection and their task priorities.
* Here, is the set of robots that pass through intersection during the decision interval, is the task priority of robot (high, medium, low), and is the corresponding priority reward weight.
* **Waiting Cost ()**: A penalty imposed on robots still waiting in the queue at the intersection, based on their accumulated waiting time and task priority.
* Here, is the set of robots still waiting at intersection , is the waiting time of robot (in ticks), and is the corresponding priority penalty weight.
* **Phase Switch Cost ()**: A fixed penalty for each signal phase switch to avoid overly frequent and inefficient changes, encouraging the controller to maintain the continuity of traffic flow.
* where is an indicator function that is 1 if a phase switch occurred and 0 otherwise. is a fixed penalty weight.
* **Critical Intersection Weighting ()**: To enable the agent to prioritize learning to manage traffic in bottleneck areas, such as those near picking stations, the reward value for an intersection marked as “critical” is multiplied by an amplification factor greater than 1.

### 2. Global Reward

The global reward mode provides a sparse, delayed feedback signal aimed at guiding the agent to learn complex strategies that are beneficial to the system’s long-term, macroscopic goals. In this mode, the agent receives no immediate feedback throughout the entire evaluation episode (), and a single reward value is calculated only at the end of the episode based on the final overall performance of the system.

To avoid the reward signal being dominated by a single metric due to direct addition/subtraction of indicators with different scales, this study designs a global reward function based on an efficiency ratio. This function places the system’s “output” as the numerator and the system’s “cost” as the denominator. Its formula is defined as follows:

where the symbols are defined as follows:

* : The total number of orders completed during the evaluation episode (unit: count).
* : The reward weight for order completion.
* : The total energy consumption of the system during the episode (unit: EU), as detailed in Section 3.2.5.
* : An energy scaling factor used to adjust the energy cost to a comparable scale with the time cost.
* : The total duration of the evaluation episode (unit: ticks).
* : The time penalty weight per tick.
* : A large penalty applied if severe spillback occurs at a picking station.
* : A very small positive constant (e.g., ) to avoid division by zero.

This ratio-based design encourages the agent to pursue a high number of completed orders while simultaneously considering energy and time efficiency, thereby learning a more balanced and sustainable operational strategy.

## 3.5 Experimental Design and Evaluation Methodology

To objectively and quantitatively answer the central research question of this study—whether our proposed deep reinforcement learning traffic control strategy offers a significant advantage over traditional methods in improving warehouse system operational efficiency—this section details the overall experimental design, model training procedures, performance evaluation framework, and the statistical analysis methods for the results.

A rigorous experimental design is the cornerstone for ensuring the reliability of research conclusions. To this end, we will establish a comparative experimental matrix covering multiple control strategies and test them under a unified simulation environment and system load. All experiments will follow standardized training and evaluation procedures to eliminate the interference of irrelevant variables, ensuring that comparisons between different algorithms are fair and meaningful.

The structure of this section is as follows: - **3.5.1 Training Configuration and Experimental Groups**: Defines all controller types included in the experiment (experimental groups) and describes the hardware and software environment configurations used for training. - **3.5.2 Model Training Procedures**: Details the specific training steps for the DRL models (DQN and NERL) to ensure the reproducibility of the experiments. - **3.5.3 Evaluation Method and Comparison Framework**: Describes the standardized procedure used for final performance evaluation after model training is complete, and the unified framework for comparing all experimental groups. - **3.5.4 Statistical Analysis and Results Validation**: Explains the statistical methods that will be used to analyze the experimental data to scientifically verify the significance of performance differences between different strategies.

## 3.5.1 Experimental Design and Group Definition

To systematically evaluate the performance of different traffic control strategies, this study designs a comparative experiment consisting of twelve independent experimental groups. This design aims to comprehensively compare the performance of the two deep reinforcement learning methods proposed in this study (DQN and NERL), under different reward modes and hyperparameter configurations, against two heuristic baseline controllers.

### Experimental Group Definition

All experimental groups run in the standardized warehouse simulation environment described in **Section 3.2.1**, with the only variable being the traffic controller used at the intersections and its specific configuration. The detailed definitions of each experimental group are shown in the table below.

**Table 3.5.1: Experimental Group Definition and Description**

| Group | Controller Type | Reward Mode | NERL Variant | NERL Evaluation Duration (ticks) | Category | Description |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | TimeBased | - | - | - | Baseline | Static controller with fixed time cycles |
| 2 | QueueBased | - | - | - | Baseline | Dynamic reactive controller based on real-time queue length |
| 3 | DQN | step | - | - | DRL | DQN trained with step-wise rewards (see 3.5.3 for params) |
| 4 | DQN | global | - | - | DRL | DQN trained with a global reward (see 3.5.3 for params) |
| 5 | NERL | step | **A (Exploratory)** | 3,000 | DRL | High mutation rate, short evaluation duration (see 3.5.3) |
| 6 | NERL | global | **A (Exploratory)** | 3,000 | DRL | High mutation rate, short evaluation duration (see 3.5.3) |
| 7 | NERL | step | **B (Exploitative)** | 3,000 | DRL | Low mutation rate, short evaluation duration (see 3.5.3) |
| 8 | NERL | global | **B (Exploitative)** | 3,000 | DRL | Low mutation rate, short evaluation duration (see 3.5.3) |
| 9 | NERL | step | **A (Exploratory)** | 8,000 | DRL | High mutation rate, long evaluation duration (see 3.5.3) |
| 10 | NERL | global | **A (Exploratory)** | 8,000 | DRL | High mutation rate, long evaluation duration (see 3.5.3) |
| 11 | NERL | step | **B (Exploitative)** | 8,000 | DRL | Low mutation rate, long evaluation duration (see 3.5.3) |
| 12 | NERL | global | **B (Exploitative)** | 8,000 | DRL | Low mutation rate, long evaluation duration (see 3.5.3) |

### NERL Variant Parameter Details

To investigate the impact of the balance between “Exploration” and “Exploitation” during the evolutionary process on the final policy performance, this study designs two NERL variants with different evolutionary hyperparameters. The core difference lies in the mutation operation settings:

* **Variant A (Exploratory)**: This configuration aims to promote a broad search in the parameter space. It features a higher mutation rate (mutation\_rate = 0.3) and a larger mutation strength (mutation\_strength = 0.2). This gives offspring individuals greater potential to jump out of the neighborhood of existing solutions and discover entirely new, potentially better policies, but it may also risk slower convergence.
* **Variant B (Exploitative)**: This configuration focuses on fine-tuning already discovered superior solutions. It uses a lower mutation rate (mutation\_rate = 0.1) and a smaller mutation strength (mutation\_strength = 0.05). This conservative mutation strategy helps with stable policy convergence but may also increase the risk of getting stuck in a local optimum.

Furthermore, to study the effect of the adequacy of individual policy evaluation on learning outcomes, each NERL variant will be trained and evaluated under two different evaluation durations: 3,000 ticks and 8,000 ticks.

### Hardware and Software Configuration

To ensure the consistency and reproducibility of the experimental results, all experiments were conducted in a standardized environment. Detailed hardware and software information has been provided in **Section 3.2.3**.

## 3.5.2 Model Training Procedures

To ensure that the DRL models can fully learn and converge to an optimal policy, and to guarantee fairness in comparisons between different models, this study designs a standardized model training procedure. This procedure details every step from model initialization to final model saving.

### 1. DQN Training Procedure (for Groups 3, 4)

The training of DQN is an on-policy, continuous learning process. A single complete DQN training experiment follows this procedure:

1. **Initialization**:
   1. Create a DQNController instance based on the hyperparameters defined in 3.4.4 and the reward mode (step or global) specified by the experimental group.
   2. Create an instance of the simulation warehouse environment, Warehouse.
2. **Training Loop**:
   1. Start a simulation that lasts for N = 550,000 time steps (ticks).
   2. At each time step t, the IntersectionManager iterates through all intersections.
   3. For each intersection i:
      1. The controller gets the current state s\_t from the environment.
      2. An action a\_t is selected using the policy network and an ε-greedy strategy.
      3. The action a\_t is executed, the environment transitions to the next state s\_{t+1}, and the UnifiedRewardSystem calculates the immediate reward r\_t (this reward is 0 in global mode).
      4. The experience tuple (s\_t, a\_t, r\_t, s\_{t+1}) is stored in the experience replay memory.
   4. **Experience Replay**: Every k=32 time steps, a batch of experiences is randomly sampled from the memory for learning.
   5. **Target Network Update**: Every M=1,000 time steps, the weights of the policy network are copied to the target network.
3. **Model Saving**: After the training is fully completed, the final weights of the policy network are saved as the final model.

### 2. NERL Training Procedure (for Groups 5-12)

The training of NERL is a generation-iterative, off-policy learning process. Its core procedure is uniform for all NERL groups but incorporates different hyperparameters based on the specific group’s configuration.

1. **Initialization**:
   1. Create a NEController instance based on the definitions in 3.4.4 and the **specific experimental group** (5-12). This step determines the following key hyperparameters:
      * **Reward Mode**: step or global.
      * **Mutation Variant**: **A (Exploratory)** or **B (Exploitative)**, which determines the values of mutation\_rate and mutation\_strength.
      * **Evaluation Ticks**: 3,000 or 8,000.
   2. The controller randomly initializes a population of 20 network individuals.
2. **Evolutionary Loop**:
   1. Start an evolution that lasts for G = 30 Generations.
   2. In each generation g:
      1. **Parallel Evaluation**: For the 20 individuals in the population, 20 independent, parallel simulation environments are launched.
      2. Each individual j runs a full evaluation episode in its dedicated environment. The duration of the episode is determined by the eval\_ticks parameter of that experimental group (3,000 or 8,000 ticks).
      3. After the episode ends, the UnifiedRewardSystem calculates the fitness score f\_j for individual j based on the reward mode (step or global) specified for that experimental group.
      4. **Evolutionary Operations**: Once the fitness scores for all individuals have been calculated, the controller performs a full evolutionary operation (selection, crossover, mutation) based on the mutation configuration of that experimental group (A or B) to generate a new offspring population.
      5. The new offspring population becomes the starting population for the next generation g+1.
3. **Model Saving**:
   1. At the end of each generation, the algorithm saves the individual with the highest fitness in that generation as the best model of the generation.
   2. After all 30 generations of evolution are complete, the model with the highest fitness score from all generations’ best models is selected and saved as the final model for that experimental group.

## 3.5.3 DRL Model Hyperparameter Settings

To ensure the reproducibility and validity of the DRL experiments in this study, this section details the key hyperparameters used in training the DQN and NERL controllers. These parameters were set based on preliminary convergence and stability experiments and remained fixed during the formal training period.

### 1. Common Neural Network Architecture

To ensure a fair comparison between the baseline (DQN) and the core method (NERL), both adopted the exact same neural network architecture. This architecture strikes a balance between the model’s expressive power and computational efficiency, and is sufficient to handle the traffic control problem in this study.

| Layer | Type | Input Dim | Output Dim | Activation |
| --- | --- | --- | --- | --- |
| 1 | Input | 17 | 17 | - |
| 2 | Fully Connected (FC 1) | 17 | 128 | ReLU |
| 3 | Fully Connected (FC 2) | 128 | 64 | ReLU |
| 4 | Output | 64 | 6 | - |

### 2. DQN-Specific Hyperparameters

The following table lists the main hyperparameters used during the training of the DQN controller (experimental groups 3, 4).

| Hyperparameter | Code Variable | Value | Description |
| --- | --- | --- | --- |
| Learning Rate | learning\_rate | 5e-4 | The learning rate for the Adam optimizer. |
| Gamma | gamma | 0.99 | The discount factor for future rewards. A value closer to 1 means more emphasis on long-term returns. |
| Initial Epsilon | epsilon | 1.0 | The probability of choosing a random action at the beginning of training. |
| Minimum Epsilon | epsilon\_min | 0.01 | The lower bound for epsilon decay. |
| Epsilon Decay Rate | epsilon\_decay | 0.9995 | The exponential decay rate by which epsilon is multiplied after each training step. |
| Experience Replay Memory Size | memory\_size | 50,000 | The maximum number of transition samples to store. |
| Batch Size | batch\_size | 8,192 | The number of samples to draw from memory for each network update. |
| Target Network Update Frequency | target\_update\_freq | 1,000 | The frequency (in training **steps**) at which the policy network’s weights are copied to the target network. |

### 3. NERL-Specific Hyperparameters

The following table lists the main hyperparameters used during the evolution of the NERL controller (experimental groups 5-12). The mutation rate and mutation strength differ according to the **Exploratory (A)** and **Exploitative (B)** variants defined in **Section 3.6.1**.

| Hyperparameter | Code Variable | Variant A (Exploratory) | Variant B (Exploitative) | Description |
| --- | --- | --- | --- | --- |
| Population Size | population\_size | 20 | 20 | The number of individuals (neural networks) in each generation. |
| Elite Ratio | elite\_ratio | 0.2 | 0.2 | The proportion of the fittest individuals directly preserved in each generation. |
| Tournament Size | tournament\_size | 4 | 4 | The number of individuals randomly compared in each tournament selection. |
| Crossover Rate | crossover\_rate | 0.8 | 0.8 | The probability of performing genetic crossover between two parent individuals. |
| Mutation Rate | mutation\_rate | **0.2** | **0.1** | The base probability of an individual’s genes (network weights) undergoing mutation. |
| Mutation Strength | mutation\_strength | **0.15** | **0.15** | The standard deviation of the Gaussian mutation, controlling the magnitude of the mutation. |
| Evaluation Ticks | eval\_ticks | 3000 / 8000 | 3000 / 8000 | The duration (in ticks) for evaluating each individual. |

These hyperparameters collectively define the learning behavior of the two DRL methods and are an important basis for the subsequent experimental analysis and results comparison.

## 3.5.4 Evaluation Method and Comparison Framework

After the DRL models are trained according to the procedure described in Section 3.5.2, a standardized evaluation process is designed to ensure that all controllers are compared on a fair and unbiased basis. This process aims to simulate a fixed, reproducible “test day” scenario and to quantify the performance of each control strategy using a predefined set of Key Performance Indicators (KPIs).

### 1. Standardized Evaluation Procedure

For each experimental group defined in Section 3.5.1 (including the two baseline controllers and the four trained DRL controllers), we will execute the following standardized evaluation procedure:

1. **Model Loading**: For the DRL experimental groups, load their corresponding final trained models and set them to pure **Inference Mode**. In this mode, the ε-greedy exploration of DQN will be turned off (ε=0), and NERL will consistently use its historically fittest individual for decision-making.
2. **Environment Reset**: Initialize a simulation environment that is identical to the one used during training but with a **fixed set of random seeds that were never used in training**. This ensures that all controllers face the exact same initial conditions, order sequences, and random events, thus eliminating interference from randomness.
3. **Execution of Evaluation**: In the standardized environment, run a complete simulation of a fixed duration (e.g., T\_{eval} = 50,000 ticks).
4. **Data Recording**: During the simulation, the PerformanceReportGenerator will record time-series data of all key performance indicators at fixed intervals (e.g., every 10 ticks).
5. **Repeated Execution**: To eliminate the impact of extreme random events in a single run and to obtain more statistically significant results, the above steps 2-4 will be **repeated 3 times** for each experimental group, each time using a different but shared set of random seeds. The final performance will be the average of these 3 runs.

### 2. Performance Comparison Framework

After the 3 repeated evaluation runs are completed, we will aggregate and compare the data for each experimental group. The comparison framework will revolve around the Key Performance Indicators defined in Section 3.2.4, which can be categorized into three main types:

**A. Core Efficiency Indicators** These indicators directly reflect the system efficiencies that are of most interest in this study. - **Total Energy Consumption**: Evaluates the energy efficiency of the system; the lower, the better. - **Completed Orders Count**: Measures the total output of the system within a fixed time; the higher, the better.

**B. Process Quality Indicators** These indicators indirectly reflect the smoothness of system operation and service quality. - **Average Order Processing Time**: Measures the system’s response speed; the lower, the better. - **Average Intersection Waiting Time**: Directly measures the effectiveness of the traffic control strategy; the lower, the better. - **Total Stop-and-Go Count**: Reflects the smoothness of traffic flow; the lower, the better.

**C. Resource Utilization Indicators** - **Average Robot Utilization**: Reflects the degree to which robot resources are utilized in achieving the output.

Finally, the average KPI data for all experimental groups will be compiled into a comprehensive performance comparison table for in-depth analysis and discussion in the next chapter. Additionally, the collected time-series data will be plotted into charts to visually demonstrate the dynamic behavior differences between the different strategies during the simulation.

## 3.5.5 Statistical Analysis and Results Validation

To ensure that the conclusions of this study are based on a more rigorous statistical foundation rather than just descriptive mean comparisons, we will use Hypothesis Testing to verify whether the observed performance differences between different control strategies are statistically significant. This step is crucial for distinguishing between genuine performance improvements and chance results caused by random fluctuations.

### 1. Hypothesis Testing Method

Since we have conducted 3 independent repeated evaluations for each experimental group, this provides us with multiple sets of sample data. When comparing the performance of two different experimental groups (e.g., DQN-step vs. QueueBased) on a specific performance indicator (e.g., total energy consumption), we will use the **Independent Samples t-test**.

The basic procedure for the t-test is as follows: 1. **Establish Null and Alternative Hypotheses**: - **Null Hypothesis (\(H\_0\))**: There is no difference in the true mean values of the two control strategies for that indicator. For example, \(*{DQN} =* {QueueBased}\). - **Alternative Hypothesis (\(H\_1\))**: There is a difference in the true mean values of the two control strategies for that indicator. For example, \(*{DQN}* {QueueBased}\).

1. **Set Significance Level**:
   * We will adopt the conventional significance level of \(= 0.05\). This means that the maximum acceptable probability of wrongly rejecting the null hypothesis (a Type I error) is 5%.
2. **Calculate p-value**:
   * The **p-value** is calculated using the t-test. The p-value represents the probability of observing the current sample data, or more extreme data, given that the null hypothesis is true.

### 2. Interpretation of Results and Conclusion

Based on the calculated p-value, we will make the following interpretations: - **If p < 0.05**: We will **reject the null hypothesis**. This means that the observed performance difference is statistically significant, and we can be 95% confident that there is a real performance difference between the two strategies. - **If p ≥ 0.05**: We will **fail to reject the null hypothesis**. This means that the observed performance difference is not statistically significant and is likely due to random factors. We do not have sufficient evidence to claim a real difference between the two strategies.

By conducting t-tests on all key performance indicators and core experimental group pairings (e.g., DRL vs. baselines), we can provide strong statistical support for the experimental results analysis in Chapter 4, leading to more reliable and convincing research conclusions.