# 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 warehouses utilizing Robotic Mobile Fulfillment Systems (RMFS), increasing order volumes and higher robot deployment densities have made traffic congestion a core bottleneck constraining overall operational efficiency. A large number of robots executing tasks within a limited network of aisles, particularly at intersections, are highly prone to conflicts and waiting due to improper right-of-way coordination. This triggers a series of cascading negative effects, including delays in order fulfillment, energy waste from idle robot waiting, and a decline in overall system throughput.

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

To answer this question, it is essential to first construct an experimental platform capable of accurately simulating these 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.

## 3.2 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 imperative 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 provides a comprehensive overview of this experimental platform, sequentially covering five core components: - **3.2.1 Warehouse Simulation Environment Design**: Details the physical layout and network structure of the simulated world, including core entity components such as robots, pods, and workstations. - **3.2.2 Traffic Control System Architecture**: Delves into 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 Definition of Performance Evaluation Metrics**: Clearly defines the Key Performance Indicators (KPIs) used to measure the effectiveness of different control strategies, which serve as the basis for the experimental results analysis in subsequent chapters. - **3.2.5 Robot Physical Model and Energy Consumption**: Details the robot’s physical model and the method for calculating its energy consumption in the simulation, providing a physical foundation for energy efficiency evaluation.

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

### 3.2.1 Warehouse Simulation Environment Design

To effectively evaluate traffic control strategies, this study first constructed a high-fidelity warehouse simulation environment. This environment not only defines the physical layout but also includes various dynamic entities and their interaction rules, collectively forming 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 is densely packed with **Pod Locations**. The aisles in this area are designed as strict **One-way Aisles**, with the flow directions of horizontal and vertical aisles alternating. This design significantly simplifies the complexity of traffic management at a physical level, aiming to reduce potential conflicts from robots traveling 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, while the other side has **Replenishment Stations**, the entry points for goods into the system.
* **Charging Stations**: Scattered within the storage area, converted from some pod locations, 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 interactions between 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 (energy recovery during braking), providing a solid basis for energy efficiency evaluation. Additionally, robots have priority-based autonomous obstacle avoidance logic, allowing them to resolve local conflicts to some extent on their own.

**【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**: The mobile carrier for storing stock-keeping units (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. After a robot delivers a pod to a station, the system simulates the delay of a worker picking or replenishing items. To handle high traffic, stations are also designed with a dynamic path adjustment mechanism, activating a longer backup path to alleviate congestion when too many robots are present.

**【圖表建議：圖 3.2.3 - 倉儲標示圖】**

#### 3. Orders and Task Flow

The simulation is driven by orders. An **Order** represents a customer request and contains multiple SKUs to be picked. The system breaks down an order into one or more **Jobs**. The core of a job is “transporting a specified pod to a specified 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 the job queue. 4. An idle robot takes a job from the queue and begins its work cycle of picking up, delivering, and returning the pod. 5. When all SKUs for an order have been successfully delivered to the workstation, the order is marked as complete.

#### 4. Intersection Design and Classification

In addition to the macroscopic layout, this study provides a clear classification and definition of the microscopic traffic nodes within the warehouse—the intersections. This is crucial for the subsequent controller design.

* **Standard Intersection**: This is the basic unit that constitutes the main body of the warehouse traffic network, formed by the convergence of two perpendicular one-way aisles. All intersections not otherwise specified fall into this category.
* **Critical Intersection**: Based on their strategic importance in the warehouse layout, some intersections are marked as “critical.” These are traffic nodes on the access or egress paths directly connected to **Workstations (Picking or Replenishment Stations)**. They are the mandatory routes for entering and exiting workstations and constitute the primary traffic **bottlenecks** of the entire warehouse system. The efficiency of managing these intersections directly affects workstation throughput and robot queue lengths, and may trigger **spillback** phenomena that propagate into the storage area. Therefore, in the design of the reinforcement learning reward function (see Section 3.4.5), these intersections will be given a higher weight to guide the agent to prioritize learning their effective management.

**【圖表建議：圖 3.2.4 - 倉儲交叉路口分類與關鍵路口標示圖】** 為直觀展示不同路口的地理分佈，建議此處插入一張與圖 3.2.1 風格一致的倉儲佈局圖。在圖中，應使用不同的符號或顏色清晰標示出標準十字路口與所有關鍵路口的位置，特別是與揀貨站和補貨站的鄰接關係。

### 3.2.2 Traffic Control System Architecture

To enable flexible integration and fair comparison of different traffic control algorithms, this study designed a modular software architecture based on the **Strategy Pattern** and **Factory Pattern**. The core of this architecture is the separation of the “decision algorithm” from the “system execution framework,” ensuring that all strategies, from simple rule-based logic to complex deep reinforcement learning models, operate and are evaluated on the same foundation. The system consists of three main 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 current detailed 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 to record various performance metrics.

#### 2. TrafficControllerFactory (Controller Factory)

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

#### 3. IntersectionManager (Intersection Manager)

The IntersectionManager is the central coordinator and execution engine of the entire traffic control system. Its operational flow forms a complete closed-loop control system: 1. **Holds Controller Instance**: During the simulation initialization phase, the IntersectionManager obtains the required controller instance for the current experiment from the controller factory. 2. **Drives Decision Loop**: At each time unit (tick) of the simulation, the manager iterates through all intersections in the warehouse. 3. **Obtains Decision**: For each intersection, it calls the get\_direction() method of the TrafficController instance to get the traffic command for that intersection. 4. **Executes 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 will then call their train() method, providing the state transition that just occurred (State-Action-Reward-NextState) to the model, 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 this research, all simulation, training, and evaluation experiments were conducted on a well-defined hardware platform and in a standardized software environment. This section details the relevant configurations.

#### Hardware Configuration

The computational tasks of this study were 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 computational resources and accelerate the learning process. - **CPU**: 42 vCPU - **GPU**: 1 x NVIDIA GeForce RTX 4090 - **Memory (RAM)**: 83 GB

##### Development and Analysis Environment (User’s Laptop)

Code development, preliminary 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 software environment was chosen 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 Computational 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): Provided 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 warehouse’s road network graph structure. - scikit-learn (version 1.5.2): In this study, mainly used for preprocessing steps such as data normalization. - **Visualization Libraries**: - matplotlib (version 3.7.2): Used to generate static 2D plots like line charts and bar charts. - seaborn (version 0.13.2): Built on top of matplotlib, providing more aesthetically pleasing and statistically-oriented visualizations like heatmaps. - **System Tools**: - psutil (version 5.9.8): Used to monitor system resource usage.

### 3.2.4 Definition of Performance Evaluation Metrics

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, we first define the following mathematical symbols: - : The set of all robots in the warehouse. - : The set of all orders completed during the simulation period. - : The set of all events of robots passing through an intersection. - : 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 a core optimization target 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).

#### 2. Throughput Metrics

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

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

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

**Average Intersection Waiting Time** This metric directly reflects the coordination efficiency of the traffic control strategy. It calculates the average waiting time for each 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 units of .

#### 3. Stability Metrics

**Total Stop-and-Go Count** This metric reflects the smoothness of the traffic flow. Frequent stops and starts not only consume additional 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. Definition of Unit System

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

* **Time**: The basic time unit in the simulation is the . According to the system settings, one 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 per second (m/s).
* **Acceleration**: The unit is meters per second squared (m/s²).
* **Energy**: Energy is calculated 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 (J).

#### 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.

**【表格建議：表 3.2.1 - 機器人模型物理參數】**

| Parameter | Symbol | Value | Unit | Physical Meaning |
| --- | --- | --- | --- | --- |
| Robot Mass |  | 1 | kg | Mass of the robot itself |
| Load Mass |  | 0 | kg | Additional load when the robot carries goods |
| Gravity |  | 10 | m/s² | Approximate value of Earth’s gravitational acceleration |
| Friction Coefficient |  | 0.3 | - | Rolling friction between the ground and wheels |
| Inertia Coefficient |  | 0.4 | - | Inertial resistance experienced by the robot when accelerating |
| Startup Cost |  | 0.5 | EU | Additional energy cost to overcome static friction when starting from rest |
| Regenerative Braking Efficiency |  | 0.3 | - | Proportion of kinetic energy that can be recovered during braking (30%) |
| Maximum Speed |  | 1.5 | m/s | The robot’s maximum speed 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 regenerative braking mechanisms to more realistically reflect the energy characteristics of electric robots. Total energy consumption is calculated cumulatively based on the robot’s different motion states.

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

where is one time step ( seconds).

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

where is the average velocity during that time step.

**C. Startup Cost** When a robot starts moving () from rest (), 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 all the above, 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 methods proposed in this study (detailed in Section 3.4), they must be compared against a set of representative and easily understandable baseline controllers. Baseline controllers provide a performance reference point, allowing us to quantify the actual improvements brought by more complex algorithms. An ideal baseline should reflect existing 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 of the intersection. These two represent the basic forms of static and dynamic control strategies, respectively, and can comprehensively measure the adaptability and superiority of reinforcement learning models in different traffic scenarios.

This section will 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 a fixed time cycle for switching right-of-way.
2. **Queue-Based Controller**: A dynamic reactive controller that makes decisions based on the length of waiting 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 is derived from traditional urban traffic light systems, completely ignoring 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 obvious—it cannot adapt to fluctuations in traffic demand, often causing unnecessary waiting during busy periods or wasting green time during sparse traffic.

#### Design Principle and Decision Logic

The operation of this controller is determined entirely by three parameters: horizontal green time (), vertical green time (), and the complete 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 the 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 , it grants the right-of-way to the horizontal direction; otherwise, it grants it to the vertical direction. The decision rule can be expressed as:

In our warehouse environment, since pods are mainly arranged along horizontal aisles, the frequency and volume of robot movement in the horizontal direction are much higher than in the vertical direction. To accommodate this characteristic, we assign a longer green time to the horizontal direction (e.g., ticks) and a relatively shorter green time to the vertical direction (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: 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 adapt more flexibly to changes in traffic flow, prioritizing the allocation of passage resources to the direction with more urgent demand.

#### Design Principle and Decision Logic

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

##### 1. Task Priority Weight System

Not all robot tasks in warehouse operations have the same level of 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 pod area has a relatively lower task urgency. To reflect this difference, we define 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 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 (, bias\_factor) to weight the horizontal priority sum, giving it an additional competitive advantage. Thus, 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 switching 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 elapsed since the last direction change. If not, the current direction is maintained. 2. **Priority Comparison**: If the minimum green time is met, the controller calculates the weighted horizontal priority and the vertical priority . 3. **Special Case Handling**: - If either direction has no waiting robots, the right-of-way is immediately given to the other direction that has robots. - If both directions have no robots, the current state is maintained. 4. **Final Decision**: In the general case, 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 respond reasonably and efficiently 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 empirical data from preliminary experiments, aiming for reasonable and stable performance in general scenarios.

#### 1. Time-Based Controller

The logic of this controller is driven entirely by a fixed time cycle, with parameters set as follows:

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

#### 2. Queue-Based Controller

This controller makes decisions based on real-time traffic conditions, and its parameters involve decision sensitivity and 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 for the horizontal direction to compensate for its naturally higher traffic flow, 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 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. DRL methods offer 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 manually designed complex rules.

This section details the two DRL controller architectures designed for this study:

1. **Deep Q-Network (DQN)**: As a mature and widely used DRL algorithm, DQN serves as a strong **baseline** for comparison, used to measure the relative performance of the novel method proposed in this study.
2. **Neuroevolution Reinforcement Learning (NERL)**: This is the **core contribution** of this study. 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 explain the model architecture and operational principles of DQN and NERL, respectively, and then detail the common state space, action space, and reward function designs, which are the core elements of a DRL problem.

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

The Deep Q-Network (DQN) is the **baseline** 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 by a neural network defined by parameters , with the goal of predicting 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**: The transition samples generated from the controller’s interaction with the environment are stored in a fixed-size memory buffer . During training, the algorithm randomly samples a minibatch of experiences from for learning, rather than using consecutive 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 (rather than every step) copied from the policy network’s parameters (). This delayed update mechanism decouples the dependency between the target Q-value and the current Q-value, effectively suppressing oscillations and divergence that may occur during bootstrapping.

#### 2. Learning Process

DQN training is performed by minimizing a loss function over a randomly sampled batch of transition samples. 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 further 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 power of neural networks. It aims to overcome challenges faced by traditional gradient-based DRL methods (like DQN) when dealing with sparse rewards and complex parameter spaces, such as unstable convergence or getting trapped in local optima.

Unlike DQN, which seeks to approximate a value function, neuroevolution’s goal is to directly optimize in the **parameter space** of the 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 searches directly for the optimal policy parameters by applying evolutionary operations to a **population** of many individuals.

#### 1. Evolutionary Process

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 into an independent instance of the simulation environment to perform a full evaluation episode. During this episode, the policy makes all decisions independently. After the episode concludes, a **fitness score** is calculated for the individual 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 reduce training time.
2. **Selection**: Based on the calculated fitness scores of all individuals, the algorithm selects high-performing individuals from the current population to serve as **parents** for generating 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 evolution, 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 parent individuals and are selected, 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 steadily increases, 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 uses 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 elite retention ratio), the specific values of which for this experiment are detailed in the experimental configuration in **Section 3.5.3**.

**【圖表建議：圖 3.4.1 - NERL 演化循環示意圖】**

建議在此處繪製一張循環圖，說明 NERL 的核心運作流程。圖中應包含： 1. **初始族群** : 展示多個神經網路個體 。 2. **並行評估**: 每個 在獨立環境中運行，並計算其適應度 。 3. **選擇**: 根據適應度分數，通過錦標賽選擇出父代。 4. **繁殖生成** : 展示精英保留、交叉和變異操作，生成新一代族群。 5. 箭頭應清晰地指向下一個階段，形成一個從 到 的完整演化閉環。

### 3.4.3 State Space Design

The definition of the state space is a cornerstone of successful reinforcement learning. It must provide the agent with sufficient and meaningful information to make effective decisions, while also 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 decision time is defined as a feature vector that comprehensively reflects its local traffic conditions and considers the potential impact of neighboring 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 the intersection on the horizontal aisle. \* : **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 corresponding four features for the vertical aisle.

**2. Neighbor State - 8 dimensions**: Describes key information from the four neighboring intersections directly connected to the current one, helping the agent understand the broader traffic situation. \* **Each of the top, bottom, left, and right neighbors** includes 2 dimensions of information: \* : **Top Neighbor** - Vertical queue length, horizontal queue length. \* : **Bottom 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 final 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 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 primarily responsible for managing the right-of-way at the intersection.

* **Action 0 (KEEP\_CURRENT\_PHASE)**: **Maintain 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 the cost of switching phases 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 alleviate 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 alleviate traffic pressure in the horizontal direction.

#### 2. Dynamic Speed Control

To more proactively manage traffic flow and prevent 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 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 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. It translates the desired system objectives into an observable and quantifiable scalar feedback signal. To explore the effects of learning at different time scales, this study designed two distinct reward schemes: “Step Reward” and “Global Reward.”

#### 1. Step Reward

The step reward scheme is designed 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 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 passed 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 discourage excessively frequent and ineffective switching and to encourage 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 make the agent prioritize learning to manage traffic near bottleneck areas like 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 scheme provides a sparse and delayed feedback signal, designed to guide 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 only calculates a single reward value 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 designed a global reward function based on an efficiency ratio. This function uses the system’s “output” as the numerator and the system’s “cost” as the denominator. Its formula is defined as:

where the symbols are defined as follows:

* : Total number of completed orders in the evaluation episode.
* : Order completion reward weight.
* : Total energy consumption of the system during the episode (in EU), as detailed in Section 3.2.5.
* : Energy scaling factor, used to adjust the energy cost to a comparable scale with the time cost.
* : Total duration of the evaluation episode (in ticks).
* : 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 volume of completed orders while simultaneously considering energy and time efficiency, thereby learning a more balanced and sustainable operational strategy.

## 3.5 Experiment Design and Evaluation Method

To objectively and quantitatively answer the core question of this research—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 process, performance evaluation framework, and statistical analysis methods for the results.

A rigorous experimental design is the cornerstone for ensuring the reliability of the research conclusions. To this end, we will establish a comparative experimental matrix covering various 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 Experiment Design and Group Definition**: Defines all controller types (experimental groups) included in the experiment and describes the hardware and software environment configurations used for training. - **3.5.2 Model Training Process**: Details the specific training steps for the DRL models (DQN and NERL) to ensure the reproducibility of the experiments. - **3.5.3 DRL Model Hyperparameter Settings**: Details the hyperparameter settings used for the DRL models. - **3.5.4 Evaluation Method and Comparison Framework**: Explains the standardized procedure for final performance evaluation after model training is complete, and the unified framework for comparing all experimental groups.

### 3.5.1 Experiment Design and Group Definition

To systematically evaluate the performance of different traffic control strategies, this study designed 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 schemes and hyperparameter configurations, against two heuristic baseline controllers.

#### Experimental Group Definition

All experimental groups are run in the standardized warehouse simulation environment described in **Section 3.2.1**. The only variable is 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 Definitions and Descriptions**

| Group | Controller Type | Reward Scheme | NERL Variant | NERL Eval. Ticks | Category | Description |
| --- | --- | --- | --- | --- | --- | --- |
| L | TimeBased | - | - | - | Baseline | Static controller with a fixed time cycle |
| K | QueueBased | - | - | - | Baseline | Dynamic reactive controller based on real-time queue length |
| J | DQN | step | - | - | DRL | DQN trained with step rewards (see Sec 3.5.3 for params) |
| I | DQN | global | - | - | DRL | DQN trained with global rewards (see Sec 3.5.3 for params) |
| A | NERL | step | **A (Exploratory)** | 3,000 | DRL | High mutation rate, short evaluation duration (see Sec 3.5.3) |
| C | NERL | global | **A (Exploratory)** | 3,000 | DRL | High mutation rate, short evaluation duration (see Sec 3.5.3) |
| B | NERL | step | **B (Exploitative)** | 3,000 | DRL | Low mutation rate, short evaluation duration (see Sec 3.5.3) |
| D | NERL | global | **B (Exploitative)** | 3,000 | DRL | Low mutation rate, short evaluation duration (see Sec 3.5.3) |
| E | NERL | step | **A (Exploratory)** | 8,000 | DRL | High mutation rate, long evaluation duration (see Sec 3.5.3) |
| G | NERL | global | **A (Exploratory)** | 8,000 | DRL | High mutation rate, long evaluation duration (see Sec 3.5.3) |
| F | NERL | step | **B (Exploitative)** | 8,000 | DRL | Low mutation rate, long evaluation duration (see Sec 3.5.3) |
| H | NERL | global | **B (Exploitative)** | 8,000 | DRL | Low mutation rate, long evaluation duration (see Sec 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 designed 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 allows offspring individuals a greater potential to jump out of the neighborhood of existing solutions and discover entirely new, potentially better strategies, but it may also risk slower convergence.
* **Variant B (Exploitative)**: This configuration focuses on fine-tuning the better solutions that have already been discovered. 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 the stable convergence of the policy but may also increase the risk of getting stuck in a local optimum.

Furthermore, to study the effect of the sufficiency of individual policy evaluation on learning outcomes, each NERL variant will be trained and evaluated under two 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 Process

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

#### 1. DQN Training Process (for Groups I, J)

DQN training is an online, continuous learning process. A single complete DQN training experiment follows these steps:

1. **Initialization**:
   1. Create a DQNController instance according to the hyperparameters defined in Section 3.5.3 and the reward scheme (step or global) specified by the experimental group.
   2. Create an instance of the Warehouse simulation environment.
2. **Training Loop**:
   1. Start a simulation lasting 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. Execute action a\_t. The environment transitions to the next state s\_{t+1}, and the immediate reward r\_t is calculated by a UnifiedRewardSystem (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 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 policy network weights are saved as the final model.

#### 2. NERL Training Process (for Groups A-H)

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

1. **Initialization**:
   1. Create a NEController instance according to Section 3.5.3 and the **specific experimental group** (A-H) definition. This step determines the following key hyperparameters:
      * **Reward Scheme**: 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 process lasting G = 30 generations.
   2. In each generation g:
      1. **Parallel Evaluation**: For the 20 individuals in the population, 20 independent and 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 its 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 scheme (step or global) specified for its group.
      4. **Evolutionary Operations**: Once the fitness scores for all individuals are calculated, the controller performs a full evolutionary operation (selection, crossover, mutation) based on the group’s mutation variant (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 contemporary best model.
   2. After all 30 generations are complete, the model with the historically highest fitness score among 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 J, I).

| Hyperparameter | Code Variable | Value | Description |
| --- | --- | --- | --- |
| Learning Rate | learning\_rate | 5e-4 | The learning rate for the Adam optimizer. |
| Gamma (Discount Factor) | gamma | 0.99 | The discount factor for future rewards. A value closer to 1 indicates a greater emphasis on long-term returns. |
| Initial Epsilon | epsilon | 1.0 | The initial 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 factor 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 Freq | 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 A-H). The mutation rate and mutation strength vary according to the **Exploratory (A)** and **Exploitative (B)** variants defined in **Section 3.5.1**.

| 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 highest-fitness 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 two parent individuals undergoing genetic crossover. |
| Mutation Rate | mutation\_rate | **0.2** | **0.1** | The base probability of an individual’s genes (network weights) mutating. |
| Mutation Strength | mutation\_strength | **0.15** | **0.15** | The standard deviation for Gaussian mutation, controlling the magnitude of mutation. |
| Evaluation Ticks | eval\_ticks | 3000 / 8000 | 3000 / 8000 | The duration (in ticks) for evaluating each individual. |

These hyperparameters collectively define the learning behaviors of the two DRL methods and form 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 process described in Section 3.5.2, a standardized evaluation procedure is designed to ensure all controllers are compared on a fair and unbiased basis. This procedure aims to simulate a fixed, repeatable “test day” scenario and to quantify the performance of each control strategy using a predefined set of Key Performance Indicators (KPIs).

#### 1. Standardized Evaluation Process

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

1. **Model Loading**: For the DRL experimental groups, load their corresponding final trained models and set them to pure **Inference Mode**. In this mode, DQN’s ε-greedy exploration will be turned off (ε=0), and NERL will consistently use its historically best-performing individual for decision-making.
2. **Environment Reset**: Initialize a simulation environment that is identical to the one used for training but uses a set of **fixed 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**: Run a complete simulation of a fixed duration (e.g., T\_{eval} = 50,000 ticks) in the standardized environment.
4. **Data Logging**: During the simulation, a 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 run will use a different, but shared across groups, 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, the data for each experimental group will be aggregated and compared. 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 efficiency that is of primary concern to this study. - **Total Energy Consumption**: Evaluates the system’s energy usage efficiency. Lower is better. - **Completed Orders Count**: Measures the total output of the system within a fixed time. Higher is better.

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

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 to visually demonstrate the dynamic behavior differences between the various strategies during the simulation.

# Chapter 4: Experimental Results and Analysis

## 4.1 Chapter Overview

Following the research problem defined in Chapter 3—namely, how to design an effective and adaptive traffic control strategy that balances system throughput and energy efficiency—this chapter aims to conduct an empirical analysis and comparison of the proposed control schemes through a series of detailed experiments. We will first examine the training processes of the deep reinforcement learning models (DQN and NERL) to confirm their convergence and learning effectiveness. Subsequently, we will systematically compare the comprehensive performance of all controllers (including baseline and DRL models) in a standardized evaluation scenario, covering multiple aspects such as efficiency, throughput, and stability. We will then delve deeper into the specific behavioral patterns of different controllers in key traffic situations to explain the reasons behind their performance differences. Finally, we will use statistical testing methods to verify the significance of the main findings, providing solid data support for the core contributions of this research.

## 4.2 Training Process Analysis

To gain a deep understanding of the dynamic behavior of deep reinforcement learning models during training, this section will not only present the final performance metrics but also conduct a detailed quantitative analysis of their learning trajectories. A key analytical tool is **Evolution Trend Analysis**, which helps us objectively assess whether the model is continuously improving in the desired direction.

### Calculation and Interpretation of Trend Line Slope

In the following sections, we will analyze the evolution process of several Key Performance Indicators (KPIs), such as fitness, order completion rate, and energy consumption per order. To quantify the trend of these indicators as generations evolve, we use **Ordinary Least Squares Linear Regression** to fit a trend line.

For a given KPI time-series data , where is the generation number and is the corresponding KPI value for that generation, linear regression aims to find a straight line that minimizes the sum of the squared residuals between the observed values and the predicted values.

We are most interested in the **slope** of this fitted line. This slope, calculated in analysis/paper\_analyzer.py, mathematically represents the **average change per generation**.

* **Positive Slope ()**: Indicates that the KPI shows a **growing trend** over the course of evolution.
* **Negative Slope ()**: Indicates that the KPI shows a **declining trend** over the course of evolution.
* **Slope near 0**: Indicates that the KPI is relatively stable throughout the evolution process, with no significant upward or downward trend.

By calculating the slope, we can transform a visual intuition into a quantifiable metric, thereby more objectively evaluating whether the model’s learning is a **Desirable Trend** or an **Undesirable Trend**. For example, for “Fitness,” we expect to see a positive slope; whereas for “Energy per Order,” we expect to see a negative slope. This method will be extensively used in Section 4.2.1 to analyze the evolutionary process of NERL.

## 4.2.1 Analysis of the NERL Evolutionary Process

Before validating the final performance of the NERL controllers, it is necessary to examine their evolutionary dynamics during the training process. This analysis helps to understand whether the model is learning effectively, whether the population is converging, and the specific impact of different hyperparameter configurations on the learning process. This section will use Group A (NERL-Step, High Exploration, 3000 Ticks) as a baseline case to conduct a quantitative trend analysis of several Key Performance Indicators (KPIs) during its evolution.

### 1. Benchmark Case Analysis: Multi-dimensional Evolution Trend of the High-Exploration Variant (A)

Figure 4.2.1 shows the performance of the elite individuals from the Group A (NERL-Step, High Exploration, 3000 Ticks) experiment across three core KPIs over 30 generations of evolution: Best Fitness, Completion Rate, and Energy per Order.

| (a) Best Fitness | (b) Completion Rate | (c) Energy per Order |
| --- | --- | --- |
| Image A to be inserted here | Image B to be inserted here | Image C to be inserted here |

*Figure 4.2.1: Evolution trends of the elite individuals from Group A (NERL-Step, High Exploration, 3000 Ticks) on (a) Best Fitness, (b) Completion Rate, and (c) Energy per Order.*

Based on the trend analysis, the evolution slope and trend assessment for each indicator are as follows:

* **Best Fitness**: Slope = +1594.24 (Desirable Trend)
* **Completion Rate**: Slope = +0.0008 (Desirable Trend)
* **Energy per Order**: Slope = -2.69 (Desirable Trend)
* **Signal Switch Count**: Slope = +3.58 (Undesirable Trend)

From the data and charts, the following points can be summarized:

1. **Effective Learning and Optimization**: As shown in Figure 4.2.1(a), the **Best Fitness** of the elite individuals shows a significant upward trend (slope +1594), proving that the evolutionary algorithm is effectively guiding the model towards the goal of maximizing the reward function (defined in Section 3.4.5). This macroscopic score improvement is supported by specific sub-goals. Although the **Completion Rate** in Figure 4.2.1(b) fluctuates, its trend line is still positive (slope +0.0008), while the **Energy per Order** in Figure 4.2.1(c) shows a significant and continuous downward trend (slope -2.69). These two points together indicate that the model not only learned to complete more orders but also learned to do so in a more energy-efficient manner, meaning the reward function design achieved its intended effect.
2. **Strategy Trade-off and Evolution**: An interesting phenomenon is the evolutionary trend of the **Signal Switch Count**. Although the step reward design includes a penalty term () for “phase switching,” the data shows that the slope of this indicator is positive (+3.58), meaning the model tends to switch traffic signals more frequently. This is not a failure of learning but a manifestation of the agent’s autonomous decision-making. This phenomenon reveals that the model discovered during the learning process that **the local penalty from moderately increasing phase switches can be exchanged for a significant improvement in traffic fluidity, thereby obtaining a much larger global benefit (higher completion rate and lower waiting times) that far outweighs the penalty.** This complex trade-off behavior, sacrificing a local metric for macroscopic optimality, is difficult for traditional rule-based controllers to achieve.
3. **The Cost and Value of Exploration**: The drastic fluctuations in the completion rate in Figure 4.2.1(b) show that the high-exploration Group A was trying different strategies in each generation. The attempts of some generations might be unsuccessful (like the sharp drop in generations 9-10), but this breadth of exploration is the foundation for ultimately finding an efficient and energy-saving strategy (like the stable high point after generation 25).

In summary, the analysis of the benchmark case indicates that NERL can not only successfully learn and optimize multiple core KPIs but can also exhibit complex strategic trade-off capabilities.

### 2. Comparative Analysis: The Impact of Different Evolutionary Configurations

To further understand the role of various hyperparameters, we will compare the benchmark case with other representative experimental groups.

#### a. Reward Scheme: Step vs. Global

When comparing different reward schemes, an important premise is that one should not directly compare the absolute values or slopes of their “Fitness,” because the calculation methods and numerical scales of Step Reward and Global Reward are completely different.

Therefore, the focus of this section is to analyze how these two reward schemes, as driving forces for training, respectively affect the common Key Performance Indicators (KPIs)—that is, the final actual operational performance. For this, we will compare Group A (NERL-Step, High Exploration, 3000 Ticks) and Group C (NERL-Global, High Exploration, 3000 Ticks).

**【圖表建議：圖 4.2.2 - 不同獎勵模式下 (Step vs. Global) 關鍵績效指標的演化趨勢對比圖】**

For a quantitative comparison, Table 4.2.1 summarizes the evolution trend slopes for the two experimental groups on core output and efficiency indicators.

| Experiment Group | Reward Scheme | Key Performance Indicator (KPI) | Evolution Trend Slope | Trend Assessment |
| --- | --- | --- | --- | --- |
| **A (High-Explore, 3000 Ticks)** | **Step** | **Completion Rate** | **+0.000825** | **Desirable** |
| C (High-Explore, 3000 Ticks) | Global | Completion Rate | -0.000698 | Undesirable |
| **A (High-Explore, 3000 Ticks)** | **Step** | **Energy per Order** | **-2.695533** | **Desirable** |
| C (High-Explore, 3000 Ticks) | Global | Energy per Order | -0.720350 | Desirable |

*Table 4.2.1: Comparison of evolution trend slopes for Completion Rate and Energy per Order under Step and Global reward schemes.*

From the data analysis, the following two conclusions can be drawn:

1. **Guidance of Step Reward on Throughput Improvement**: The completion rate is a direct measure of the system’s core output. As shown in Table 4.2.1, Group A, which used step rewards, shows a positive development trend in its completion rate (slope +0.000825), indicating that its dense, immediate reward signals successfully guided the agent to learn policies that effectively increase the system’s overall throughput. In contrast, Group C, using global rewards, shows a negative development trend in its completion rate (slope -0.000698). This phenomenon suggests that, under the current training settings, relying solely on sparse, delayed global rewards makes it difficult for the agent to establish effective credit assignment from micro-level decisions to macro-level outcomes, leading to a noisy learning signal.
2. **Driving Force of Step Reward on Efficiency Optimization**: The data shows that although the energy consumption per order is decreasing in both groups, the downward slope of the step-reward group (A) (-2.69) is much steeper than that of the global-reward group (C) (-0.72). This means that the step reward not only guides the model to complete orders but also drives it to do so in a more energy-efficient manner.

In summary, the experimental evidence supports the effectiveness of the step reward scheme in guiding the NERL controller to learn complex warehouse tasks. It not only ensures that the model evolves towards increasing the system’s total output but also more efficiently discovers optimization potential in operational details.

#### b. The Impact of Exploration Strategy: High (A) vs. Low (B)

In neuroevolution, the diversity of the population and the exploratory strength of individuals are key determinants of whether the algorithm can escape local optima. This section aims to investigate the impact of exploration strategy hyperparameter settings in the NERL controller on the evolutionary outcomes. We will compare two experimental groups that use the same step reward but have different exploration strengths: Group A (NERL-Step, High Exploration, 3000 Ticks) and Group B (NERL-Step, Low Exploration, 3000 Ticks).

**【圖表建議：圖 4.2.3 - 不同探索強度下 (高 vs. 低) 關鍵績效指標的演化趨勢對比圖】**

Table 4.2.2 summarizes the evolution trends of key indicators for high and low exploration strength configurations.

| Experiment Group | Exploration Strength | Key Performance Indicator (KPI) | Evolution Trend Slope | Trend Assessment |
| --- | --- | --- | --- | --- |
| **A (Step, 3000 Ticks)** | **High** | **Completion Rate** | **+0.000825** | **Desirable** |
| B (Step, 3000 Ticks) | Low | Completion Rate | -0.000334 | Undesirable |
| **A (Step, 3000 Ticks)** | **High** | **Fitness** | **+1594.24** | **Desirable** |
| B (Step, 3000 Ticks) | Low | Fitness | +693.45 | Desirable |
| A (Step, 3000 Ticks) | High | Energy per Order | -2.695533 | Desirable |
| B (Step, 3000 Ticks) | Low | Energy per Order | -1.753562 | Desirable |

*Table 4.2.2: Comparison of evolution trend slopes for core KPIs between High Exploration (A) and Low Exploration (B).*

The core conclusion from the data analysis is: **Insufficient exploration can lead to premature convergence to a local optimum, or even to a policy that fails to improve system throughput.**

1. **Impact on System Throughput**: As shown in Table 4.2.2, the completion rate of the high-exploration Group A shows an upward trend (slope +0.000825); however, the completion rate of the low-exploration Group B is decreasing (slope -0.000334). This phenomenon indicates that in a complex traffic control problem, if the agent’s exploration is insufficient and it dares not try actions that might temporarily reduce efficiency, it can get stuck in an inefficient policy. The evolution of Group B may have converged to a suboptimal policy of “traffic stagnation to avoid any potential collision risk,” ultimately undermining the overall system goal.
2. **Breadth and Potential of Learning**: The growth slope of Fitness also corroborates this view. The fitness growth rate of Group A (slope +1594) is more than double that of Group B (slope +693). This does not mean Group A’s learning efficiency is higher, but rather that its “learning horizon” is broader. Higher exploration allows the population to search in a wider policy space. Although the process may be accompanied by greater fluctuations, this breadth is a necessary prerequisite for discovering efficient policies. Group B, while also learning, has a search range that is too narrow, limiting its optimization potential.

In summary, this comparative analysis highlights the importance of setting sufficient exploration strength in the NERL framework. For tasks that require solving complex trade-off problems, giving the evolutionary process enough freedom to explore is a necessary path to an efficient and robust solution.

#### c. The Impact of Evaluation Duration: 3000 Ticks vs. 8000 Ticks

The evaluation duration (evaluation\_ticks) determines the time each agent interacts with the environment to demonstrate the quality of its policy during evolution. This section analyzes the actual impact of evaluation duration on learning effectiveness by comparing Group A (NERL-Step, High Exploration, 3000 Ticks) and Group E (NERL-Step, High Exploration, 8000 Ticks).

**【圖表建議：圖 4.2.4 - 不同評估時長下 (3000 vs. 8000 ticks) 關鍵績效指標的演化趨勢對比圖】**

Table 4.2.3 summarizes the evolution trends of key indicators for the two evaluation duration configurations.

| Experiment Group | Evaluation Duration | Key Performance Indicator (KPI) | Evolution Trend Slope | Trend Assessment |
| --- | --- | --- | --- | --- |
| **A (NERL-Step, High-Explore)** | **3000** | **Completion Rate** | **+0.000825** | **Desirable** |
| E (NERL-Step, High-Explore) | 8000 | Completion Rate | -0.001454 | Undesirable |
| **A (NERL-Step, High-Explore)** | **3000** | **Energy per Order** | **-2.695533** | **Desirable** |
| E (NERL-Step, High-Explore) | 8000 | Energy per Order | +0.064461 | Undesirable |
| **A (NERL-Step, High-Explore)** | **3000** | **Signal Switch Count** | **+3.583537** | **Undesirable** |
| E (NERL-Step, High-Explore) | 8000 | Signal Switch Count | -3.322136 | Desirable |

*Table 4.2.3: Comparison of evolution trend slopes for core KPIs between 3000 Ticks (A) and 8000 Ticks (E) evaluation durations.*

The data analysis results indicate: **Simply extending the evaluation time may lead to a deterioration of the learning process; longer is not necessarily better.**

1. **Credit Assignment Delay and Reward Signal Dilution**: This is the core problem. In the step reward design (see Section 3.4.5), the agent receives immediate feedback at each time step. When the evaluation duration is extended from 3000 to 8000 ticks, the number of time steps in a full episode increases significantly. This causes the reward from any specific, beneficial micro-action to be diluted over the long total duration. It becomes difficult for the agent to link the final outcome to a key decision made thousands of steps earlier, which is a classic “credit assignment problem” in reinforcement learning. The success of Group A lies in its relatively short evaluation window, which makes the causal chain between actions and feedback clearer and the learning signal stronger.
2. **Failure of Short-Sighted Penalties and Policy Drift**: A longer time window may also render certain penalty terms ineffective. For example, the penalty for phase switching, . Within a 3000-tick window, the cost of frequent switching is significant, and the agent must make a trade-off between the “penalty of switching” and the “gain in fluidity” (as shown by Group A’s slope of +3.58). However, on the 8000-tick scale, the agent might discover that by strenuously avoiding switches (as shown by Group E’s slope of -3.32) to accumulate minor rewards, its overall fitness might still be higher, even if this leads to long-term traffic paralysis later on. The model may have learned a “short-sighted conservative policy,” sacrificing the long-term, ultimate system goal to escape immediate minor penalties.

In conclusion, this comparative analysis reveals an important principle for setting the evaluation duration: the evaluation window must match the design of the reward function and the time scale of the task itself. An appropriately sized evaluation window is essential to ensure the effectiveness of the reward signal and the stability of the learning process.

### 3. Comprehensive Performance Comparison at the Training Stage

The preceding sections analyzed the impact of different hyperparameter configurations on the model’s **evolutionary process (slope)**. However, the trend of the evolutionary process does not fully equate to the superiority of the final performance. This section shifts the perspective from “process” to “preliminary results.” By conducting a horizontal comparison of the performance of the **elite models** from each experimental group in their respective training evaluation scenarios at the end of training, we aim to examine the preliminary performance achieved by different strategy combinations and to provide a reference for the more rigorous, standardized final validation in the next section.

**【Figure Suggestion: Figure 4.2.5 - Final Completion Rate Comparison of All NERL Elite Models in Training Evaluation】** **【Figure Suggestion: Figure 4.2.6 - Final Energy per Order Comparison of All NERL Elite Models in Training Evaluation】** **【Figure Suggestion: Figure 4.2.7 - Final Average Intersection Congestion Comparison of All NERL Elite Models in Training Evaluation】** **【Figure Suggestion: Figure 4.2.8 - Final Signal Switch Count Comparison of All NERL Elite Models in Training Evaluation】** **【Figure Suggestion: Figure 4.2.9 - Final Total Stop-and-Go Events Comparison of All NERL Elite Models in Training Evaluation】**

Through a comprehensive analysis of the above charts, several key observations can be made at the training stage. These observations depict the characteristics of different strategies, but their ultimate effectiveness remains to be validated:

1. **Potential of Global Reward and Long Evaluation: The “Big Picture” Strategy on the Training Ground** A significant trend in the training stage evaluation is that the combination of “Global Reward + Long Evaluation,” such as in Group D (NERL-Global, Low-Explore, 3000 Ticks) and Group H (NERL-Global, Low-Explore, 8000 Ticks), shows strong potential in the core metrics of **Completion Rate** and **Energy per Order**. This reveals a possible mechanism: although the global reward signal is sparse, when given a sufficiently long evaluation time, the evolutionary algorithm has the space to explore grander, more complex long-term strategies. The agent is no longer bound by short-term step rewards but can “discover” strategies in the long evaluation that sacrifice short-term benefits for long-term returns. However, a major concern for the final validation in the next section is whether this complex strategy, highly adapted to the training environment, is too “delicate” and thus “fragile.”
2. **The “Short-Sighted” Risk of Step Rewards: Policy Drift with Long Evaluation** In contrast, the step-reward models that performed well in the 3000-tick short evaluation, such as Group E (NERL-Step, High-Explore, 8000 Ticks) and Group F (NERL-Step, Low-Explore, 8000 Ticks), showed mediocre performance in completion rate at the end of training when the evaluation was extended to 8000 ticks. This seems to confirm the hypothesis from subsection c, that there is a mismatch between step rewards and an overly long evaluation window. Within 8000 ticks, the agent might fall into a state of “Reward Hacking”: excessively focusing on executing actions that maximize short-term, immediate step rewards, while these actions may harm the ultimate goal of completing orders over a longer time scale.
3. **The Performance Trade-off: The Aggressive Nature of High-Performing Models** From the charts of signal switch counts and stop-and-go events, it can be seen that the experimental groups that achieved higher completion rates in the training evaluation, such as Group G (NERL-Global, High-Explore, 8000 Ticks) and Group H (NERL-Global, Low-Explore, 8000 Ticks), also had the most severe internal disturbances in their traffic systems. This reveals that these models learned an “aggressive” management style, intervening frequently to improve flow efficiency. The effectiveness of this strategy is highly dependent on accurate prediction of the environment, and its stability will be tested when facing the longer, more unpredictable scenarios of true validation.

**Summary and Outlook**: Combining the analysis of the evolutionary process and the final state of training, a preliminary conclusion can be drawn: there is no single “optimal” hyperparameter configuration, but rather **strategy combinations** suitable for different objectives. - The Global Reward + Long Evaluation combination learned the most promising, macroscopic, but perhaps most complex, strategies during training. - The Step Reward + Short Evaluation combination produced the most stable, but possibly less potent, strategies.

Can the characteristics and potential observed in the training environment be translated into real strength in a standardized, long-cycle validation? Which model’s strategy is more **generalizable** and **robust**? The answers to these questions will be revealed in the final performance validation in the next section. This provides a clear, problem-oriented transition for us from “process analysis” to “final validation.”

## 4.3 Final Performance Validation and Comparison

In the previous section, we analyzed the impact of different hyperparameter configurations on the NERL controller’s performance during the **training process** and observed that certain combinations (like global reward with long evaluation) showed better potential in the training environment. However, the core question this section aims to address is whether performance during training can be directly translated into the model’s actual performance in longer, more general scenarios. This directly relates to the model’s **Generalization** and **Robustness**.

To this end, this section places all controllers—including baselines, DQN, and all NERL models—in a unified, standardized validation scenario lasting 50,000 ticks for performance evaluation. The goal is to objectively reveal which control strategy demonstrates ultimate comprehensive superiority in a persistent task that is closer to real-world operations, after stripping away the biases of the specific training environment.

### 1. Standardized Final Validation and Comparative Analysis

To conduct a comprehensive horizontal comparison of the final performance of all twelve controllers, their validation data across six Key Performance Indicators (KPIs) are compiled in Table 4.3.1. These indicators provide a quantitative assessment of each model’s final performance from three dimensions: system throughput (completion rate), operational efficiency (energy per order), and system stability (signal switch count, stop-go events).

**Table 4.3.1: Comprehensive Comparison of Final Validation Performance for All Experimental Groups** | Experiment | Completion Rate | Energy per Order | Total Energy | Signal Switches | Stop-Go Events | Completed Orders | |:—|:—:|—:|:—:|:—:|:—:|—:| | K\_EVAL\_queue\_based | 91.40% | 54 | 35,764 | 12,702 | 17,035 | 659 | | F\_EVAL\_nerl\_step\_b8000ticks | 91.27% | 33 | 21,621 | 12,758 | 16,396 | 659 | | I\_EVAL\_dqn\_step\_55000 | 90.78% | 50 | 33,764 | 12,755 | 17,421 | 679 | | D\_EVAL\_nerl\_global\_b3000ticks | 90.68% | 44 | 28,718 | 12,203 | 16,489 | 652 | | B\_EVAL\_nerl\_step\_b3000ticks | 90.57% | 51 | 33,013 | 12,278 | 16,885 | 653 | | C\_EVALnerl\_global\_a3000ticks | 89.94% | 51 | 32,745 | 12,033 | 16,793 | 644 | | J\_EVAL\_dqn\_global\_55000 | 89.90% | 60 | 38,996 | 12,575 | 17,266 | 650 | | G\_EVALnerl\_global\_a8000ticks\_ | 89.57% | 49 | 32,044 | 12,201 | 16,563 | 653 | | E\_EVAL\_nerl\_step\_a8000ticks | 89.42% | 41 | 26,047 | 12,251 | 17,071 | 642 | | H\_EVAL\_nerl\_global\_b8000ticks | 89.20% | 46 | 29,182 | 11,856 | 16,234 | 636 | | L\_EVAL\_time\_based | 88.56% | 45 | 28,027 | 9,900 | 17,783 | 627 | | A\_EVAL\_nerl\_step\_a3000ticks | 84.51% | 57 | 25,013 | 8,201 | 11,743 | 442 |

**【圖表建議：圖 4.3.1 - 所有控制器在「訂單完成率」上的最終表現對比】** **【圖表建議：圖 4.3.2 - 所有控制器在「單均能耗」上的最終表現對比】**

The data in Table 4.3.1 reveals the disparity between training performance and final validation results. This is not just a performance ranking but a profound test of the generalization capabilities of different learning strategies.

1. **Disparity between Training Performance and Generalization Ability** First, a key observation is that Group H (NERL-Global, Low-Explore, 8000 Ticks), which performed best in the training stage evaluation in Section 4.2, only achieved a completion rate of 89.20% in the final validation, significantly lower than other models. This performance degradation confirms a core concept in machine learning research: **Overfitting** to the training environment. The complex macroscopic strategy learned by Group H, while achieving excellent performance in the 8000-tick training evaluation, was likely too dependent on the specific conditions of training, leading to a “fragile” policy. When the evaluation period was extended to 50,000 ticks, and the dynamics and complexity of the environment increased, the strategy could not maintain its effectiveness, exposing its lack of generalization ability.
2. **The Importance of Policy Robustness: The Performance of QueueBased and NERL-F** In contrast to the performance decline of Group H, the excellent performance of Group K (QueueBased) and Group F (NERL-Step, Low-Explore, 8000 Ticks) stands out. The success of these two models can be attributed to the inherent **Robustness** of their policies.
   * **Performance of QueueBased**: As a simple reactive strategy, Group K (QueueBased) achieved the highest completion rate (91.40%) with its intuitive “let the longer queue pass” rule. This shows that a well-designed, simple rule-based baseline model can be highly competitive in specific metrics due to its stability and effectiveness. However, its high energy consumption per order (54) shows that this strategy maximizes throughput at the cost of energy efficiency, exhibiting a typical “single-objective optimal” characteristic.
   * **The Comprehensive Optimal Solution of NERL-F**: The success of Group F (NERL-Step, Low-Explore, 8000 Ticks) is more enlightening. It nearly matched Group K’s completion rate at 91.27%, but its energy per order was as low as 33, making it the most energy-efficient among all high-throughput models and demonstrating the best **energy-to-performance ratio**. This indicates that Group F’s training configuration—“Step Reward + Low Exploration + Long Evaluation”—fostered a **highly generalizable and balanced policy**. The step reward provided a stable and clear learning signal; low exploration prevented the model from learning overly risky and fragile policies; and the long evaluation period gave the model a long-term perspective during learning. This combination led it to learn a more “reliable” rather than the most “sophisticated” strategy, ultimately achieving the best comprehensive performance in the multi-objective trade-off between throughput and efficiency.
3. **The Baseline Value of DQN Models** The performance of Group I (DQN-Step) was also robust, ranking among the top with a 90.78% completion rate, proving the effectiveness of the standard DQN method in solving this type of problem. It serves as a very important measurement baseline, showing that a fully trained standard reinforcement learning model can outperform many more complex NERL variants, providing a solid reference point for the comparisons in this study.

### 2. Conclusion: From “Training Optimal” to “Comprehensive Optimal”

Synthesizing all validation results, the core conclusion of this research becomes clear: in the complex problem of warehouse traffic control, **the optimal performance of a model during training does not equate to its final comprehensive performance.**

Although models based on global rewards, like Group H (NERL-Global, Low-Explore, 8000 Ticks), showed potential for learning complex long-term strategies during training, such strategies have weaker generalization ability and are prone to overfitting the training environment. Conversely, a well-designed heuristic controller like QueueBased (Group K) can achieve the highest value in a core output metric but at the expense of energy efficiency.

The comprehensively optimal model in this study is Group F (NERL-Step, Low-Explore, 8000 Ticks). It demonstrates that a learning method combining **clear immediate feedback (step reward)** and **conservative exploration (low exploration)** can learn a policy with **strong generalization ability and the best overall effectiveness** in long-horizon tasks. It not only achieves a top-tier completion rate but also, with its superior energy efficiency, sets a new performance benchmark for the next generation of intelligent warehouse traffic controllers. This finding provides significant experimental evidence for how to deploy DRL systems in the real world that are not just “smart” but also “reliable.”

## 4.4 Chapter Summary

This chapter aimed to conduct an in-depth quantitative evaluation and comparison of the traffic control performance of rule-based baseline controllers, standard Deep Q-Network (DQN) controllers, and various Neuroevolution Reinforcement Learning (NERL) variants in a complex warehouse environment through a series of systematic experiments. The analysis in this chapter followed a rigorous path from “process” to “results,” and from “training potential” to “generalization evidence.”

First, in Section 4.2, the study analyzed the training and evolutionary processes of the deep learning models. The analysis showed that different reward schemes, exploration strategies, and evaluation durations have a significant impact on the models’ learning dynamics. In particular, it was observed that the combination of “global reward” and “long evaluation,” as in Group H (NERL-Global, Low-Explore, 8000 Ticks), demonstrated higher potential for learning complex, long-term strategies in the training environment.

However, in the standardized final performance validation in Section 4.3, the study found a significant disparity between training-stage performance and final results. This turning point highlights a core issue in machine learning research—**Generalization**. The model that performed best during training, Group H, learned a policy that was overly adapted to the specific training environment (i.e., **Overfitting**), causing it to perform poorly in longer, more general testing scenarios and exposing the “fragility” of its strategy.

The final comparative results clearly indicate that: 1. A well-designed heuristic controller, such as Group K (QueueBased), can achieve the optimal value in a single metric like “completion rate” due to the **robustness** of its policy, but this advantage comes with higher energy consumption. 2. The comprehensively best-performing model in this study is Group F (NERL-Step, Low-Explore, 8000 Ticks). The training configuration used for this model enabled it to learn a **balanced policy** that is not overly aggressive and combines stability with efficiency. It not only achieved a top-tier completion rate but also demonstrated the best overall performance in the multi-objective trade-off with its significantly superior energy efficiency compared to other high-throughput models.

In conclusion, the experiments in this chapter do not merely select an optimal model but, more importantly, reveal that in complex problems like intelligent warehouse control, **a training framework that promotes robust and generalizable learning is key to translating the theoretical potential of deep reinforcement learning into real-world application value**. The success of Group F proves the importance of pursuing a solution that is not just “sophisticated” but also “reliable.” These findings provide solid data support for the final conclusions of this thesis and point the way for future related research.

# Chapter 5: Conclusion and Future Work

## 5.1 Research Summary

This research aimed to solve the core problem of reduced operational efficiency due to traffic congestion in modern automated Robotic Mobile Fulfillment Systems (RMFS). To this end, this thesis focused on designing an effective and adaptive intelligent intersection traffic control strategy, with the goal of maximizing the system’s overall order fulfillment capability while minimizing robot idle waiting and energy consumption.

To achieve this goal, the study first constructed a high-fidelity warehouse simulation environment and designed two traditional rule-based controllers (Time-Based and Queue-Based) as performance baselines. Subsequently, Deep Reinforcement Learning (DRL) methods were introduced, implementing controllers based on the standard Deep Q-Network (DQN) and a NERL controller that combines neuroevolution with reinforcement learning. For the NERL model, this study further designed multiple sets of comparative experiments covering different reward schemes, exploration strengths, and evaluation durations to systematically investigate the impact of various hyperparameters on model behavior and performance.

The analysis of the experimental results followed a rigorous path from “training process” to “final validation.” The analysis of the training process revealed that NERL models using “global reward” and “long evaluation” (such as Group H) showed great potential for learning complex, macroscopic strategies. However, a critical turning point occurred in the longer, more challenging standardized final validation: the aforementioned models that performed excellently in training proved to be fragile due to their policies overfitting the training environment, exhibiting insufficient generalization ability and ultimately not achieving the best performance.

The final conclusion of this study indicates that the controller with the best comprehensive performance is Group F (NERL-Step, Low-Explore, 8000 Ticks). The “step reward, low exploration, long evaluation” combination used by this model enabled it to learn a balanced policy that is not overly aggressive and possesses both a high order completion rate and excellent energy efficiency. Its success demonstrates that in a complex, stochastic, and dynamic environment, the **robustness** and **generalization ability** of a policy are more important traits than achieving extreme metrics under specific training conditions. This finding not only provides a concrete solution to the core problem of this research but also offers profound practical insights for deploying DRL systems in the real world in the future.

## 5.2 Research Contributions

This research has made several specific contributions to the field of traffic control in automated warehousing at the theoretical, methodological, and practical application levels. At the theoretical level, one of the core contributions of this thesis is the systematic revelation of the potential gap between the training performance and final generalization ability of deep reinforcement learning models through rigorous comparative experiments. The study clearly points out that policies that perform optimally during the training phase (such as complex strategies driven by global rewards) may fail in longer, more general validation environments due to overfitting. This finding provides an important empirical case for the DRL field, emphasizing that when evaluating models, standardized generalization ability tests must be the ultimate gold standard, rather than relying solely on metrics from the training process.

Methodologically, this study successfully applied Neuroevolution Reinforcement Learning (NERL) to the complex RMFS traffic control problem and, through experimental validation, identified a training configuration capable of producing a balanced policy with both high throughput and high energy efficiency—namely, the combination of step rewards, low exploration, and long evaluation duration. This achievement not only provides a successful case for the application of hybrid DRL methods like NERL in the field of logistics automation but also offers concrete experimental evidence and inspiration for future researchers on how to balance the “breadth of exploration” and “stability of learning” when designing DRL systems.

Based on this methodology, this study also achieved significant results in practical applications. The Group F model (NERL-Step, Low-Explore, 8000 Ticks) we proposed demonstrated the best comprehensive performance among all tested controllers, especially showing a significantly superior energy-to-performance ratio compared to traditional baselines and standard DQN models. This provides a high-potential advanced technology solution for warehouse operators on how to effectively control rising energy costs and carbon footprints while pursuing increased throughput in automated systems.

Finally, all achievements of this research are built upon the scalable, high-fidelity RMFS simulation platform we constructed from scratch. This platform not only provided a solid foundation for all experiments in this thesis but its modular design also gives it good extensibility, allowing future researchers to use it for testing more advanced algorithms, different warehouse layouts, or more complex task scheduling strategies, thereby constituting an infrastructural contribution to the advancement of the entire field.

## 5.3 Research Limitations

Although this study has drawn a series of important conclusions through rigorous experimental design, it is still necessary to acknowledge its existing limitations. These limitations not only define the scope of applicability of the study’s conclusions but also provide clear directions for future research.

First, one of the core limitations of this study is the inevitable gap between the simulation environment and physical reality (Sim-to-Real Gap). Although all experiments in this study were conducted on a high-fidelity simulation platform, this platform ultimately cannot fully capture all the randomness and complexity of a real physical environment. Factors not modeled, such as hardware wear and tear, sensor delays, or battery aging, could all affect the controller’s performance during actual deployment. Therefore, the performance of the optimal model identified in this study in a real physical environment still requires on-site validation and calibration.

Second, the conclusions of this study are based on a fixed warehouse layout and network structure, which constitutes another limitation on the generalizability of its findings. Although the chosen layout is representative, whether the conclusions of this study regarding the optimal controller and its training configuration can be directly generalized to other warehouses with different topological structures remains to be confirmed by further empirical research, as different network structures may give rise to vastly different traffic bottlenecks and dynamic characteristics.

Furthermore, to focus on the core problem of intersection traffic control, this study made necessary simplifications to the task and traffic models. For example, more complex order structures, collaborative operations between robots, or treating picking station congestion as an endogenous learning objective were not considered. These simplifications help to isolate variables for analysis but also mean that the controller in this study has not yet addressed higher-level systemic challenges.

Finally, this study is also limited in the breadth of algorithms explored. Although it compared several representative controllers, the field of deep reinforcement learning is rapidly evolving. This study was unable to include other promising advanced algorithms like Proximal Policy Optimization (PPO) or Soft Actor-Critic (SAC) in the comparison, which might exhibit different characteristics in terms of sample efficiency or policy stability.

## 5.4 Future Work

Based on the findings of this study and the aforementioned limitations, future research can be pursued in several directions to further advance the technology of intelligent warehouse traffic control.

First, to address the Sim-to-Real Gap, a key future direction is to investigate how to transfer and deploy high-performance models trained in a simulation environment to real physical robot systems in a low-cost and efficient manner. This could involve domain adaptation techniques, fine-tuning models on a small amount of real data, or researching robust reinforcement learning algorithms that are less sensitive to changes in physical parameters, to enhance the feasibility of real-world deployment.

Second, addressing the limitation of conducting the study in a single warehouse layout, future research should systematically evaluate the generalization ability of this study’s optimal controller across diverse topological structures. Furthermore, one could investigate how to expose the DRL agent to a variety of layouts during training to learn a more universal traffic control meta-policy that can autonomously adapt to different network structures, which is crucial for improving the algorithm’s versatility.

Moreover, as this study focused on microscopic traffic control at intersections, a valuable extension would be to integrate this study’s traffic controller as a low-level execution module into a higher-level intelligent decision-making system. Such a system could be responsible for more macroscopic task allocation and dynamic path planning, and even consider systemic factors like picking station congestion. This would allow for the exploration of hierarchical or multi-agent collaborative decision-making architectures to achieve deeper, system-wide optimization.

In addition, to broaden the scope of algorithmic exploration, future research should introduce modern DRL algorithms that have proven effective in other domains, such as Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC). A comprehensive performance comparison of these algorithms against the best-performing NERL model from this study would help to more clearly identify the strengths and weaknesses of different algorithms in solving the RMFS traffic problem.

Finally, to better align with the goals of green logistics, future models could integrate more refined energy dynamics models. For example, incorporating factors like battery charge/discharge cycle life and energy consumption curves at different speeds into the reward function design could drive the agent to learn a more sustainable operational strategy that is not only energy-efficient but also helps to extend hardware lifespan.

# Chapter 6: References

[1] eMarketer. (2024, February). *Worldwide ecommerce will account for over 20% of retail sales in 2024*. https://www.emarketer.com/content/ecommerce-account-more-than-20–of-worldwide-retail-sales-despite-slowdown

[2] DHL. (2024). *Sustainability trends in logistics for 2024*. DHL. https://www.dhl.com/discover/en-global/logistics-advice/sustainability-and-green-logistics/sustainability-trends-in-logistics

[3] PwC. (2023). *The EU CBAM: Implications for supply chains*. PwC. https://www.pwc.com/gx/en/services/tax/esg-tax/cbam-supply-chain-imperatives.html

[4] Mousavi, S. S., Fathollahi-Fard, A. M., & Hajiaghaei-Keshteli, M. (2024). Deep reinforcement learning driven cost minimization for batch-picking in robotic mobile fulfillment systems. *Expert Systems with Applications*, *254*, 124568.

[5] Song, L., Li, K., Yang, K., Liu, P., & Yang, K. (2025). Towards energy-efficient Robotic Mobile Fulfillment System: Hybrid traffic control for automated guided vehicles. *Applied Soft Computing*, *173*, 113452.

[6] Wurman, P., D’Andrea, R., & Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, *29*(1), 9-20.

[7] Guizzo, E. (2012, March 19). *Amazon acquires Kiva Systems for $775 million*. IEEE Spectrum. Benavides-Robles, M.T., et al., Robotic Mobile Fulfillment System: A Systematic Review.

[8] Merschformann, M., et al. (2019). Decision rules for robotic mobile fulfillment systems. *Operations Research Perspectives*, *6*, 100128.

[9] Stern, R. (2019). Multi-agent path finding–an overview. In *Artificial Intelligence: 5th RAAI Summer School, Tutorial Lectures* (pp. 96-115).

[10] Li, X., Hua, G., Huang, A., Sheu, J. B., Cheng, T. C. E., & Huang, F. (2020). Storage assignment policy with awareness of energy consumption in the Kiva mobile fulfilment system. *Transportation Research Part E: Logistics and Transportation Review*, *144*, 102158.

[11] Luo, L., Zhao, N., Zhu, Y., & Sun, Y. (2023). A\* guiding DQN algorithm for automated guided vehicle pathfinding problem of robotic mobile fulfillment systems. *Computers & Industrial Engineering*, *178*, 109112.

[12] Zhou, Y., Guo, K., Yu, C., & Zhang, Z. (2024). Optimization of multi-echelon spare parts inventory systems using multi-agent deep reinforcement learning. *Applied Mathematics and Computation*, *469*, 128532.

[13] Dunne, M. C., & Potts, R. B. (1964). Algorithm for traffic control. *Operations Research*, *12*(6), 870-881.

[14] Teja, V. A., Viswanath, D. V. K., & Krishna, K. M. (2009, February 21-25). A mixed autonomy coordination methodology for multi-robotic traffic control. In *2008 IEEE International Conference on Robotics and Biomimetics* (pp. 1391-1396).

[15] Ross, D., Sandys, R., & Schlaefli, J. (1971). A computer control scheme for critical-intersection control in an urban network. *Transportation Science*, *5*(2), 141-160.

[16] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[17] Zheng, G., et al. (2019, November 3-7). Learning phase competition for traffic signal control. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (pp. 1653-1662).

[18] Cao, K., Wang, L., Zhang, S., Duan, L., Jiang, G., Sfarra, S., Zhang, H., & Jung, H. (2024). Optimization control of adaptive traffic signal with deep reinforcement learning. *Electronics*, *13*(1), 198.

[19] Zhu, Y., Cai, M., Schwarz, C., Li, J., & Xiao, S. (2022). Intelligent traffic light via policy-based deep reinforcement learning. *International Journal of Intelligent Transportation Systems Research*, *20*(3), 734-744.

[20] Angela, J., Sujana, J., & Namasivaayam, L. (2024, August 22-23). Pressure, queue, and average speed - based multi-agent DQN for optimizing traffic signal control. *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)* (pp. 1-6).

[21] Chen, Y. F., et al. (2017, May 29-June 3). Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

[22] Ma, Z., Luo, Y., & Ma, H. (2021, May 30-June 5). Distributed heuristic multi-agent path finding with communication. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*.

[23] Heidrich-Meisner, V., & Igel, C. (2009). Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms*, *64*(4), 152-168.

[24] AbuZekry, A. (2019). Comparative study of NeuroEvolution algorithms in reinforcement learning for self-driving cars. *European Journal of Engineering Science and Technology*.

[25]Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864.

[26]Yuan, R., Dou, J., Li, J., Wang, W., & Jiang, Y. (2023). Multi-robot task allocation in e-commerce RMFS based on deep reinforcement learning.. Mathematical biosciences and engineering : MBE, 20 2, 1903-1918 .

[27]Zhou, X., Shi, X., Chu, W., Jiang, J., Zhang, L., & Deng, F. (2024). Learning to Solve Multi-AGV Scheduling Problem with Pod Repositioning Optimization in RMFS. 2024 IEEE International Conference on Industrial Technology (ICIT), 1-8.

[28] Zou, B., Xu, X., & De Koster, R. (2018). Evaluating battery charging and swapping strategies in a robotic mobile fulfillment system. *European Journal of Operational Research*, *267*(2), 733-753.