



Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Deep Reinforcement Learning for Stochastic Locomotive Assignment

A Rolling Horizon Framework with Proactive Maintenance and Operational Interpretability

BRUNO CARCHIA

Deep Reinforcement Learning for Stochastic Locomotive Assignment

A Rolling Horizon Framework with Proactive Maintenance and Operational Interpretability

BRUNO CARCHIA

Date: July 24, 2026

Supervisors: Kiarash Kazari, Zohreh Ranjbar, Paolo Garza
Examiner: György Dán

Host company: RISE AB

d |

Abstract

Heavy freight rail operations typically necessitate the allocation of one or multiple locomotives to satisfy specific tractive effort requirements for successful transit. The Locomotive Assignment Problem (Locomotive Assignment Problem (LAP)) seeks to optimally route and assign these limited motive power assets to a scheduled timetable, minimizing systemic lateness while strictly adhering to rigorous operational constraints. As a highly constrained combinatorial optimization problem, traditional exact solvers and heuristic methodologies suffer from severe computational bottlenecks. Incorporating real-world stochasticity renders these classical approaches intractable, severely limiting their scalability in industrial applications.

To overcome these computational limitations, this thesis proposes a highly scalable Deep Reinforcement Learning (Deep Reinforcement Learning (DRL)) framework. By shifting the computational burden to an offline training phase, the agent can execute near-instantaneous routing decisions during live operations. The proposed architecture leverages Shared-Weight Multi-Layer Perceptrons (Multi-Layer Perceptrons (MLPs)) to achieve permutation invariance across the fleet, operates within an asynchronous event-driven simulation, and incorporates a continuous Weibull hazard model to simulate probabilistic mechanical degradation and incentivize proactive maintenance.

The experimental methodology follows a phased escalation of environmental complexity across three distinct architectural configurations. In its final configuration, the DRL agent successfully mitigates systemic maintenance bottlenecks, achieving a 96.8% win rate against the baseline and reducing mean episodic lateness by approximately 60%. Finally, to resolve the inherent opacity of deep neural networks and foster operational trustworthiness, the framework integrates Explainable AI (XAI) principles. By translating the agent's continuous, sequential decisions into industry-standard visual interfaces, the DRL framework functions not as a black box, but as a transparent, high-fidelity decision-support tool for human operational planners.

Keywords

Deep Reinforcement Learning, Explainable AI, Engine Rescheduling, PPO, Rolling Horizon Planning, Locomotive Assignment Problem,MLP Condition-Based Maintenance , Rolling stock planning , Continuous Hazard Model , Shared-Weight Neural Networks

Sammanfattning

Unga godståg behöver ofta ett eller flera lok för att orka dra lasten och komma fram i tid. Problemet med att fördela lok (LAP) handlar om att ge rätt lok till rätt tåg enligt en tidtabell. Målet är att minska förseningar och samtidigt följa alla hårda regler. Eftersom detta är ett mycket svårt pussel med många regler, tar det för lång tid för vanliga datorprogram att lösa det. Om man dessutom lägger till oväntade händelser som sker i verkligheten, slutar dessa gamla metoder att fungera bra i stor skala. För att lösa dessa datorproblem föreslår det här arbetet en smart AI-metod som kallas Deep Reinforcement Learning (DRL). Genom att låta datorn (agenten) träna länge i förväg, kan den fatta blixtsnabba beslut när den väl används på riktigt. Vårt system använder speciella AI-nätverk för att kunna hantera alla lok på ett enhetligt sätt. Det körs i en flexibel simulering och använder en matematisk modell för att förutse när loken riskerar att gå sönder, så att man kan boka in service i god tid. Vi testade systemet i tre steg, där vi gjorde miljön svårare och svårare för varje steg. Från att bara hantera en enkel planering, fick agenten till slut hantera en mycket rörig miljö där den var tvungen att göra tre saker samtidigt: planera tåg, hantera oväntade fel och boka in förebyggande underhåll. I det allra sista testet lyckades DRL-agenten lösa problemen med underhåll. Den var bättre än standardmetoden i 96,8% av fallen och minskade förseningarna med ungefär 60%. Eftersom det ofta är svårt att förstå hur AI tänker, har vi också lagt till funktioner för att förklara besluten (Explainable AI). Genom att visa agentens val i tydliga och välkända diagram, fungerar DRL-systemet inte som en obegriplig ”svart låda”. Istället blir det ett tydligt och pålitligt verktyg som hjälper mänskliga planerare att göra ett bra jobb.

Sommario

Le operazioni di trasporto ferroviario merci pesante richiedono tipicamente l'allocazione di una o più locomotive per soddisfare gli specifici requisiti di sforzo di trazione necessari al successo del transito. Il Problema di Assegnazione delle Locomotive (LAP, *Locomotive Assignment Problem*) mira a instradare e assegnare in modo ottimale queste limitate risorse di forza motrice secondo un orario prestabilito, minimizzando il ritardo sistemico e nel contempo rispettando rigorosamente severi vincoli operativi. Essendo un problema di ottimizzazione combinatoria fortemente vincolato, i tradizionali risolutori esatti e le metodologie euristiche soffrono di gravi colli di bottiglia computazionali. L'incorporazione della stocasticità del mondo reale rende questi approcci classici intrattabili, limitandone gravemente la scalabilità nelle applicazioni industriali.

Per superare tali limitazioni computazionali, questa tesi propone un framework altamente scalabile basato sul *Deep Reinforcement Learning* (DRL). Trasferendo l'onere computazionale a una fase di addestramento offline, l'agente è in grado di eseguire decisioni di instradamento quasi istantanee durante le operazioni dal vivo. L'architettura proposta sfrutta percettroni multistrato a pesi condivisi (MLPs, *Shared-Weight Multi-Layer Perceptrons*) per ottenere l'invarianza permutazionale attraverso la flotta, opera all'interno di una simulazione asincrona guidata dagli eventi (*event-driven*) e incorpora un modello di rischio continuo di Weibull per simulare il degrado meccanico probabilistico e incentivare la manutenzione proattiva.

La metodologia sperimentale segue un'escalation incrementale della complessità ambientale attraverso tre distinte configurazioni architetturali. Il framework evolve da un ambiente di instradamento di base vincolato verso un ecosistema operativo altamente stocastico, in cui l'agente deve ottimizzare simultaneamente le assegnazioni di dispacciamento, gestire guasti imprevisti degli asset e programmare la manutenzione preventiva.

Nella sua configurazione finale, l'agente DRL mitiga con successo i colli di bottiglia sistemici dovuti alla manutenzione, raggiungendo un tasso di vittoria (*win rate*) del 96,8% rispetto alla baseline e riducendo il ritardo episodico medio di circa il 60%. Infine, per risolvere l'opacità intrinseca delle reti neurali profonde e promuovere la fiducia operativa (*trustworthiness*), il framework integra i principi dell'*Explainable AI* (XAI). Traducendo le decisioni continue e sequenziali dell'agente in interfacce visive standard per il settore, il framework DRL funge non da *black box*, bensì come uno strumento di supporto decisionale trasparente e ad alta fedeltà per i pianificatori umani.

Contents

1 Introduction **1**
1.1 Problem . . . 2
1.2 Purpose and Societal Impact . . . 3
1.3 Research Objectives and Methodology . . . 5
1.4 Delimitations . . . 6
1.5 Ethical Aspects and Sustainability . . . 8
1.6 Structure of the thesis . . . 8

2 Background **11**
2.1 Fundamentals of Reinforcement Learning . . . 11
2.1.1 Markov Decision Process (MDP) . . . 13
2.1.2 Bellman Expectation Equations . . . 14
2.1.3 Bellman Optimality Equations . . . 15
2.1.4 Policy Gradient Methods . . . 16
2.1.5 Advantage Actor-Critic (A2C) . . . 20
2.1.6 Trust Region Policy Optimization (TRPO) . . . 21
2.1.7 Proximal Policy Optimization (PPO) . . . 24
2.2 Architectural Paradigms and Constraint Handling in Policy Optimization . . . 26
2.2.1 The Sensory Neuron as a Transformer: Permutation-Invariant Neural Networks for Reinforcement Learning 26
2.2.2 A Closer Look at Invalid Action Masking in Policy Gradient Algorithms . . . 28
2.2.3 Action Space Shaping in Deep Reinforcement Learning 31
2.2.4 Addressing Function Approximation Error in Actor-Critic Methods . . . 32
2.3 Locomotive Assignment Problem (LAP) . . . 33

2.3.1 An advanced learning environment and a scalable deep reinforcement learning approach for rolling stock circulation on urban rail transit line 36
2.3.2 Deep Reinforcement Learning for Integrated Train Scheduling and Rolling Stock Circulation 38
2.3.3 Solving high-speed rolling stock planning with an improved LSTM-PPO reinforcement learning algorithm 40
2.3.4 Rolling Stock Planning with Integrated Maintenance Constraints: A Rolling Horizon MILP Approach . . . 42
2.3.5 Dynamic wagon assignment and reassignment for pick-up and delivery requests: A deep reinforcement learning framework to address uncertainty in locomotive availability 45
2.3.6 An actor-critic deep reinforcement learning approach for metro train scheduling with rolling stock circulation under stochastic demand 47
2.3.7 Optimization of Locomotive Repair Operations via Preventive Maintenance 49

3 Methods and Implementation 53
3.1 Overview of the Research Methodology 53
3.2 Hardware and development setup 54
3.3 Sequential Decision-Making Formulation 55
3.4 The Simulation Environment 56
3.4.1 Entity Definitions and System Attributes 56
3.4.2 Maintenance Dynamics and Episodic Initialization . . 57
3.4.3 Event-Based Simulation Architecture 57
3.5 Training Details 58
3.5.1 Optimization Strategy 59
3.5.2 The PPO Update Loop 59
3.5.3 Evaluation and Logging Protocol 61
3.6 Baseline Formulation 63
3.7 Observation and Action Space Formulation (Model v1) 64
3.7.1 Observation Space 64
3.7.2 Action Space 64
3.8 Neural Network Architecture (Model v1) 66
3.8.1 Stochastic Continuous Policy and Gaussian Sampling . 68
3.9 Reward Function Engineering (Model v1) 69
3.9.1 Dense and Event-Driven Formulations 69

3.9.2 Sparse and Terminal Formulations 69
3.9.3 Implementation Architecture and Hyperparameter Sensitivity 69
3.10 Model v2 71
3.10.1 The Continuous Hazard Model 71
3.10.2 Weibull Distribution and Reliability Engineering . . . 73
3.11 Model v3 74
3.11.1 Observation Space Augmentation 75
3.11.2 Action Space and Sequential Logic 75
3.11.3 Critic Neural Network Architecture 76
3.11.4 Reward Shaping for Maintenance 77
3.12 Explainability and Decision Support (XAI) 78
3.12.1 Emulating Industry-Standard Topologies 79
3.12.2 Iterative Refinement of the Visualization Framework . 81

4 Results and Analysis **85**
4.1 Evaluation Framework and Empirical Protocols 85
4.2 Model v1 Results 86
4.3 Model v2 Results 87
4.4 Model v3 Results 90
4.5 Model v3: Evaluation under Stochastic Hazard Conditions . . 93

5 Discussion **97**
5.1 Limitations 98

6 Conclusions and Future Work **99**
6.1 Conclusions 99
6.2 Future Work 100

References **101**

List of Figures

2.1 Agent-Environment Interaction in a Reinforcement Learning System. Taken from Jiang et al., 2019 [5]. 13
2.2 The foundational architecture of Policy Gradient Algorithms, illustrating equivalent mathematical forms across different objective functions. 17
2.3 TRPO algorithm pseudocode. Adapted from OpenAI Spinning Up [15]. 23
2.4 PPO algorithm pseudocode. Adapted from OpenAI Spinning Up [15]. 25
2.5 Architecture of the neural network for the permutation-invariant agent. Adapted from Tang et al., 2021 [17]. 27
2.6 Architecture of the proposed Actor-Critic Neural Network. Taken from (Yang et al., 2025 [25]) 37
2.7 Architecture of the proposed Actor-Critic Neural Network. Taken from (Zhao et al., 2026 [26]) 39
2.8 PPO-LSTM Algorithm. Taken from (SU et al., 2026 [27]) . . 42
2.9 Temporal decomposition of the rolling horizon approach, distinguishing between detailed activity scheduling and expected maintenance requirements. Taken from Bröchin et al. (2023) [28]. 43
2.10 Heuristic rules for wagon assignment or re-assignment. Taken from Habibi et al. (2026) [29]. 46
2.11 Test of the actor-critic agent during the training process.Taken from (Ying et al., 2020 [30]) 49

3.1 Architectural schematic of the bifurcated continuous Actor policy. The left pathway deduces macro-level dispatch strategies, while the right pathway establishes strict permutation invariance by independently evaluating each locomotive i through shared weights. 67
3.2 The implemented stochastic failure probability mapping. Driven by a shape parameter of $\beta = 2$, the breakdown probability scales quadratically with the locomotive's cumulative wear ratio, peaking at an artificially capped maximum hazard rate of 98%. 74
3.3 The upgraded Critic Network architecture for Model v3. By exclusively processing macro-state abstractions rather than raw positional micro-states, the network achieves highly stable baseline value estimations. 77
3.4 The legacy (V1) Gantt chart formulation. Utilizing an hourly X-axis and Swedish localization, the chart isolates every entity, leading to fragmented visual tracking. 82
3.5 The legacy (V2) Gantt chart demonstrating severe vertical scaling issues. As the episode progresses, the strict one-row-per-train logic creates visual bloat, rendering long-horizon evaluations intractable for human planners. 83
3.6 The refined DRL evaluation interface. Trains are consolidated into grouped rows detailing actual versus scheduled transit, maintenance is categorically hatched, and the temporal axis is scaled to multi-day increments, drastically enhancing operational interpretability. 84

4.1 Performance evolution of Model v1 across intermediate trials. The charts highlight the period of operational parity (Trials 10–19) followed by a definitive phase shift in policy performance (Trial 20+) resulting from the Shared-Weight architectural upgrade. 88
4.2 Direct performance comparison between the peak configuration of Model v1 (deterministic environment) and the most robust configuration of Model v2 (stochastic environment with 4 random mechanical failures). The chart illustrates the minor but expected performance degradation caused by real-time asset breakdowns. 90

4.3 Comparative progression of the Model Win Rate and Lose Rate across the evaluation of Model v3. By autonomously managing fleet health, the agent nearly eliminates scenarios where the heuristic out-performs the network, converging to a strict operational dominance at Trial 33. 91
4.4 Evolution of the Delta Lateness Average during the training of Model v3. The steady downward trajectory indicates the agent's increasing proficiency in utilizing proactive maintenance to eliminate structural delays, peaking at an unprecedented −51.21 hours in Trial 33. 92
4.5 The Delta Success Rate Average for Model v3. The graph quantifies the agent's absolute superiority in adhering strictly to the scheduled departure timetable, culminating in a +23.3% operational advantage over the baseline heuristic. 92
4.6 Evolution of the Delta Lateness Average during the stochastic evaluation of Model v3 (Trials 35–37). The data demonstrates the agent's capacity to preserve superior delay mitigation even when subjected to dynamic, Weibull-distributed mechanical failures. 94
4.7 Progression of the Model Win Rate and Model Lose Rate metrics for Model v3 evaluated under continuous mechanical hazard constraints. The variance highlights the agent's structural resilience to unpredicted fleet degradation. 95
4.8 The Delta Success Rate Average across the final stochastic trials. The metric isolates the agent's superior capacity to maintain strict timetable punctuality despite randomly injected asset unavailability. 95

List of Tables

2.1 Comparative analysis of operational and physical characteristics between passenger and freight rail systems. 34
2.2 Taxonomy of foundational Locomotive Assignment Problem (LAP) models (Adapted from Piu and Speranza, 2014). 36
2.3 Comparative analysis between the reference study and the proposed DRL approach. 38

3.1 Daily operational schedule and turnaround parameters for the Station A to Station B corridor. Turnaround buffers are expressed in hours. 56
3.2 Episodic Metrics (Aggregated per Rollout) 61
3.3 Proximal Policy Optimization (PPO) Parameters 62
3.4 Comparative Benchmarks (Evaluated at 10-Episode Intervals) 62
3.5 Observation Space Attributes and Dimensions for Model v1 . 65
3.6 Continuous Action Space Output and Downstream Mapping . 65
3.7 Dense and Event-Driven Reward Components 70
3.8 Sparse and Terminal Reward Components 70
3.9 Augmented Global Features for Proactive Maintenance (Model v3) 75
3.10 Architectural Comparison: DRL Visualization vs. Industry-Standard Interfaces 80

4.1 Core Evaluation Metrics Computed over the 500-Episode Validation Phase 86
4.2 Comprehensive performance metrics for the development of Model v1 (Trials 10–23). Missing values (–) indicate metrics that were either not tracked or not relevant during early experimental phases prior to framework stabilization. 89

4.3 Performance metrics for Model v2 under stochastic failure conditions (Trials 24–25). The table highlights the agent's robustness when subjected to unexpected mechanical breakdowns. 89
4.4 Comprehensive performance metrics for the development and optimization of Model v3 (Trials 26–34). The table encapsulates the final performance ceiling of the proactive fleet management architecture. 93
4.5 Comprehensive performance metrics for Model v3 evaluated under the Continuous Hazard Model (Trials 35–37). The metrics quantify the framework's ultimate capability to synthesize proactive maintenance scheduling with reactive crisis management. 96

List of acronyms and abbreviations

A2C	Advantage Actor-Critic
ADP	Approximate Dynamic Programming
DRL	Deep Reinforcement Learning
GAE	Generalized Advantage Estimation
KL	Kullback-Leibler
LAP	Locomotive Assignment Problem
LSTM	Long Short Term Memory
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
MILP	Mixed-Integer Linear Programming
MLP	Multi-Layer Perceptron
PI	Permutation Invariance
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RSC	Rolling Stock Circulation
RSU	Rolling Stock Units
TRPO	Trust Region Policy Optimization

Chapter 1

Introduction

The Locomotive Assignment Problems (LAPs) entails the optimal allocation of motive power, specifically locomotives, to a defined set of scheduled trains operating between designated origin and destination terminals. The fundamental objective is to satisfy stringent tractive effort constraints while simultaneously minimizing operational expenditures. Historically, the LAP has been approached utilizing conventional combinatorial optimization and operations research methodologies [1, 2, 3]. However, these traditional computational frameworks are highly computationally expensive, often requiring prohibitive execution times, and suffer from a distinct lack of generalizability. Existing literature predominantly presents highly customized, case-specific models that fail to capture the stochastic intricacies of real-world rail logistics, such as dynamic maintenance scheduling, unforeseen operational disruptions, and complex routing constraints. A critical limitation of these deterministic approaches is their fundamental lack of scalability. As the fleet size increases, the computational complexity grows non-linearly due to combinatorial explosion, rendering them intractable for large-scale industrial applications.

These traditional limitations are particularly evident in the highly complex logistical ecosystem operated by GreenCargo, a major Swedish freight logistics operator. Their network, primarily focused on heavy timber transport, spans demanding topographies from southern to northern Sweden. Directional load imbalances strictly dictate motive power requirements: northbound trains typically haul empty freight wagons, necessitating only a single locomotive, whereas southbound trains, heavily laden with timber, require a minimum of two locomotives to satisfy tractive demands. Furthermore, the operational reality involves a heterogeneous fleet with varying locomotive specifications,

introducing stringent driver qualification and licensing constraints, alongside critical operational latencies such as the four to six hours of setup and shunting time required to couple a locomotive to a train.

In collaboration with GreenCargo and the Research Institutes of Sweden (RISE), this research explores a novel paradigm for solving the LAP utilizing Deep Reinforcement Learning (DRL). By formulating the assignment task as a sequential decision-making process, we train an autonomous agent to maximize a cumulative reward signal. This reward mechanism indirectly encapsulates operational objectives, allowing the agent to discover optimal, scalable assignment policies without relying on rigid predefined heuristics.

To systematically bridge the gap between traditional literature and GreenCargo's ultimate operational reality, this thesis develops a foundational DRL framework. The research trajectory progresses from a constrained baseline architecture to a sophisticated decision-making agent. To ensure algorithmic tractability and verifiable policy convergence during this initial proof-of-concept, the experimental setup introduces deliberate operational boundaries. Specifically, the simulated environment abstracts the spatial topology to a dual-node network comprising two stations with dedicated maintenance workshops. The fleet is restricted to 10 locomotives operating within a simplified, deterministic daily timetable of 6 trains executed over a five-day simulation horizon. Furthermore, transit durations are abstracted to a fixed 15 hours per journey, and maintenance protocols are triggered strictly by a deterministic distance threshold.

Despite these experimental abstractions, this thesis establishes a robust, scalable DRL methodology capable of navigating complex state spaces. Ultimately, this framework lays the essential algorithmic groundwork to empower GreenCargo to transition from a reactive to a proactive approach to the LAP, achieving significant cost optimizations while paving the way for future integration of their complete, real-world operational constraints.

1.1 Problem

This thesis addresses the locomotive assignment and rescheduling problem through the lens of advanced machine learning paradigms, seeking to systematically investigate the following primary research question:

> *RQ T.1: How can a rolling horizon DRL approach improve adaptability and planning quality in locomotive rescheduling*

under disruptions, compared to a purely sequential allocation strategy?

Furthermore, given the inherent opacity of deep neural network architectures, this project investigates a secondary, highly critical challenge regarding operational interpretability:

RQ T.2: How can explainable AI techniques be integrated into a deep reinforcement learning-based planning system to improve transparency and understanding of rescheduling decisions for human planners?

The depth and validity of the proposed solutions are intrinsically tied to the fidelity of the underlying environment and the granularity of the modeled operational constraints. Scaling the complexity of the domain to closely approximate real-world logistics necessitates a commensurate increase in the representational capacity and algorithmic sophistication of the DRL agent. To facilitate this optimization without risking active operational disruption, the agent is trained utilizing synthetically generated environments that rigorously emulate empirical operational distributions.

The subsequent chapters systematically address the primary research question through a graduated, curriculum-based methodology. The investigation initiates with a highly constrained, low-fidelity abstraction of the assignment problem, fundamentally restricting the state-action space to ensure baseline policy convergence. Progressively, the environmental complexity and operational realism are scaled up. This sequential relaxation of constraints provides a rigorous, multi-faceted analysis of the agent's performance, evaluating the rolling horizon framework across varying degrees of computational hardness and operational stochasticity.

1.2 Purpose and Societal Impact

This thesis aims to contribute to the existing academic literature by investigating the application of Deep Reinforcement Learning (DRL), specifically Proximal Policy Optimization (PPO), within the LAP. By deploying these advanced solutions in practical, empirical scenarios, this research provides critical insights into how autonomous agents can resolve highly constrained industrial operations. Furthermore, it establishes a generalized methodological framework outlining typical approaches that can

be adapted for analogous combinatorial optimization challenges within the logistics sector.

The successful implementation of this research yields tangible benefits across three primary stakeholder domains:

- **GreenCargo:** The formulation provides a mechanism to significantly reduce operational expenditures. Specifically, it minimizes costs associated with catastrophic mechanical failures, unproductive light locomotive movements (deadheading), and systemic transit delays.
- **Operational Planners:** Rather than functioning as an autonomous replacement, the proposed machine learning system acts as a sophisticated decision-support tool. It empowers human dispatchers to dynamically refine schedules and routing plans, thereby yielding superior operational outcomes while reducing cognitive load.
- **The Scientific Community:** This thesis bridges the gap between theoretical reinforcement learning and applied operations research by providing a rigorous empirical framework based on an industrial use case. The specific architectural design of the agent establishes a foundational baseline for future research aiming to tackle real-world complexity in a graduated manner.

Beyond immediate operational and academic metrics, the broader implications of this work are deeply intertwined with sustainability and socio-ethical considerations. From an economic and societal perspective, reducing operational latencies and infrastructural costs directly enhances the economic efficiency of rail freight, which can ultimately lower the end-user cost of transported goods. Environmentally, the minimization of deadhead locomotive transits directly correlates with a reduction in the ecological footprint and energy waste of the rail network.

Moreover, the integration of proactive maintenance scheduling mitigates the risk of sudden mechanical failures and subsequent incidents along the railway, thereby enhancing public safety and infrastructural resilience. Finally, from an ethical standpoint, the deployment of this DRL agent is explicitly designed within a human-in-the-loop paradigm. It is positioned not as a substitute for specialized human labor, but as an augmenting technology intended to optimize resource allocation, minimize systemic waste, and ensure the reliable delivery of logistical commitments while strictly preserving human agency and oversight.

1.3 Research Objectives and Methodology

The primary objective of this project is to synthesize a policy that minimizes corporate operational expenditures and ensures robust freight delivery, while simultaneously maintaining a computationally tractable interface for human operational planners. To systematically approach this overarching goal, the research is delineated into the following highly granulated, technical sub-objectives:

1. **Policy Parameterization:** Architecting a dynamic neural network topology capable of accurately representing the DRL agent's decision-making policy within a complex state space.

2. **Environmental Transition:** Migrating the underlying simulation framework to an event-based architecture to more accurately reflect asynchronous logistical operations.

3. **Metric Optimization:** Maximizing the relative performance of the agent against baseline heuristics. Specifically, this entails maximizing the *Delta Win Rate* (the frequency with which the agent achieves lower episodic lateness than the baseline), maximizing the *Delta Success Rate* (the differential in the volume of trains dispatched strictly on time), and minimizing the *Delta Average Lateness* (the relative reduction in mean episodic delay).

4. **Proactive Asset Management:** Extending the agent's action space to seamlessly integrate proactive maintenance scheduling, balancing locomotive duty cycles with operational demands.

5. **Stochastic Robustness:** Evaluating the resilience of the agent's assignment strategy under the introduction of stochastic operational disruptions, specifically random intra-transit asset failures.

To rigorously quantify the agent's efficacy, all performance metrics in this study are evaluated relatively against a baseline policy. This baseline is defined as a constraint-aware stochastic heuristic; while it allocates exactly two locomotives per train at random, it is constrained to sample exclusively from the pool of currently available assets (excluding those in active transit or undergoing maintenance).

To ensure systematic validation, the research methodology is structured around the progressive development and evaluation of three distinct model

configurations, each introducing an escalating degree of operational complexity:

1. **Model v1 (Nominal Operations):** The agent operates within a relaxed constraint space, possessing the freedom to execute locomotive assignments without the encumbrance of maintenance scheduling or stochastic asset failures.

2. **Model v2 (Reactive Disruption Management):** The environmental complexity is increased by introducing random locomotive failures. The agent must dynamically detect and react to unexpected asset unavailability, requiring robust rescheduling capabilities.

3. **Model v3 (Proactive Lifecycle Management):** The framework transitions from rigid, hard-coded maintenance thresholds to an autonomous policy paradigm. The agent is tasked with endogenous maintenance scheduling, optimizing the trade-off between maximizing asset utilization and adhering to critical service requirements.

These three models are presented sequentially, mirroring the chronological trajectory of the research. The underlying methodology relies on a highly iterative, step-wise refinement process. The initial phase focused on algorithmic exploration, testing diverse strategies to identify the optimal mapping to the defined Markov Decision Process (MDP). Subsequent to strategy selection, rigorous empirical validation phases were conducted to verify the integrity of state transitions and ensure the simulation environment behaved deterministically under known constraints. Finally, comprehensive hyperparameter optimization was executed to maximize the target reward metrics. This graduated approach is critical; it ensures sequential empirical validation, thereby establishing profound statistical reliability and operational trustworthiness in the final model prior to scaling its complexity.

1.4 Delimitations

To maintain a rigorous focus on the algorithmic evaluation of the Deep Reinforcement Learning (DRL) agent, and to ensure computational tractability, several explicit boundaries have been established for this research. The primary delimitations of this thesis are defined as follows:

- **Data Provenance and Preprocessing:** The empirical validation relies entirely on synthetically generated datasets sampled from

baseline statistical distributions. Consequently, typical data engineering pipelines—including data cleaning, feature extraction, and augmentation—are explicitly excluded from the scope of this study. This isolation ensures that the performance of the DRL architecture is evaluated independently of dataset variance.

- **Environmental and Physical Degradation:** The simulated environment abstracts away meteorological conditions. The physical wear and stochastic degradation of locomotive components due to adverse weather impacts are not modeled, assuming a uniform operational environment.

- **Spatial and Infrastructural Abstractions:** The spatial topology of the rail network is restricted to a simplified two-node system. This contrasts with real-world logistics networks, which comprise highly complex, multi-node configurations. Furthermore, each of the two simulated stations is assumed to possess a maintenance workshop with unconstrained, infinite capacity. The realities of restricted maintenance throughput and heterogeneous station capabilities are not considered.

- **Theoretical Generalization:** The primary objective is to evaluate the theoretical applicability of DRL to the LAP. Therefore, the model introduces strict operational abstractions. It must be acknowledged that the resulting policy formulations cannot be directly deployed into unconstrained, real-world operational environments without the future relaxation of these deterministic assumptions.

- **Scope of Explainability (XAI):** While this thesis addresses the interpretability of the DRL agent to foster operational trustworthiness, the scope of Explainable AI is deliberately bounded. The research does not attempt to develop or integrate mathematically rigorous algorithmic explainability methods (e.g., attribution mapping or value-function decomposition). Instead, the scope is restricted to user-centric transparency, focusing heavily on UI/UX design paradigms to ensure that the agent's scheduling decisions are easily interpretable and actionable for human operational planners.

1.5 Ethical Aspects and Sustainability

A main ethical issue is transparency. Since deep reinforcement learning models are difficult to interpret, planners must be able to understand why a locomotive is assigned, delayed, or sent to maintenance. The use of visual explanations and human-readable scheduling interfaces is therefore important for trust and accountability. However, these explanations should be validated with real users before any operational deployment.

Data governance is also relevant. This thesis uses synthetic data, which avoids direct risks related to confidential operational records. Future work based on real GreenCargo data would require secure access control, anonymization where appropriate, and careful handling of commercially sensitive or personal information.

From a safety perspective, punctuality and cost reduction must never be optimized at the expense of maintenance requirements. The model therefore treats maintenance as a central operational constraint. Before practical use, the system would need further validation against railway safety procedures and rare disruption scenarios.

The sustainability contribution lies mainly in improving rail freight efficiency. By reducing deadheading, delays, and unnecessary locomotive movements, the framework may lower energy waste and improve fleet utilization. Proactive maintenance can also extend asset lifetime and reduce severe failures. At the same time, the computational cost of training DRL models should be considered, especially in future large-scale versions of the system.

Overall, the proposed approach can support more sustainable rail freight operations, provided that it remains transparent, safety-oriented, and firmly embedded in a human-in-the-loop planning process.

1.6 Structure of the thesis

The rest of the report is structured as follows:

- **Chapter 2** presents the background of the thesis. It introduces the Locomotive Assignment Problem, reinforcement learning, Proximal Policy Optimization, maintenance-aware scheduling, and related work in rail optimization.
- **Chapter 3** describes the methodology. It explains how the locomotive assignment problem is formulated as a DRL task, including the

simulation environment, state space, action space, reward function, and model architecture.

- **Chapter 4** presents the experimental results. It evaluates the proposed models against the baseline heuristic and analyzes performance in terms of lateness, win rate, success rate, stochastic failures, and proactive maintenance.
- **Chapter 5** discusses the findings. It reflects on the strengths and limitations of the proposed approach, including the use of synthetic data, simplified network topology, and the role of explainability for human planners.
- **Chapter 6** concludes the thesis and outlines future work. It summarizes the main contributions and identifies the steps required to move toward a more realistic and deployable decision-support system for locomotive assignment.

Chapter 2

Background

This section provides a critical analysis of the literature and technical reports that informed the selection of the methodologies and technologies employed in this research. Each reviewed paper contributed specifically to the architectural design or the strategic formulation of the Reinforcement Learning (RL) agent. To ensure clarity, each paper is presented using the following structure:

- **Core Summary:** A brief overview of the research objectives and findings of the paper.
- **Technical Contribution:** An explanation of how the paper's insights directly influenced the development of this thesis.

2.1 Fundamentals of Reinforcement Learning

RL is a distinct machine learning paradigm characterized by several foundational principles:

1. **Evaluative Feedback:** Unlike supervised learning, there is no external supervisor providing labeled data. The system relies entirely on a reward signal, which indicates the quality of an action (good or bad) without providing the absolute optimal target.
2. **Delayed Consequences:** Feedback is rarely instantaneous; actions taken at the current time step may have profound effects on rewards received many steps in the future.

3. **Sequential Decision-Making:** Time is a critical variable. RL deals with sequential states and actions where data points are highly correlated, fundamentally breaking the independent and identically distributed (i.i.d.) assumption common in classical machine learning.

4. **Agent-Environment Interaction:** An active agent continuously moves through and interacts with a dynamic environment.

5. **Active Sampling:** The agent's actions directly influence the subsequent data it observes, creating a continuous feedback loop between behavior and data generation.

A standard RL system is defined by the following core components:

1. **State (S_t):** A mathematical function of the history H_t (a sequence of sequential observations, actions, and rewards, defined as $H_t = O_1, R_1, A_1, \dots, A_{t-1}, O_t, R_t$) used to determine subsequent dynamics. States can be categorized into three types:

   (a) *Environment State (S_t^e):* The internal representation used by the environment to dictate state transitions and emissions.

   (b) *Agent State (S_t^a):* The internal representation utilized by the agent to summarize historical data and inform future action selection.

   (c) *Information/Markov State (S_t^M):* A formal mathematical property indicating that the state sufficiently retains all necessary historical information. A state is Markovian [4] if the future is conditionally independent of the past, given the present: $\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]$.

2. **Action (A_t):** The behavioral output executed by the agent.

3. **Reward (R_t):** A scalar feedback signal indicating the immediate utility of the agent's behavior at time step t. The primary objective of the agent is to maximize the cumulative sum of these expected rewards over time.

4. **Policy (π):** The agent's behavioral mapping from states to actions, formulated as $\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$. In a stochastic policy, this defines a probability distribution over the action space.

5. **Value Function ($v(s)$):** A prediction of future cumulative rewards, estimating the long-term desirability of being in a particular state or taking a specific action: $v(s) = \mathbb{E}[G_t \mid S_t = s]$, where G_t represents the expected return.

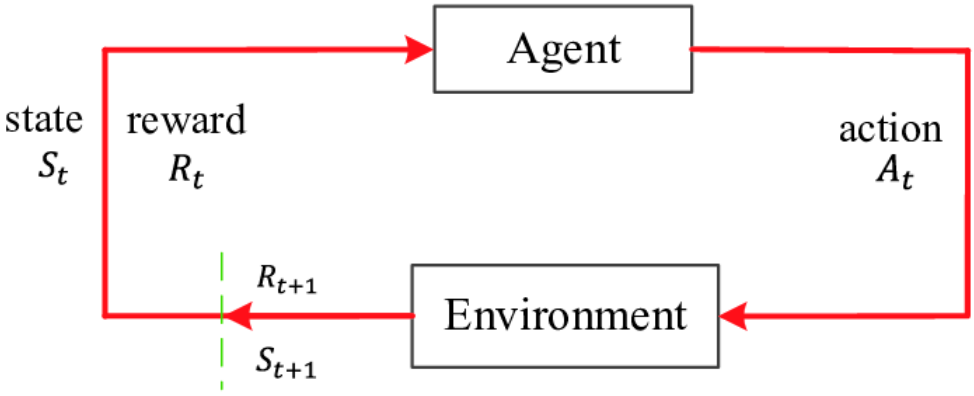


Figure 2.1: Agent-Environment Interaction in a Reinforcement Learning System. Taken from Jiang et al., 2019 [5].

The nature of the environment dictates the agent's observation capabilities. An environment is *fully observable* if the agent can directly observe the complete environment state ($S_t^a = S_t^e$), formally modeled as a Markov Decision Process (MDP). Conversely, if the agent only has access to a limited view of the state ($S_t^a \neq S_t^e$), the system is modeled as a Partially Observable Markov Decision Process (POMDP).

2.1.1 Markov Decision Process (MDP)

A Markov process constitutes a memoryless stochastic sequence of random states $S_1, S_2, \ldots$ that strictly adheres to the Markov Property [4]. An MDP extends this by introducing actions and rewards, formalized as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:

- $\mathcal{S}$: A set of states.
- $\mathcal{A}$: A set of actions.
- $\mathcal{P}$: The state transition probability matrix, where $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$.
- $\mathcal{R}$: The expected reward function, defined as $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$.
- $\gamma \in [0, 1]$: The discount factor, quantifying the present value of future rewards.

The fundamental goal in an MDP is to maximize the expected return G_t, defined as the total discounted cumulative reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

RL agents are fundamentally categorized by their algorithmic approach:

1. **Value-Based:** The agent approximates the value function, while the policy is implicitly derived (e.g., via an ϵ-greedy strategy).
2. **Policy-Based:** The agent explicitly parameterizes and optimizes the policy π without maintaining a separate value function.
3. **Actor-Critic:** A hybrid architecture utilizing both functions: the *actor* updates the policy, while the *critic* evaluates the policy by estimating the value function.

2.1.2 Bellman Expectation Equations

Value functions can be systematically decomposed into two constituent parts: the immediate expected reward and the discounted expected value of the successor state.

1. **State-Value Function ($v_\pi(s)$):** Evaluates the expected return starting in state s and strictly following policy π.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \end{aligned} \tag{2.2}$$

   By marginalizing over discrete actions, this function connects to the action-value function: $v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) q_\pi(s,a)$.

2. **Action-Value Function ($q_\pi(s,a)$):** Evaluates the expected return of taking action a in state s, and subsequently following policy π.

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \tag{2.3}$$

   Similarly, this can be expressed by averaging over the environmental dynamics: $q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$.

By substituting these relationships, we derive the recursive **Bellman Expectation Equations**:

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right) \tag{2.4}$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a' \mid s') q_\pi(s',a') \tag{2.5}$$

2.1.3 Bellman Optimality Equations

The primary objective of RL is to identify the optimal policy π_*, defined as the policy that yields the maximum possible state-value function $v_{\pi_*}(s) = \max_\pi v_\pi(s)$ and action-value function $q_{\pi_*}(s,a) = \max_\pi q_\pi(s,a)$ across all states. According to established theory [6], there always exists at least one optimal deterministic policy $\pi_* \geq \pi$, $\forall \pi$.

This principle leads to the **Bellman Optimality Equations**:

- For the optimal state-value function $v_*(s)$: The value of a state under the optimal policy must equal the expected return for the best available action. It requires a one-step look-ahead, maximizing over the action space and averaging over the stochastic transition dynamics determined by the environment.

$$v_*(s) = \max_a \left[\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \right] \tag{2.6}$$

- For the optimal action-value function $q_*(s,a)$: The value of taking an action relies on the immediate reward plus the discounted expected value of the best possible action in the subsequent state.

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a') \tag{2.7}$$

Due to the inherent non-linearity of the max operator, Bellman optimality equations generally lack a closed-form solution. Consequently, they necessitate iterative algorithmic solvers such as Value Iteration, Policy Iteration [7] , Q-Learning [8], and SARSA [9].

2.1.4 Policy Gradient Methods

In contrast to value-based methods, which derive behavior implicitly from value estimates, policy gradient methods directly parameterize and optimize the policy itself. By representing the policy as a parameterized probability distribution, $\pi_\theta(a|s) = \mathbb{P}[a \mid s; \theta]$, the agent selects actions conditioned on both the current state and the parameter vector θ.

This direct parameterization offers several fundamental **advantages** , as stated in Suton et al. [10]:

- *Enhanced Convergence Properties:* The policy update follows the gradient smoothly, reducing the oscillatory behavior often seen in value-based approximations.
- *Continuous and High-Dimensional Action Spaces:* It naturally accommodates continuous action spaces without the computational bottleneck of calculating a global maximum over actions (as required by the max operator in Q-learning).
- *Stochastic Policies:* It allows the agent to learn and maintain stochastic policies, which is essential in environments with imperfect information or non-stationary dynamics.

Conversely, policy gradient approaches possess notable **disadvantages** , as stated in Sutton et al. [10]:

- *Local Optima:* Gradient ascent guarantees convergence to a local optimum, which may not correspond to the global optimum.
- *High Variance:* Evaluating a policy natively via trajectory sampling is typically highly inefficient and suffers from significant variance, which can destabilize learning.

To optimize the parameters θ, an objective function $J(\theta)$ must be defined to quantitatively evaluate the quality of the policy. The formulation of $J(\theta)$ depends fundamentally on the nature of the environment:

- **Start Value (Episodic Environments):** For episodic environments, the objective is the expected long-term return from the initial state. Because the environment may start in different states according to an initial state distribution $\rho(s_1)$, the objective function is the expectation of the value function over this distribution:

$$J(\theta) = \mathbb{E}_{s_1 \sim \rho}\left[V^{\pi_\theta}(s_1)\right] = \sum_{s_1 \in \mathcal{S}} \rho(s_1) V^{\pi_\theta}(s_1) \tag{2.8}$$

This metric asks: "If the episode begins at a state drawn from the initial distribution ρ and strictly follows π_θ, what is the total expected reward?"

- **Average Value (Continuing Environments):** In non-episodic environments lacking a distinct start state, the objective evaluates the expected value of all states, weighted by the stationary distribution $d^{\pi_\theta}(s)$ of the Markov chain induced by π_θ:

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s) \tag{2.9}$$

- **Average Reward per Time-Step (Continuing Environments):** Alternatively, the objective can be framed as the expected immediate reward per time-step under the stationary distribution:

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(a|s) \mathcal{R}_s^a \tag{2.10}$$

Here, $d^{\pi_\theta}(s)$ acts as a generalized probability distribution over the state space as time approaches infinity.

Remarkably, the gradient of the objective function takes the same underlying form regardless of which specific $J(\theta)$ is utilized.

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta e\right] && \text{TD}(\lambda)\text{ Actor-Critic} \\
G_\theta^{-1} \nabla_\theta J(\theta) &= w && \text{Natural Actor-Critic}
\end{aligned}$$

Figure 2.2: The foundational architecture of Policy Gradient Algorithms, illustrating equivalent mathematical forms across different objective functions.

Policy Gradient Algorithms

The core mechanism of these algorithms relies on the Policy Gradient Theorem, which states that if the policy π_θ is differentiable, the parameters can be updated via stochastic gradient ascent. The general form of the gradient is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)] \tag{2.11}$$

- **Monte Carlo Policy Gradient (REINFORCE):** To approximate the expected return without requiring a complete model of the environment dynamics, the REINFORCE algorithm relies on empirical returns v_t sampled from full trajectories as an unbiased estimate of the true action-value function $Q^{\pi_\theta}(s_t, a_t)$.

  $$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) v_t \tag{2.12}$$

  The gradient term $\nabla_\theta \log \pi_\theta(a_t|s_t)$, known as the score function, effectively steers the parameter updates to increase the probability of actions associated with higher empirical returns. However, the reliance on full Monte Carlo rollouts renders this approach computationally slow and introduces substantial variance, potentially destabilizing the learning process [6].

- **Actor-Critic Policy Gradient:** To alleviate the high variance inherent to Monte Carlo sampling, Actor-Critic architectures introduce a parameterized value function, termed the *Critic* ($Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$), to explicitly estimate the expected return. This substitution yields the approximate policy gradient:

  $$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_w(s,a)] \tag{2.13}$$

  This architecture fundamentally comprises two interacting components:

  - The **Critic**, which updates its parameters w to accurately evaluate current state-action pairs by minimizing the temporal-difference error.
  - The **Actor**, which updates the policy parameters θ in the direction of the gradient formulated by the Critic's evaluations.

  Action-Value Actor-Critic with Baseline: In a standard formulation utilizing linear value function approximation ($Q_w(s,a) = \phi(s,a)^T w$),

the iterative updates are defined as follows:

$$\delta = r + \gamma Q_w(s', a') - Q_w(s, a) \quad \text{(TD Error)} \tag{2.14}$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) Q_w(s, a) \quad \text{(Actor Update)} \tag{2.15}$$

$$w \leftarrow w + \beta \delta \phi(s, a) \quad \text{(Critic Update)} \tag{2.16}$$

To further reduce the variance of the gradient estimator without introducing bias (i.e., without altering the expectation of the gradient), an arbitrary state-dependent baseline function $B(s)$ can be subtracted from the value estimate. This subtraction preserves the expected gradient because the expectation of the baseline term rigorously evaluates to zero:

$$\begin{aligned} \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s) B(s)] &= \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a \mid s) B(s) \\ &= \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) B(s) \nabla_\theta \left(\sum_{a \in \mathcal{A}} \pi_\theta(a \mid s) \right) \\ &= 0 \end{aligned} \tag{2.17}$$

The final equality holds because the sum over all action probabilities in a given state must equal 1 ($\sum_{a \in \mathcal{A}} \pi_\theta(a \mid s) = 1$), and the gradient of a constant is mathematically zero.

The theoretically optimal choice for this baseline is the state-value function $V^{\pi_\theta}(s)$, which quantifies the average expected performance starting from state s. Subtracting this baseline from the action-value function yields the **Advantage Function** $A^{\pi_\theta}(s, a)$:

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \tag{2.18}$$

By substituting the Advantage function into the objective, the modified policy gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A^{\pi_\theta}(s, a)] \tag{2.19}$$

This formulation profoundly stabilizes the learning trajectory. Instead of updating parameters based on the absolute magnitude of the expected return (Q), the gradient is scaled by the relative advantage (A). That is, how much better the chosen action performed compared to the average expected baseline for that state. This centering mechanism mitigates

noise and yields significantly more efficient convergence.

2.1.5 Advantage Actor-Critic (A2C)

Advantage Actor-Critic (A2C) algorithm is a synchronous, deterministic variant of the Asynchronous Advantage Actor-Critic (A3C) method. It operates by coordinating parallel environment data collection and executing batch updates. By simulating N parallel environments simultaneously and updating the policy based on the aggregated batch, A2C stabilizes the training process; rather than updating parameters based on a single, high-variance trajectory, the gradient is averaged across N diverse trajectories.

A fundamental distinction between A2C and traditional Q-Learning Actor-Critic architectures is that A2C does not directly approximate the action-value function $Q(s,a)$. Instead, it relies on empirical returns collected from the environments to compute the advantage function. This reliance on grounded trajectory data , rather than initially biased action-value approximator ,contributes significantly to the improved stability of A2C [11].

Algorithm Overview:

1. Initialize network parameters θ with random weights.
2. Execute N steps across parallel environments using the current policy π_θ, storing the transition tuples (s_t, a_t, r_t).
3. Initialize the return G_t: set $G_t = 0$ if a terminal state is reached; otherwise, bootstrap the value estimate $G_t = V_\theta(s_t)$.
4. For $i = t-1$ down to t_{start} (iterating backward for computational efficiency):
   (a) Accumulate the discounted return: $G_t \leftarrow r_i + \gamma G_t$.
   (b) Compute the Advantage function, defined as the difference between the empirical return and the baseline expected return estimated by the critic: $A(s_i, a_i) = G_t - V_\theta(s_i)$.
   (c) Accumulate the policy gradients (Actor update):

$$\Delta\theta_\pi = \Delta\theta_\pi + \nabla_{\theta_\pi} \log \pi_{\theta_\pi}(a_i \mid s_i)\,(G_t - V_\theta(s_i))$$

   (d) Accumulate the value gradients (Critic update) by minimizing the Mean Squared Error (MSE):

$$\Delta\theta_v = \Delta\theta_v + \nabla_{\theta_v}\left(G_t - V_{\theta_v}(s_i)\right)^2$$

5. Update the neural network parameters in the direction of the accumulated gradients using an optimization algorithm (e.g., Adam).

6. Repeat from step 2 until convergence.

In practice, the A2C loss function is often augmented with an entropy regularization term. This term quantifies the uncertainty of the agent's policy, penalizing premature convergence to deterministic behavior. Higher entropy promotes exploration by encouraging the agent to sample diverse actions, whereas lower entropy signifies exploitation of known optimal actions.

2.1.6 Trust Region Policy Optimization (TRPO)

Schulman et al. [12] introduced Trust Region Policy Optimization (TRPO), an algorithm designed to ensure monotonic policy improvement. The problem is formalized as a Markov Decision Process (MDP) defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma \rangle$, where $\mathcal{S}$ is a finite state space, $\mathcal{A}$ is a finite action space, $\mathcal{P}$ represents the transition probability dynamics, r is the reward function, ρ_0 is the initial state distribution, and $\gamma \in (0, 1)$ is the discount factor.

For a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, the expected discounted return $\eta(\pi)$ is defined as:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right] \tag{2.20}$$

where $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi(a_t \mid s_t)$, and $s_{t+1} \sim \mathcal{P}(s_{t+1} \mid s_t, a_t)$.

The algorithm utilizes the standard definitions of the action-value function $Q_\pi(s_t, a_t)$, state-value function $V_\pi(s_t)$, and advantage function $A_\pi(s, a)$:

$$\begin{aligned} Q_\pi(s_t, a_t) &= \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right] \\ V_\pi(s_t) &= \mathbb{E}_{a_t, s_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right] \\ A_\pi(s, a) &= Q_\pi(s, a) - V_\pi(s) \end{aligned} \tag{2.21}$$

Given a current policy π and a candidate policy $\tilde{\pi}$, the expected return of $\tilde{\pi}$ can be expressed in terms of the advantage over π (as proven in Lemma 1 of [12]):

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \ldots \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right] \tag{2.22}$$

This formulation allows the evaluation of a new policy based on the baseline performance of the current policy. Introducing the unnormalized discounted visitation distribution $\rho_\pi(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s)$, which quantifies the visitation frequency of state s, the expected return can be rewritten as:

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{s \in \mathcal{S}} \rho_{\tilde{\pi}}(s) \sum_{a \in \mathcal{A}} \tilde{\pi}(a \mid s) A_\pi(s, a) \tag{2.23}$$

Standard policy gradient methods suffer from two primary limitations: first, the gradient estimate is only a first-order local approximation; second, standard step sizes can lead to overshooting (resulting in performance collapse) or excessively slow convergence.

To address this, TRPO enforces a trust region to mathematically guarantee monotonic improvement, ensuring $\eta(\pi_{\theta_{k+1}}) \geq \eta(\pi_{\theta_k})$. The algorithm achieves this by maximizing a surrogate objective function $\mathcal{L}(\theta_k, \theta)$ subject to a strict constraint on the Kullback-Leibler (KL) divergence between the old and new policies:

$$\begin{aligned} \theta_{k+1} = \arg\max_{\theta} \; & \mathcal{L}(\theta_k, \theta) \\ \text{s.t. } & \bar{D}_{KL}(\theta \parallel \theta_k) \leq \delta \end{aligned} \tag{2.24}$$

Where the surrogate objective is defined as:

$$\mathcal{L}(\theta_k, \theta) = \mathbb{E}_{s,a \sim \pi_{\theta_k}} \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} A^{\pi_{\theta_k}}(s, a) \right] \tag{2.25}$$

And the average KL-divergence penalty prevents destructively large updates:

$$\bar{D}_{KL}(\theta \parallel \theta_k) = \mathbb{E}_{s \sim \pi_{\theta_k}} \left[D_{KL} \left(\pi_\theta(\cdot \mid s) \parallel \pi_{\theta_k}(\cdot \mid s) \right) \right] \tag{2.26}$$

Because solving this constrained optimization problem exactly is intractable for large neural networks, TRPO employs a Taylor expansion to form a linear approximation of the objective and a quadratic approximation of the constraint:

$$\begin{aligned} \theta_{k+1} \approx \arg\max_{\theta} \; & g^T(\theta - \theta_k) \\ \text{s.t. } & \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) \leq \delta \end{aligned} \tag{2.27}$$

Here, $g = \nabla_\theta \mathcal{L}(\theta_k, \theta)|_{\theta = \theta_k}$ is the gradient vector, and H is the Fisher Information Matrix (the Hessian of the average KL-divergence). Using

Lagrangian duality [13, Ch. 2], the theoretical update step is derived as:

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g \tag{2.28}$$

where $\alpha \in (0, 1)$ is a backtracking line-search coefficient. Since computing the inverse Hessian H^{-1} is computationally prohibitive for deep networks, TRPO uses the conjugate gradient algorithm [14] to efficiently estimate the natural gradient direction $x = H^{-1}g$ via matrix-vector products ($Hx = g$) without explicitly forming H.

Algorithm 1 Trust Region Policy Optimization

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K
3: **for** $k = 0, 1, 2, \ldots$ **do**
4: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
5: Compute rewards-to-go $\hat{R}_t$.
6: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
7: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

8: Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.
9: Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

where $j \in \{0, 1, 2, \ldots K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.
10: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
11: **end for**

Figure 2.3: TRPO algorithm pseudocode. Adapted from OpenAI Spinning Up [15].

2.1.7 Proximal Policy Optimization (PPO)

To overcome the computational complexity of second-order optimization in TRPO, Schulman et al. [16] introduced PPO. PPO retains the reliability and monotonic improvement guarantees of TRPO while significantly simplifying the implementation by utilizing purely first-order optimization techniques.

While the original paper proposes two variants (PPO-Penalty and PPO-Clip), the most widely adopted formulation is PPO-Clip. Rather than imposing a hard KL-divergence constraint, PPO-Clip alters the objective function to mathematically penalize excessive divergence between the updated policy and the old policy.

Let $r_t(\theta)$ denote the probability ratio between the target and behavior policies:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{2.29}$$

The clipped surrogate objective function is formulated as:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2.30}$$

By explicitly clipping the probability ratio within the interval $[1-\epsilon, 1+\epsilon]$, the algorithm removes the incentive to move r_t outside of this bound when the advantage is positive, thereby establishing a local "trust region."

To compute the advantage estimates $\hat{A}_t$, PPO commonly integrates Generalized Advantage Estimation (GAE). GAE computes an exponentially weighted average of multi-step temporal-difference errors ($\delta_t = r_t + \gamma V_w(s_{t+1}) - V_w(s_t)$) to strike an optimal balance between bias and variance in the value estimations.

V_w represents the value function approximated by a neural network (or another function approximator), and w represents the weights (or parameters) of that network.

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
  Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

  typically via stochastic gradient ascent with Adam.
  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.
end for

Figure 2.4: PPO algorithm pseudocode. Adapted from OpenAI Spinning Up [15].

Advantages of PPO:

- *Computational Efficiency:* By relying exclusively on first-order derivatives (gradients), PPO avoids the expensive Fisher Information Matrix computations of TRPO, leading to vastly superior wall-clock training times.
- *Stability and Robustness:* The surrogate clipping mechanism safely restricts step sizes, preventing the catastrophic performance collapse common in standard policy gradients.
- *Versatility:* It is highly scalable and consistently performs well across continuous and discrete action spaces.

Limitations of PPO:

- *Sample Inefficiency:* As an on-policy algorithm, PPO requires fresh data generated by the current policy for every update phase, rendering it inherently less sample-efficient than off-policy algorithms (e.g., SAC, DQN) which utilize experience replay buffers.

- *Hyperparameter Sensitivity:* Performance is sensitive to specific tuning parameters, particularly the clipping ratio (ϵ), learning rate, and GAE coefficient (λ).
- *Local Optima Vulnerability:* Because the clipping function only enforces a local trust region, noisy gradient estimates or excessively small batch sizes can still cause the policy to drift into suboptimal local minima.

2.2 Architectural Paradigms and Constraint Handling in Policy Optimization

To contextualize the design choices made in this thesis, this section explores established technical literature regarding neural network architectures in reinforcement learning. It specifically focuses on strategies for feature engineering, augmenting the Actor's decision-making framework, and scaling the structural complexity of the Critic

2.2.1 The Sensory Neuron as a Transformer: Permutation-Invariant Neural Networks for Reinforcement Learning

The work by Tang et al. (2021) [17] investigates the property of Permutation Invariance (PI) within neural networks, a concept that is highly central to the methodology developed in this thesis. A fundamental premise of their research is that neural networks should be capable of recognizing that a set of items represents the same collective entity, regardless of the order in which those items are presented. The authors formally demonstrate that forcing an agent to learn a coherent representation of a permutation-invariant observation space leads to more robust and reliable policies, which exhibit superior generalization when evaluated in unseen environments.

Mathematically, an agent's policy is considered Permutation-Invariant if its output remains unchanged when the input elements are reordered. Formally, for a function $f : \mathbb{R}^n \to \mathbb{R}^m$ and any permutation matrix P, the function is PI if $f(Px) = f(x)$ for all $x \in \mathbb{R}^n$. The authors also explore a related paradigm known as Permutation Equivariance (PE). PE applies when the dimensions of the input and output spaces are equal ($f : \mathbb{R}^n \to \mathbb{R}^n$), such

that applying a permutation to the input results in the exact same permutation being applied to the output: $f(Px) = Pf(x)$.

To achieve PI, Tang et al. augmented the agent's policy network (π) with an additional layer termed the *AttentionNeuron*. This layer comprises a decentralized set of "sensory neurons," each possessing a restricted receptive field of the observation space. Specifically, at time step t, the i-th sensory neuron observes only the i-th component of the observation vector, $o_t[i]$, alongside the previous action, a_{t-1}. Each sensory neuron independently computes messages by applying shared transformation functions $f_k(o_t[i], a_{t-1})$ and $f_v(o_t[i])$. Subsequently, an attention mechanism (the detailed mechanics of which fall outside the primary scope of this review) aggregates these individual messages to generate a global latent code m_t. This latent representation is then broadcasted to the downstream components of the network.

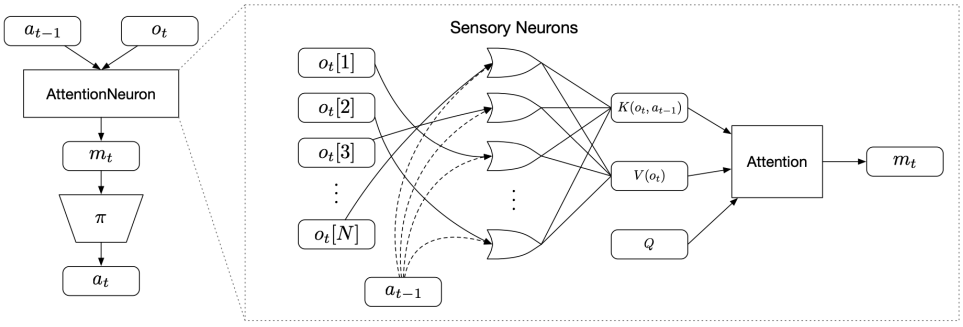


Figure 2.5: Architecture of the neural network for the permutation-invariant agent. Adapted from Tang et al., 2021 [17].

For their implementation, the authors opted for a Feed-Forward Neural Network (FFNN) rather than a Recurrent Neural Network (RNN), noting that FFNNs are better suited for processing high-dimensional inputs in this context. Furthermore, they applied normalization techniques to both the inputs and outputs of the network, utilizing a softmax non-linear activation function.

Technical Contribution and Thesis Relevance

A significant finding from Tang et al. is that agents utilizing this architecture maintain high performance even when the ordering of observations is randomly permuted multiple times during an episode. Remarkably, this robust performance is achieved without explicitly training the agent on highly re-shuffled data. However, the authors note that while permutation invariance is naturally applicable to observation spaces, seamlessly extending this property

to action spaces remains an ongoing challenge.

This paper provided crucial foundational insights for designing the neural network architecture of the actor in this thesis. Because the scheduling algorithm evaluates sets of locomotives , whether ranking them for maintenance or dispatching them to trains, it was imperative that the model exhibit permutation invariance with respect to the ordering of the locomotive array. While the architecture implemented in this thesis represents a computationally cheaper, structurally simplified version of Tang et al.'s model, the core principles for managing unordered sets and ensuring representational consistency were directly inspired by their findings.

2.2.2 A Closer Look at Invalid Action Masking in Policy Gradient Algorithms

In the context of this thesis, several actions within the state space are operationally infeasible; for instance, a locomotive cannot be assigned to a service if it is currently undergoing maintenance or has already departed the terminal. Huang et al. [18] provide a comprehensive analysis of *invalid action masking*, a technique widely utilized in DRL to manage such constraints, and one that serves as a cornerstone of the implementation presented here.

DRL algorithms applied to complex environments, such as Real-Time Strategy (RTS) games or industrial logistics, often contend with high-dimensional discrete action spaces characterized by numerous rules and constraints. In such settings, random sampling from the full action space frequently results in the selection of invalid actions. The standard mitigation strategy for policy gradient algorithms involves invalid action masking [10], which effectively filters out prohibited actions, allowing the agent to sample exclusively from the valid subset. Huang et al. evaluate the efficacy of this approach in comparison to the *invalid action penalty* strategy, where the agent receives a negative reward for selecting infeasible actions to discourage such behavior.

The framework proposed by Huang et al. [18] aligns with the architecture utilized in this research, where the policy is defined by a neural network outputting unnormalized scores, or *logits*. These logits are converted into a probability distribution via a softmax function. Consider a simplified action space $\mathcal{A} = \{a_0, a_1, a_2, a_3\}$ and a state s_0. The parameterized policy π_θ—where θ corresponds to the logits $[l_0, l_1, l_2, l_3]$—is computed as follows:

$$\begin{aligned}\pi_\theta(\cdot|s_0) &= [\pi_\theta(a_0|s_0), \pi_\theta(a_1|s_0), \pi_\theta(a_2|s_0), \pi_\theta(a_3|s_0)] \\ &= \text{softmax}([l_0, l_1, l_2, l_3]) \\ &= [0.25, 0.25, 0.25, 0.25] \end{aligned} \tag{2.31}$$

where the probability of action a_i is defined by:

$$\pi_\theta(a_i|s_0) = \frac{\exp(l_i)}{\sum_j \exp(l_j)} \tag{2.32}$$

If an action is sampled (e.g., a_0) while another action is invalid (e.g., a_2), the standard policy gradient is derived as:

$$\begin{aligned} g_{\text{policy}} &= \mathbb{E}_\tau \left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) G_t \right] \\ &= \nabla_\theta \log \pi_\theta(a_0|s_0) G_0 \\ &= [0.75, -0.25, -0.25, -0.25] \end{aligned} \tag{2.33}$$

based on the derivative of the softmax function:

$$(\nabla_\theta \log \text{softmax}(\theta)_j)_i = \begin{cases} 1 - \pi_\theta(a_j|s) & \text{if } i = j \\ -\pi_\theta(a_i|s) & \text{otherwise} \end{cases} \tag{2.34}$$

The invalid action masking framework facilitates the sampling of valid actions by replacing the logits of invalid actions with a large negative constant M (e.g., $M = -1 \times 10^9$). This ensures that the post-softmax probability of invalid actions effectively approaches zero:

$$\begin{aligned}\pi'_\theta(\cdot|s_0) &= \text{softmax}(\text{mask}([l_0, l_1, l_2, l_3])) \\ &= \text{softmax}([l_0, l_1, M, l_3]) \\ &= [\pi'_\theta(a_0|s_0), \pi'_\theta(a_1|s_0), \epsilon, \pi'_\theta(a_3|s_0)] \\ &\approx [0.33, 0.33, 0.00, 0.33] \end{aligned} \tag{2.35}$$

Upon completion of the trajectory, the policy is updated using the *invalid*

action policy gradient:

$$\begin{aligned} g_{\text{invalid action policy}} &= \mathbb{E}_\tau \left[\nabla_\theta \sum_{t=0}^{T-1} \log \pi'_\theta(a_t|s_t) G_t \right] \\ &= \nabla_\theta \log \pi'_\theta(a_0|s_0) G_0 \\ &= [0.67, -0.33, 0.00, -0.33] \end{aligned} \tag{2.36}$$

Significantly, this framework does not merely renormalize the distribution; it fundamentally forces the policy gradient of invalid actions to zero, preventing them from influencing the parameter updates.

Huang et al. compare four distinct strategies:

- Invalid Action Penalty: A negative reward is assigned for sampling infeasible actions [19].
- Invalid Action Masking: The distribution is masked to ensure only valid actions are available for sampling and gradient calculation.
- Naive Invalid Action Masking: The standard probability distribution is calculated, but the gradient for invalid actions is manually set to zero.
- Masking Removed: Masking is applied during training but disabled during evaluation to test policy robustness.

The empirical results of the study conclude that:

- Invalid action masking scales effectively to complex, high-dimensional tasks.
- Invalid action penalty fails to scale, as the agent struggles to explore valid regions of the state space.
- Naive invalid action masking causes the KL divergence to explode, leading to training instability.
- Masking removed allows the agent to maintain performance to a significant extent, suggesting the agent learns the underlying constraints.

2.2.3 Action Space Shaping in Deep Reinforcement Learning

Kanervisto et al. [20] provide a formal framework for **Action Space Shaping**, the process of transforming the original action space of an environment $\mathcal{A}$ into a modified space $\mathcal{A}'$ to facilitate more efficient policy optimization. This paper is essential for addressing the "Exploration Bottleneck" that occurs when an agent must select a subset of assets from a large, discrete pool. The authors define action shaping as a mapping function $g : \mathcal{A}' \to \mathcal{A}$, where the agent learns a policy $\pi_\theta(a'|s)$ over the shaped space $\mathcal{A}'$. The fundamental motivation for this paper is the reduction of the search space cardinality. In many real-world environments, the raw action space $\mathcal{A}$ is combinatorial; for a set of N assets where k must be selected, the cardinality $|\mathcal{A}| = \binom{N}{k}$ grows exponentially with N. This exponential growth leads to the Sparse Probability Problem: in a standard Categorical distribution, the probability of selecting the optimal combination a^* by random exploration is:

$$P(\text{exploration success}) = \frac{1}{\binom{N}{k}} \tag{2.37}$$

As N increases, this probability approaches zero, rendering standard stochastic exploration (e.g., ϵ-greedy or Entropy-based exploration) ineffective. Kanervisto et al. argue that by reshaping $\mathcal{A}$ into a continuous space $\mathcal{A}' = [0, 1]^N$, the agent can utilize the topology of the continuous space to "rank" assets, effectively transforming the search from a needle-in-a-haystack discrete problem into a smoother, gradient-friendly optimization.

Technical Contribution: The paper provides a mathematical justification for using Continuous Proxies for discrete decisions. Instead of the agent outputting a single discrete class, it outputs a vector of continuous weights $\mathbf{w} \in \mathbb{R}^N$. The mapping function $g(\mathbf{w})$ then converts these weights into a discrete action.

This paper addresses three critical failures in standard DRL:

1. Gradient Thinning: In a large discrete space, the gradient $\nabla_\theta J(\theta)$ is computed over a massive Softmax output where most entries are near zero. This leads to vanishing gradients. Shaping concentrates the "probability mass" on relevant features.

2. Action Entanglement: In many systems, actions are not independent. Selecting "Locomotive A" might depend on the state of "Locomotive

B." A shaped continuous space allows the network to learn relational "importance" scores rather than memorizing isolated discrete pairs.

3. Exploration Efficiency: By using a continuous distribution (e.g., Gaussian $\mathcal{N}(\mu, \sigma)$) over weights, the agent can explore "close" strategies by slightly perturbing μ. In a discrete space, there is no concept of "closeness"; the distance between any two discrete actions is essentially binary.

By adopting the strategies in [20], a system can ensure that the learning signal remains dense even when the physical requirements of the task (e.g., assigning specific units to specific trains) are highly complex and discrete.

2.2.4 Addressing Function Approximation Error in Actor-Critic Methods

Fujimoto et al. [21] introduced the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, which addresses the systematic overestimation of value functions in actor-critic methods. While overestimation bias was well-documented in discrete settings (leading to the development of Double DQN), this research formally identifies its presence and catastrophic impact on continuous control. The authors identify that in the presence of function approximation errors, the standard Bellman equation leads to a persistent positive bias in value estimates, which is subsequently exploited by the policy, leading to suboptimal behavior or training divergence. To counteract this, TD3 introduces three primary mechanisms:

1. *Clipped Double-Q Learning:* The agent maintains two independent critics, $(Q_{\theta_1}, Q_{\theta_2})$, and utilizes the minimum of the two estimates for the Bellman target. This enforces a conservative estimate of the value function.

2. *Target Policy Smoothing:* By adding small amounts of random noise to the target actions during the update, the algorithm smooths out peaks in the Q-function, preventing the policy from hyper-fixating on narrow, overestimated regions of the action space.

3. *Delayed Policy Updates:* The actor (policy) and target networks are updated less frequently than the critic. This ensures that the value function has sufficiently converged and provides a stable gradient before the policy is allowed to shift.

Technical Contribution: This paper provides a critical framework for stabilizing the *Critic* in complex environments. By decoupling the frequency of value estimation from policy optimization, it ensures that the gradient signal provided to the Actor is grounded in a converged manifold. Mathematically, the target value y is computed as:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\phi'}(s') + \epsilon), \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c) \tag{2.38}$$

The inclusion of the min operator effectively treats the function approximation error as a noise-reduction problem. For architectures utilizing specialized logic (such as shared-weight Multi-Layer Perceptrons (MLPs) or complex selectors), this stabilization is essential because it prevents the "value loss" from spiking when the agent discovers new regions of the state space.

2.3 Locomotive Assignment Problem (LAP)

The Locomotive Assignment Problem (LAP) constitutes a fundamental challenge in railway logistics, involving the optimal allocation of a motive power fleet to a predetermined set of scheduled services while satisfying complex operational constraints and minimizing aggregate expenditures. This review is primarily informed by the seminal survey of Piu and Speranza (2014) [1], which provides a taxonomy of algorithms and mathematical programming approaches to the problem. Historically, much of the research in this domain originated in North America, which manages one of the most extensive railroad infrastructures globally (omprising approximately 140,000 miles) and represents one of the most cost-efficient freight systems in existence. However, in the decade since the publication of Piu and Speranza's survey, significant contributions have emerged from other global regions, reflecting the growing universal demand for operational efficiency.

In accordance with standard hierarchical planning frameworks, locomotive scheduling is analyzed across three distinct levels based on the planning horizon:

- **Strategic Level:** Focuses on long-term fleet sizing and composition. At this stage, individual locomotive identifiers (IDs) are neglected, and the train schedule is considered immutable, typically disregarding delays or stochastic disruptions.
- **Tactical and Operational Levels:** These levels involve a lower degree of abstraction, necessitating the assignment of specific locomotive units

to services. These models must account for real-time constraints, such as fueling schedules and maintenance intervals, while maintaining robustness against operational disruptions and variance in train schedules.

A critical distinction must be drawn between passenger and freight rail operations, as the latter presents a significantly higher degree of combinatorial complexity. The operational disparities between these two systems are synthesized in Table 2.1.

Feature	**Passenger Trains**	**Freight Trains**
Scheduling Logic	Fixed, time-sensitive schedules	Tonnage-based or unscheduled departures
Network Priority	High (Time-critical priority)	Lower (Subject to delays/sidings)
Motive Power	1–2 active locomotives	2–3x more active units (up to 12 total) according to Ahuja et al. (2005b) [2]
Tractive Effort	Standard axle configurations	High active axle count (up to 24 axles) according to Ahuja et al. (2005b) [2]
Tonnage Variance	Low; predetermined	High; high variance across service types
Consist Diversity	Homogeneous/Standardized	Heterogeneous (Intermodal, automotive, etc.) [3]
Operational Complexity	Low connection constraints	High (Consist management, compatibility)

Table 2.1: Comparative analysis of operational and physical characteristics between passenger and freight rail systems.

LAP formulations can be classified according to their objective functions (e.g., cost minimization or fleet utilization) or their planning depth. A primary distinction is made between *single-locomotive models*, where a service is powered by a single unit, and *multiple-locomotive models*. The latter, which is the focus of this thesis, represents a significantly more complex problem where a train's power requirements are met by a combination of potentially heterogeneous units.

The seminal research by Florian et al. (1976) [22] was among the first to address multiple locomotive requirements, aiming to minimize capital

investment and maintenance costs over a long-term horizon. However, the computational limitations of that era restricted the scalability of their optimization techniques. Subsequently, Scholz (2000) [23] examined the Swedish railway system—the same context explored in this research—aiming to satisfy a schedule within defined departure windows using a minimal fleet size. Scholz utilized Gantt charts for visualization, where rectangular blocks represent trip durations, a methodology mirrored in the implementation of this thesis. Notably, Scholz is credited with integrating preventive maintenance into scheduling through Lagrangian relaxation, though this method is often limited in its ability to scale to high-dimensional, real-time problems.

A paradigm shift occurred with the introduction of the Approximate Dynamic Programming (ADP) framework by Powell et al. (2001) [24]. This represented a transition from traditional mathematical programming toward learning-based heuristics. Rather than solving a global ”batch” optimization for an entire week, the ADP model learns the value of locomotive positioning across time and space. Powell argued that traditional Integer Programming (IP) models are too rigid for environments characterized by stochastic delays and mechanical failures. By decomposing the problem into sequential decisions, the ADP framework estimates the long-term impact of current assignments via value function approximation. This work is widely regarded as a precursor to Multi-Agent Reinforcement Learning (MARL) in railway research, laying the groundwork for the autonomous, proactive agents utilized in this study.

Further advancements by Vaidyanathan et al. [3] proposed the Consist Flow Formulation (CFF) and the Hybrid Flow Formulation (HFF). CFF extends the Locomotive Flow Formulation (LFF) of Ahuja et al. [2] by treating locomotive ”consists” (pre-grouped sets of units) as single commodities. While these structural decompositions improved the management of consist-level rules, they often fail to model individual unit-level constraints, such as specific fueling requirements or servicing feasibility. Table 2.2 provides a chronological summary of these foundational contributions.

Authors	Problem Type	Planning	Objective Function	Method
Florian et al. (1976)	Multiple units	Strategic	Min investment/maint.	Benders decomp.
Scholz (2000)	Multiple units	Strategic	Min fleet size	Heuristic
Powell et al. (2001)	Units and cars	Operational	Min operating costs	ADP
Ahuja et al. (2005)	Single unit	Strategic	Min operating costs	Goldberg-Tarjan
Vaidyanathan et al. (2008)	Single unit	Strategic	Min operating costs	Goldberg-Tarjan

Table 2.2: Taxonomy of foundational Locomotive Assignment Problem (LAP) models (Adapted from Piu and Speranza, 2014).

2.3.1 An advanced learning environment and a scalable deep reinforcement learning approach for rolling stock circulation on urban rail transit line

Yang et al. [25] addresses the Rolling Stock Circulation (RSC) problem by formulating it as a multi-objective optimization task within a DRL framework. The primary objectives identified are the minimization of fleet utilization, the balancing of maintenance workloads, and the stabilization of depot stock levels.

State Space Representation

The authors define a hierarchical state representation where the global state s^t is a concatenation of individual rolling stock statuses s_k^t. The local state for each asset k is defined as a six-dimensional feature vector:

$$s_k^t = (n_k^t, \bar{c}_k^t, \bar{b}_k^t, f_k^t, o_k^t, g_k^t)$$

This vector captures critical operational constraints, including cumulative trip counts (n_k^t), remaining service life ($\bar{c}_k^t$), and turnaround readiness (f_k^t). Notably, the inclusion of operation direction (o_k^t) and a timetable pointer (g_k^t) ensures the agent maintains the Markov property by encoding the physical feasibility of future assignments.

Action Space

The action space $A = \{1, 2, \dots, |K|\}$ is discrete, where each action corresponds to the selection of a specific physical asset from the fleet. To handle the "hard constraints" of rail operations (e.g., assigning a busy train), the authors utilize a penalty-based approach rather than strict action masking. This forces the agent

to learn the underlying feasibility logic through the negative reinforcement of invalid selections.

Reward Mechanism

A dual-reward structure is employed to handle the credit assignment problem:

1. **Dense Step-wise Reward (r'):** Focused on immediate utilization efficiency and constraint satisfaction. It penalizes unnecessary depot extractions (r_a) and rewards the selection of idle assets with high turnaround times (r_v), promoting a "First-In, First-Out" logic.

2. **Sparse Terminal Reward (Z):** A weighted sum of normalized objectives calculated at the end of the operation horizon. This term targets fleet-wide fairness (workload standard deviation z_3) and depot equilibrium (z_2).



(a) Actor Network (b) Critic Network

Figure 2.6: Architecture of the proposed Actor-Critic Neural Network. Taken from (Yang et al., 2025 [25])

Neural Network Architecture

The problem is solved using PPO algorithm. As illustrated in Figure 2.6, the Actor and Critic networks utilize a fully-connected MLP structure with hyperbolic tangent (tanh) activation functions.

Technical Contribution and Relevance to Thesis

This paper provides the foundational logic for my research, particularly in the structuring of sparse and dense rewards to balance local efficiency with global maintenance goals. However, while the authors utilize a global MLP that scales linearly with fleet size, my work improves upon this by implementing a **Shared-Weight MLP**. This ensures that the model is permutation-invariant and can generalize to varying fleet sizes without re-training—a significant limitation observed in the original paper's architecture.

Feature	Reference Paper	This Thesis
Algorithm	PPO (Actor-Critic)	PPO (Actor-Critic)
Action Space	Discrete (Fixed $\|K\|$)	Continuous Weights (Scalable)
NN Architecture	Global Flat MLP	Shared-Weight Architecture
Observation Space	Fixed Global Vector	Masked Relational Features
Maintenance Logic	Reactive (Fixed Thresholds)	Proactive (Weibull-based)
Simulation Type	Step-based	Event-driven

Table 2.3: Comparative analysis between the reference study and the proposed DRL approach.

2.3.2 Deep Reinforcement Learning for Integrated Train Scheduling and Rolling Stock Circulation

A significant contribution to integrated rail optimization is presented by Zhao et al. [26], who propose a DRL approach to solve the joint problem of train scheduling and RSC. The authors utilize a Hybrid Proximal Policy Optimization (HPPO) algorithm to manage a hybrid action space characterized by both continuous scheduling decisions and discrete rolling stock assignments. The primary objective is to minimize a multi-objective cost function comprising passenger waiting time and operational expenses under complex constraints.

In this framework, the DRL agent acts as a dispatcher, sequentially determining section running times, station dwell times, and the allocation of rolling stock units—either by deploying units from the depot or reassigning units arriving from previous services.

State Space Representation

The state space S_n at step n captures the temporal and logistical dynamics of the network:

$$S_n = (d_{n,i}, a_{n,i}, l_{n,i}, o_{n,k}) \tag{2.39}$$

where $d_{n,i}$ and $a_{n,i}$ denote departure and arrival times at station i, $l_{n,i}$ represents the passenger queue length post-departure, and $o_{n,k}$ indicates the availability time of rolling stock unit k.

Action Space Representation

The action space A_n is hybrid, formulated as:

$$A_n = (h_{n+1}, r_{n+1,i}, e_{n+1,i}, f_{n+1,k}) \tag{2.40}$$

Here, the headway (h_{n+1}), section running times ($r_{n+1,i}$), and dwell times

($e_{n+1,i}$) are continuous variables, while the rolling stock assignment ($f_{n+1,k} \in \{0,1\}$) is a discrete binary decision.

Reward Mechanism

The reward function is defined as the negative weighted sum of three components:

1. *Passenger Waiting Costs*: Calculated as the sum of waiting time for newly arrived passengers and those stranded due to capacity limits from previous services.
2. *Operational Costs*: Defined by the total cycle time for service n, calculated as $(a_{n,2I} - d_{n,I})$.
3. *Usage Costs*: Proportional to the total number of active rolling stock units $\sum_{k \in K} \eta_k$.

Neural Network Architecture

To handle the hybrid nature of the actions, the architecture employs a dual-actor system 2.7. A discrete actor (π_{θ_d}) outputs the probability distribution for rolling stock assignments, while a continuous actor (π_{θ_c}) parameters a multivariate Gaussian distribution to sample scheduling variables.

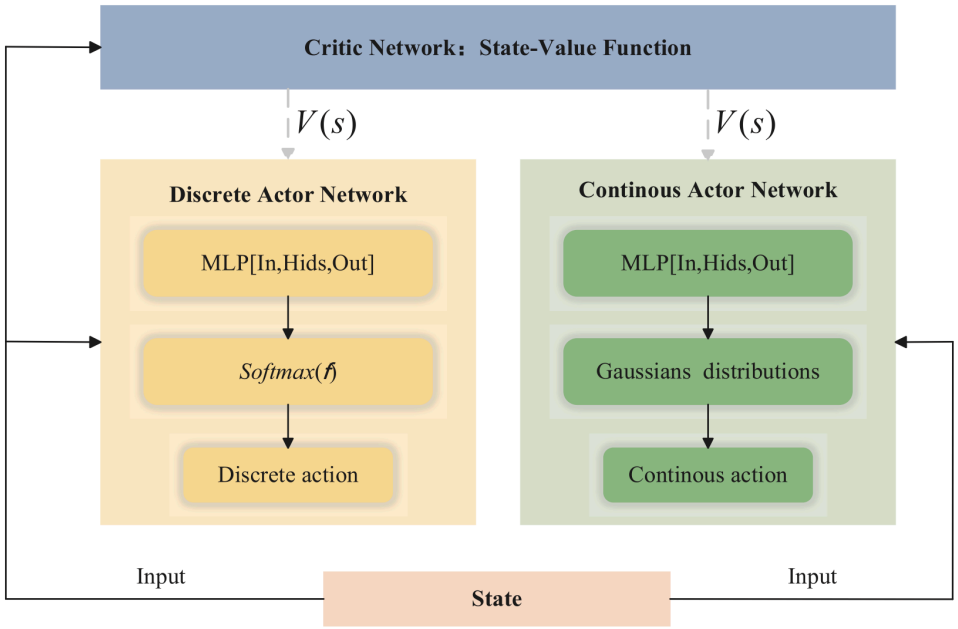


Figure 2.7: Architecture of the proposed Actor-Critic Neural Network. Taken from (Zhao et al., 2026 [26])

Relevance and Comparative Analysis

While the study by Zhao et al [26]. addresses a more geographically complex passenger network, it provides a vital methodological benchmark for this thesis. Both approaches utilize action masking to enforce operational constraints; however, several key distinctions exist. While Zhao et al. [26] optimize for passenger flows using a hybrid HPPO actor-critic model, this thesis focuses on freight logistics (specifically the transport of timber—within a two-line system).

Furthermore, instead of the dual-actor hybrid approach (HPPO) used in the cited study, this research employs a standard PPO algorithm with a single continuous actor. In this thesis, discrete requirements are handled by mapping continuous outputs to discrete values via a specialized mapping strategy, rather than maintaining a separate discrete actor network.

2.3.3 Solving high-speed rolling stock planning with an improved LSTM-PPO reinforcement learning algorithm

The work presented by Su et al. [27] investigates the Rolling Stock Planning (RSP) problem utilizing a PPO framework integrated with Long Short Term Memory (LSTM) network. To prevent the agent from converging on local optima, the authors incorporate a simulated annealing strategy, effectively balancing global exploration with local refinement. The choice of an LSTM architecture is specifically intended to capture temporal dependencies and provide the agent with a recursive memory of the episode's history.

A fundamental assumption of this study is the prioritization of a fixed maintenance threshold over train departures, ensuring safety constraints are satisfied. The operational environment is modeled with three stations and a fixed scheduling regime, establishing a scenario closely aligned with the parameters of this thesis.

State Space Representation

The state at time step t is defined by the history of assignments V_t and the current task x_t. The history V_t represents a comprehensive record of all train services already allocated to the v Rolling Stock Unitss (RSUs), while x_t denotes the specific service currently awaiting assignment:

$$V_t = \{U_t^1, U_t^2, \dots, U_t^v\} = \{\{u_1^1, u_2^1, \dots, u_r^1\}, \dots, \{u_1^v, u_2^v, \dots, u_r^v\}\}$$

Action Space Representation

The agent is tasked with mapping the pending train service (x_t) to an available RSU (v_i). The resulting action is formulated as:

$$a_t = (v_i, x_t)$$

Consistent with the methodology adopted in this research, the action space is constrained by a filtering mechanism that automatically excludes actions violating operational requirements, such as maintenance schedules.

Reward Mechanism

The reward function is a multi-objective scalar derived from three primary components, each modulated by specific hyperparameters:

1. r_t^{connect}: A positive reward for establishing a feasible connection between successive train services on the same RSU .

2. r_t^{save}: An incentive aimed at minimizing the total number of RSUs required for operations.

3. r_t^{penalty}: A penalty term for the violation of hard constraints, primarily regarding fixed maintenance intervals.

Technical Contribution and Relevance to Thesis

The problem addressed by Su et al. [27] serves as a foundational benchmark for this thesis, although it represents a slightly simplified version of the real-world system as it focuses exclusively on RSU efficiency without considering passenger demand.

There is a significant alignment between this thesis and Su et al. [27] regarding the prioritization of system safety; both models enforce maintenance requirements as a higher priority than schedule adherence. However, the maintenance modeling differs in flexibility: while Su et al. [27] restricts maintenance to nighttime windows, this thesis allows the agent to execute early maintenance at any time to accommodate operational needs. Furthermore, while the present work omits dwell times as a deliberate simplification, this feature remains a viable candidate for future integration. Finally, although this thesis utilizes a MLP architecture rather than the LSTM employed by Su et al. [27], the algorithmic structure of their work provided significant inspiration for the development of my proposed solution.

Algorithm 1 LSTM-PPO for rolling stock planning.

Require: Initialized actor π_θ, critic V_ϕ, LSTM state h_0, learning rates $\alpha_\theta, \alpha_\phi$, clipping parameter ϵ, epochs K

Initialize rolling stock planning environment
for each training iteration **do**
  Collect episodes:
  for each circulation planning episode **do**
    Reset environment and initialize LSTM hidden state h_0
    for each train service step t **do**
      Observe current state s_t (e.g. RSU status, depot availability, task x_t)
      Compute action distribution $\pi_\theta(a_t \mid s_t, h_{t-1})$
      Sample assignment decision a_t
      Execute a_t, observe reward r_t and next state s_{t+1}
      Store (s_t, a_t, r_t, h_{t-1}) in buffer
      Update hidden state: $h_t \leftarrow \text{LSTM}(s_t, h_{t-1})$
    end for
  end for
  Estimate advantages $\hat{A}_t$ and returns $\hat{R}_t$ via GAE
  for K epochs **do**
    Sample mini-batches from buffer
    for each mini-batch **do**
      Compute probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t, h_{t-1})}{\pi_{\theta_{\text{old}}}(a_t \mid s_t, h_{t-1})}$$

      Compute clipped surrogate loss:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

      Update actor parameters: $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta L^{\text{PPO}}$
      Update critic loss: $L_V = \|V_\phi(s_t, h_{t-1}) - \hat{R}_t\|^2$
      Update critic parameters: $\phi \leftarrow \phi - \alpha_\phi \nabla_\phi L_V$
    end for
  end for
end for

Figure 2.8: PPO-LSTM Algorithm. Taken from (SU et al., 2026 [27])

2.3.4 Rolling Stock Planning with Integrated Maintenance Constraints: A Rolling Horizon MILP Approach

The work by Bröchin et al. [28] presents a robust optimization framework for rolling stock circulation that explicitly integrates multi-tier maintenance

requirements. Unlike traditional ”maintenance-first” approaches, which often decouple service scheduling from technical service, this study utilizes a Mixed-Integer Linear Programming (MILP) model within a rolling horizon framework to optimize short-term operational efficiency alongside long-term fleet reliability.

Operational Complexity and Horizon Segmentation

The model accounts for high-fidelity operational constraints, including specific maintenance categories (T001, ESA, and DLR) and workshop availability fluctuations due to holidays or capacity limits. A central feature of the methodology is the temporal segmentation of the decision-making window into two distinct phases: the Activity Horizon and the Maintenance Horizon, as illustrated in Figure 2.9.

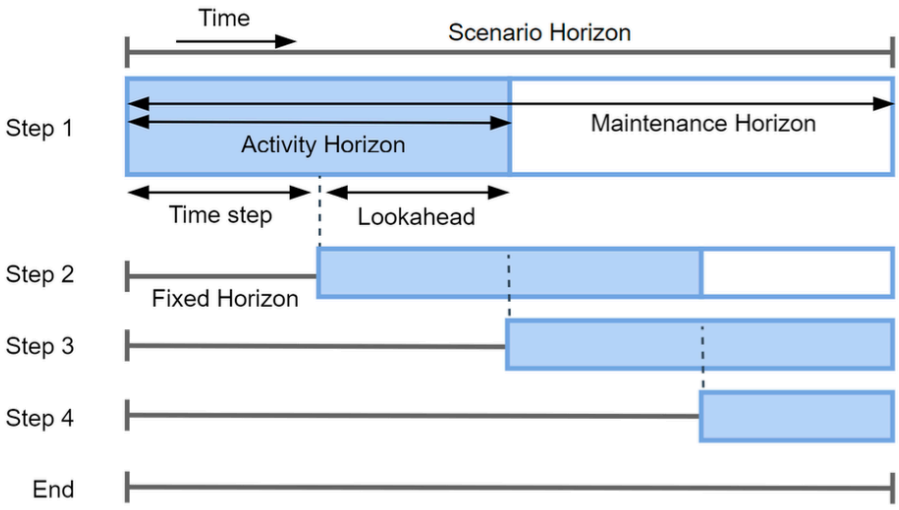


Figure 2.9: Temporal decomposition of the rolling horizon approach, distinguishing between detailed activity scheduling and expected maintenance requirements. Taken from Bröchin et al. (2023) [28].

By solving the MILP model at each step of the rolling horizon, the system performs a trade-off analysis between immediate revenue-generating activities and the expected future costs of maintenance neglect or suboptimal scheduling.

Multi-Layered Graph Architecture

The formulation utilizes a ”layered flow” network architecture, which is critical for maintaining computational tractability. The graph is decomposed into:

- **Activity Flow Layer:** A directed graph $G_A = (V_A, A_A)$ where nodes represent service trips or maintenance slots, and arcs represent feasible transitions (turnarounds, repositioning, and coupling).
- **Maintenance Flow Layers:** Specialized layers for each maintenance type $t \in T$ that track the elapsed time/distance since the last service event, ensuring safety limits are not exceeded.

Objective Function

The optimization goal is to minimize a composite cost function representing both operational and technical expenditures:

$$\min \sum_{a \in A_A} c_a x_a + \sum_{t \in T} \sum_{a \in A_t} d_{a,t} w_{a,t} \tag{2.41}$$

This function penalizes deadheading (empty runs) and minimizes deviations from the original plan, effectively balancing economic performance with schedule stability.

Technical Contribution and Relevance to Thesis

The research by Bröchin et al. [28] is highly relevant as it demonstrates the necessity of integrating maintenance into the primary circulation plan to avoid ”Maintenance Spikes.” However, a significant limitation of their MILP-based approach is its computational latency, which makes it less suitable for the real-time, event-driven rescheduling required in dynamic freight environments. The papers is trying to resolve the following problem,: the assignment and reassignment of wagons to transport freight from origins to destinations within a railway network where there multiple stations and types of wagons. In contrast to their layered network flow, this thesis proposes an Actor-Critic DRL architecture. While Bröchin et al. [28] manage complexity through ”commoditization” of flow, my approach utilizes a Shared-Weight MLP to achieve the same fleet-wide scalability without the need for solving expensive linear programs at every decision point. This allows for near-instantaneous rescheduling when stochastic events (such as the Weibull-distributed failures modeled in my work) occur.

2.3.5 Dynamic wagon assignment and reassignment for pick-up and delivery requests: A deep reinforcement learning framework to address uncertainty in locomotive availability

The research conducted by Habibi et al. (2026) [29] addresses the complexities of railway logistics through the application of DRL, specifically utilizing the PPO algorithm. The primary objective of their framework is the maximization of operational efficiency by ensuring the fulfillment of delivery requests while simultaneously minimizing logistical costs and temporal delays.

State Space Representation

The agent observes a discrete environment where the state s_t at any given time t is defined as:

$$s_t = (Bv^K_{t,m}, ya_{t,w,i})$$

This representation captures two critical components:

- The current inventory levels of freight across all pending requests.
- The distribution and availability of specific wagon types across the network's stations.

This state space design mirrors the requirements of complex railway environments, providing the necessary visibility into resource allocation and demand fulfillment.

Action Space Representation

A recurring challenge in railway scheduling—as observed in the model developed for this thesis—is the lack of scalability when the number of stations or assets increases. To mitigate the "curse of dimensionality" inherent in large-scale discrete action spaces, Habibi et al. introduce a set of twelve heuristic rules 2.10. These rules act as an abstraction layer for the agent, constraining the action space to high-level wagon assignment and re-assignment decisions. This approach contrasts with the raw weight-based selection mechanism utilized in my model, where the agent directly outputs weights for locomotive selection. While my model currently grants the agent greater freedom, the heuristic approach suggests a viable pathway for reducing complexity in future iterations of the project.

Proposed heuristic rules for wagon assignment

Assignment of wagons from the origin stations of requests based on specific assignment conditions and, in stations with the same origin request, selecting the request under selection priority criteria

Symbols	Priority criterion
$RU_{1,1}$	Lower inventory of freight
$RU_{1,2}$	Higher inventory of freight
$RU_{1,3}$	Higher commitment completion
$RU_{1,4}$	Lower commitment completion
$RU_{1,5}$	No assignment in this period

Proposed heuristic rules for wagon reassignment

Reassignment of wagons from each station to the origin of a request, while the station was the destination of that request or is within a specified distance from its origin, and ensuring that the type of wagons being reassigned is suitable for the freight of that request. When multiple requests satisfy these conditions, the request is chosen according to the following priority criteria:

Symbols	Priority criterion
$RU_{2,6}$	Higher inventory of freight at origin.
$RU_{2,7}$	Lower inventory of freight at origin
$RU_{2,8}$	Higher commitment completion
$RU_{2,9}$	Lower commitment completion
$RU_{2,10}$	Lower number of available wagons at origin
$RU_{2,11}$	Higher number of available wagons at its origin is prioritized.
$RU_{2,12}$	No reassignment in this period.

Figure 2.10: Heuristic rules for wagon assignment or re-assignment. Taken from Habibi et al. (2026) [29].

Reward Mechanism

The reward function is formulated to align the agent's behavior with corporate profitability. It is composed of four distinct weighted components:

1. Positive reinforcement for revenue generated from fulfilled freight deliveries.
2. Penalties associated with delivery delays (lateness).
3. Operational costs associated with the transportation of fully loaded wagons.
4. Variable costs incurred by the movement of empty wagons (positioning costs).

Neural Network Architecture

The actor-critic framework employs three fully connected hidden layers, each

consisting of 256 neurons. Rectified Linear Unit (ReLU) functions are utilized for non-linear activation. The optimization process is conducted via the Adam optimizer, utilizing a learning rate of 0.002 and a discount factor (γ) of 0.9. These parameters served as the baseline configuration for my model before empirical fine-tuning was performed to adapt to the specific requirements of locomotive selection.

Technical Contribution and Thesis Relevance

The work by Habibi et al. served as a foundational benchmark for this research. While my model focuses on a multi-locomotive selection environment built with the Gymnasium framework, the architectural parallels in the PPO implementation provided a robust starting point for determining network size and hyperparameters.

Critically, however, the model in [29] assumes a deterministic environment. A significant gap identified during this review is the absence of modeling for stochastic failures or external uncertainty—events that are ubiquitous in real-world railway operations. While my model improves upon the user-interface and visual action-tracking, the next phase of this thesis aims to integrate the heuristic-based dimensionality reduction seen in Habibi's work while introducing environmental uncertainty to better simulate the "real-world" drift between planned schedules and execution.

2.3.6 An actor-critic deep reinforcement learning approach for metro train scheduling with rolling stock circulation under stochastic demand

The study by Ying et al. [30] investigates an actor-critic DRL framework for metro train scheduling, with a specific focus on the rotation of a constrained fleet of rolling stock units. The system is modeled as a closed-loop network of stations $\mathbb{I} = \{1, 2, \ldots, I\}$, where I_0 represents the depot for train dispatching. The objective is to assign a fixed fleet of K trains to N service runs over a predefined horizon, ensuring efficient circulation under capacity constraints.

State Space Representation

The state tuple s_n at each stage n encapsulates the observed environment for the agent:

$$s_n = (\tau_{n,i}, \sigma_{n,i}, l_{n,i}) \tag{2.42}$$

where $\tau_{n,i}$ and $\sigma_{n,i}$ denote the arrival and departure times of service n at station i, respectively. The term $l_{n,i}$ represents the passenger accumulation remaining at station i following the departure of service n. While structurally similar to

the work of Zhao et al. [26], this formulation places greater emphasis on the specific requirements of rolling stock planning.

Action Space Representation

The action vector x_n determined at stage n for the subsequent service $n+1$ is defined as:

$$x_n = (h_{n+1}, u_{n+1,i}, w_{n+1,i}) \tag{2.43}$$

where h_{n+1} is the headway between service n and $n+1$ at the dispatching point, $u_{n+1,i}$ is the section running time between station pairs $(i, i+1)$, and $w_{n+1,i}$ denotes the station dwell times. These parameters are subject to bound-clipping to maintain operational feasibility. A critical assumption in this model is that the fleet size K is smaller than the total service requirements, thereby necessitating an effective circulation and reuse strategy.

Reward Mechanism

The reward function is a weighted multi-objective scalar derived from two perspectives:

1. Passenger Utility: A cost function $c_n^W = \zeta^W \sum_{i=1}^{I} c_{n,i}^W$ quantifies the waiting time for passengers. The factor ζ^W serves to monetize total waiting time based on established economic standards [31, 32].

2. Operator Efficiency: The operational cost c_n^V accounts for energy consumption and maintenance-related expenditures:

$$c_n^V = \sum_{i=1}^{I} \left(\zeta^F F(v_{n,i}) d_{i,i+1} + \zeta^T u_{n,i} + \zeta^D d_{i,i+1} \right) \tag{2.44}$$

   where $F(v_{n,i})$ represents energy consumption as a function of speed, and $\zeta^F, \zeta^T, \zeta^D$ are unit cost factors for fuel, time-related expenditures, and distance-related wear, respectively.

Neural Network Architecture

To mitigate the curse of dimensionality, the authors implement the Deep Deterministic Policy Gradient (DDPG) algorithm. The architecture employs two-layer artificial neural networks for both the actor and the critic, utilizing non-linear activation functions and the Adam optimizer [33]. A linear mapping function f is used to scale the bounded action space $[-1, 1]$ to real-world operational values.

Technical Contribution and Thesis Relevance

This paper serves as a primary reference for the implementation of action-value clipping and linear mapping within this thesis. Furthermore, the

inclusion of multi-faceted maintenance costs in the reward function provided a methodological foundation for our model, despite our simplified parameter set.

Most significantly, Ying et al. highlight the inherent difficulty of achieving convergence in environments characterized by stochasticity and random failure events. This aligns with the findings of the present research: while the system demonstrates high operational performance, the training curves exhibit the characteristic volatility (as seen in Figure 2.11) common in high-uncertainty DRL applications. This research confirms that while reducing stochastic variables leads to smoother convergence, it results in a less robust model capable only of finding trivial solutions.

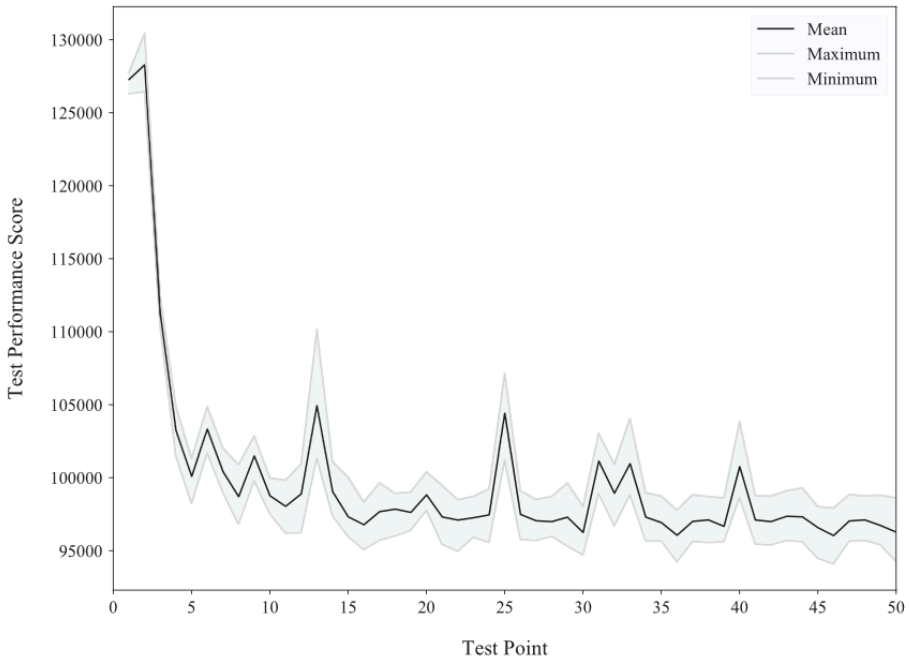


Figure 2.11: Test of the actor-critic agent during the training process.Taken from (Ying et al., 2020 [30])

2.3.7 Optimization of Locomotive Repair Operations via Preventive Maintenance

Liu et al. (2025) [34] investigate strategies to mitigate the substantial maintenance costs associated with the Harmonious High-Power AC Electric Locomotive, the predominant locomotive model utilized for freight transport

in China. The primary objective of their study is the development of a preventive maintenance schedule, optimized via a heuristic algorithm, to minimize overall repair expenditures.

While previous literature, such as the works of Hu et al. [35] and Zhu et al. [36], primarily focused on modeling the failure of isolated components under the assumption of statistical independence, Liu et al. address this limitation. They propose a genetic algorithm specifically designed for the maintenance optimization of multi-component systems, capturing the complex nature of entire locomotives.

Of particular relevance to this thesis is their application of the Weibull distribution to analyze the delay-time concept , defined as the critical interval between the initial manifestation of a defect and the ultimate failure of the system. The Weibull distribution effectively models the interdependencies between component defect probability, operational lifespan, and delay time. The authors note that an elevated probability of a defective component correspondingly diminishes the operating life relative to this delay-time stage.

The component failure rate function is mathematically defined as:

$$\lambda_a(t) = \left(\frac{\delta_a}{\eta_a}\right)\left(\frac{t}{\eta_a}\right)^{\delta_a - 1} \tag{2.45}$$

where δ_a denotes the shape parameter (frequently denoted as β in standard reliability literature) and η_a represents the scale parameter. Furthermore, the authors incorporate an improvement factor, adapting the methodology of Dong et al. [37], to mathematically represent the restoration of component health following preventive maintenance interventions. Building upon this, they extend their framework to encompass complex multi-component interactions, evaluating how systemic dependencies influence optimal maintenance intervals.

Technical Contribution and Thesis Relevance

Empirical results from Liu et al. demonstrate that their optimized preventive maintenance approach yields a 51.04% reduction in total maintenance costs and a 7.89% increase in fleet availability compared to traditional fixed-schedule repair programs. This significant performance margin emphasizes the necessity of transitioning from rigid maintenance schedules to dynamic, strategic methodologies where a decision-making agent possesses the autonomy to optimize interventions. Conclusively, the study formally

identifies maintenance operations as a critical bottleneck in freight transport efficiency.

While Liu et al. provide a comprehensive analysis of complex intra-system relationships, this thesis deliberately adopts a more foundational simulation of hazard failures utilizing the Weibull distribution [38], abstracting away the granular multi-component dependencies modeled in their later sections. This methodological divergence is driven primarily by computational scalability. The heuristic approach proposed in the cited article, while effective for demonstrating preventive maintenance concepts, is computationally intensive and scales poorly. In contrast, the Reinforcement Learning agent developed in this thesis yields a dynamic policy capable of scaling efficiently across large fleets of locomotives.

Finally, although Liu et al. successfully employ an improvement factor [37] to model post-maintenance condition restoration, a similar mechanism was omitted from this research. This limitation arises from the reliance on synthetic data; lacking complete access to historical maintenance schedules and empirical degradation profiles, applying a rigorous improvement factor was not feasible within the scope of this project.

Chapter 3

Methods and Implementation

This chapter details the methodological framework and implementation strategy utilized to address LAP. The initial sections delineate the problem boundaries and the foundational research methodology. Subsequent sections provide a comprehensive exposition of the model's implementation. A rigorous examination is dedicated to the core architecture (Model V1), followed by a concise discussion regarding the integration of stochasticity and proactive maintenance in the advanced iterations (Models V2 and V3).

3.1 Overview of the Research Methodology

The research commenced with a comprehensive literature review focusing on deep reinforcement learning paradigms and state-of-the-art training algorithms. Following this foundational phase, the operational constraints and specific parameters of the baseline RISE simulation environment were systematically analyzed. This preliminary synthesis of theoretical frameworks and domain-specific logistics informed the subsequent architectural design of the autonomous agent.

The development lifecycle followed a structured, milestone-driven methodology formulated in collaboration with industrial and academic supervisors. The methodological framework was developed through a phased, incremental escalation of environmental complexity. The initial technical contribution involved transitioning the existing baseline framework from a step-based simulation to a highly optimized, event-driven architecture. This infrastructural upgrade established the necessary computational foundation for the sequential development of Models V1, V2, and V3.

An empirical, iterative approach was adopted for architectural selection

and hyperparameter optimization. The formulation of the core agent (Model V1) required the most extensive research effort, necessitating the transition from a naive heuristic baseline to a sophisticated, strategy-driven policy capable of evaluating complex fleet dynamics. To navigate architectural bottlenecks, a rigorous experimental loop was established for every developmental phase. This loop consisted of isolating the specific operational deficit, hypothesizing three to four candidate algorithms or architectural modifications, and systematically implementing each proposed strategy. The efficacy of every candidate was subsequently quantified against baseline metrics or prior model iterations to ensure monotonic performance improvements.

When novel empirical challenges arose that could not be resolved through immediate iterative testing, solutions were derived through a synthesis of established literature and expert consultation with specialized researchers at RISE. By abstracting the highly specific architectures presented in recent operations research and reinforcement learning literature, it was possible to extract generalizable principles and adapt them to the unique constraints of the locomotive assignment domain.

3.2 Hardware and development setup

The project development was conducted using Python 3.13 with the following core dependencies: PyTorch 2.10.0, TorchRL 0.11.1, Gymnasium 1.2.3, Stable-Baselines3 2.7.1, and NumPy 2.4.4. PyTorch was selected as the primary deep learning framework due to its dynamic computational graph and flexibility in implementing custom reinforcement learning algorithms. TorchRL 0.11.1 was chosen to provide optimized tensor-based operations for policy gradient methods, while Gymnasium 1.2.3 was adopted as the standard environment interface for reinforcement learning tasks, replacing the legacy OpenAI Gym.

For the core training pipeline, we leveraged PyTorch-Optimizer 3.9.0 to implement advanced optimization techniques beyond standard SGD, essential for stable convergence of policy gradient methods. Stable-Baselines3 2.7.1 was integrated for baseline algorithms (e.g., greedy heuristics) to provide comparative performance metrics. Model training progress and metrics were monitored using TensorBoard 2.20.0, which provided real-time visualization of reward curves, episode statistics, and convergence behavior. Scientific computing tasks, including environment dynamics simulation, were handled by SciPy 1.17.0.

Visualization and result analysis were performed using Matplotlib 3.10.8, enabling the generation of Gantt diagrams for schedule visualization and performance comparison plots. Development tools included Pytest 9.0.2 for unit testing and Mypy 1.19.1 for static type checking, ensuring code robustness and maintainability. The complete environment stack was frozen and documented in `requirements.txt` to ensure reproducibility across different computational resources.

3.3 Sequential Decision-Making Formulation

The locomotive assignment problem considered in this thesis is formulated as a sequential decision-making problem. At each decision point t, the agent observes the current state of the railway system, denoted by s_t, and selects an action a_t representing locomotive dispatching or maintenance decisions. After the action is executed, the environment transitions to a new state s_{t+1} and provides a reward r_t that reflects the quality of the decision.

This formulation is suitable because each assignment decision influences the future availability of locomotives, the accumulated delay of scheduled trains, and the maintenance condition of the fleet. Therefore, the objective is not only to make the best immediate assignment, but to learn a policy that produces good long-term operational outcomes.

Formally, the problem is modeled as a Markov Decision Process defined by the tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle,$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition function, $\mathcal{R}$ is the reward function, and γ is the discount factor. The agent learns a policy $\pi(a_t \mid s_t)$ that maps observed states to actions with the goal of maximizing the expected cumulative reward:

$$\mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_t \right].$$

In this context, the state contains information about locomotive availability, train schedules, station positions, locomotive health, and maintenance status. The action represents the selection of locomotives for dispatch and, in the final model, the selection of locomotives for proactive maintenance. The reward penalizes operational lateness, missed departures, inefficient assignments, and undesirable maintenance behavior, while encouraging

feasible and reliable plans.

3.4 The Simulation Environment

The simulated logistical network is formally defined by a set of stations $\mathcal{S} = \{A, B\}$, a homogeneous fleet of ten locomotives $\mathcal{L} = \{l_0, l_1, \ldots, l_9\}$, and a daily set of six scheduled train services $\mathcal{T} = \{t_0, t_1, \ldots, t_5\}$. The baseline schedule is static for a 24-hour period and is subsequently iterated to encompass a continuous five-day operational horizon, as detailed in Table 3.1. The fundamental temporal resolution of the simulation is defined in hours.

Table 3.1: Daily operational schedule and turnaround parameters for the Station A to Station B corridor. Turnaround buffers are expressed in hours.

Service ID	Origin	Destination	Departure	Turnaround (h)
TRN-00	Station A	Station B	02:00	3
TRN-01	Station B	Station A	05:00	4
TRN-02	Station A	Station B	08:00	4
TRN-03	Station B	Station A	12:00	4
TRN-04	Station A	Station B	16:00	4
TRN-05	Station B	Station A	20:00	6

Note: The schedule operates on a 24-hour cycle.

3.4.1 Entity Definitions and System Attributes

To accurately simulate the logistical network, the physical entities are modeled computationally with specific operational attributes. Each locomotive $l \in \mathcal{L}$ is defined internally by the following parameters:

- **Identifier:** A unique integer identifier indexing the locomotive.
- **Operational Log:** The current and historical operational activity records, which dictate the asset's availability status.
- **Maintenance Distance (d_m):** The cumulative distance traveled since the completion of its last maintenance cycle.
- **Spatial Availability:** The projected station node where the locomotive will be stationed upon completion of its current active task.

- **Health (h):** A continuous internal health index where $h \in [0, 1]$. A value of $h \to 1$ indicates a recently maintained asset, whereas $h \to 0$ reflects a critical requirement for physical intervention.

Similarly, each train service $t \in \mathcal{T}$ encapsulates strict temporal and compositional software attributes: the scheduled departure time (τ_{sched}), the actual departure time (τ_{act}), the fixed transit duration, the assigned motive power (a subset of $\mathcal{L}$ successfully allocated to the train), and the projected arrival time.

3.4.2 Maintenance Dynamics and Episodic Initialization

Maintenance operations are characterized as discrete, location-bound events occurring at the stations. A maintenance instance is strictly defined by its geographical location, start time, end time, and the specific locomotive undergoing service. Irrespective of the trigger, the temporal duration of any maintenance cycle is modeled as a fixed constant of $\Delta t_m = 5$ hours.

The software environment simulates three distinct paradigms of maintenance interventions, sequentially introduced across the model iterations:

1. **Stochastic Failures:** Randomly induced mechanical breakdowns occurring intra-transit (introduced in Model v2).

2. **Deterministic Forced Maintenance:** A hard operational constraint triggered automatically when a locomotive's cumulative distance reaches the threshold of 12,000 km.

3. **Proactive Maintenance:** Endogenous maintenance scheduling executed autonomously by the system prior to mandatory failure thresholds (introduced in Model v3).

To ensure robust variability across simulated scenarios, the initial configuration of each episode is highly stochastic. Specifically, at the start of a simulation run, one randomly selected locomotive is forced into immediate maintenance, and the d_m attributes for the remaining fleet are initialized uniformly at random.

3.4.3 Event-Based Simulation Architecture

A critical architectural enhancement in this thesis is the transition from a discrete, uniform time-step simulation to an asynchronous, event-driven

architecture. In the initial legacy framework, the internal clock progressed in fixed hourly increments. This approach generated a surplus of ”dead time”—temporal steps where no operational triggers occurred (e.g., all trains were merely in transit). Evaluating an autonomous agent at these inert steps is highly inefficient, as it forces inference without meaningful environmental changes, thereby polluting the learning algorithm's replay buffer with uninformative noise.

By transitioning to an event-based horizon, the simulation advances directly to operational inflection points. Every state transition directly corresponds to a tangible systemic trigger (e.g., a train is ready for dispatch, or an asset requires immediate assignment).

Operationally, the simulation manages a priority queue (Q) of future events. Events are sorted primarily by their chronological execution time, and secondarily by a predefined operational priority constraint. The core events are:

- `TrainDestinationReachedEvent` (Priority 10)
- `MaintenanceEndEvent` (Priority 10)
- `TrainScheduledEvent` (Priority 99)

Upon the initiation of a train transit or a maintenance cycle, the corresponding completion event is dynamically injected into Q. The simulation clock subsequently fast-forwards directly to the temporal coordinate of the next popped event in Q, thereby rendering the environment computationally efficient and fully reactive to pertinent logistical triggers.

3.5 Training Details

To improve the efficiency of sample collection, this framework eschews sequential execution in favor of a vectorized environment architecture. Utilizing TorchRL's optimized tensor-based operations, the system instantiates multiple independent environments across parallel computing workers. Each worker maintains an isolated copy of the environment's state, observation space, action space, and reward function.

During the rollout phase, a `SyncDataCollector` synchronizes these parallel experiences. Rather than yielding after a single global step, the collector advances all environments for a predetermined horizon of $n_{steps} = 256$. Given $n_{envs} = 12$ parallel workers, each data collection phase yields

a unified batch tensor of $12 \cdot 256 = 3072$ transitions prior to executing a PPO update. Furthermore, the environment implements automatic episodic truncation; when a worker reaches a terminal state, it autonomously resets and initializes a new episode with a distinct random seed while parallel workers continue their rollouts. This asynchronous reset mechanism ensures that computational throughput is maximized, preventing the training pipeline from stalling on disparate episode lengths.

3.5.1 Optimization Strategy

The neural network parameters are optimized utilizing the AdamW algorithm coupled with a linear learning rate decay scheduler. Initial model configurations utilized the Ranger21 optimizer to exploit its underlying LookAhead mechanism. LookAhead maintains two sets of parameters: ”fast weights” that update at standard intervals, and ”slow weights” that periodically interpolate toward the fast weights. However, empirical observations revealed that for this specific PPO formulation, the LookAhead mechanism caused the network weights to misalign with the current mini-batch gradients. By transitioning to AdamW with linear decay, the learning rate monotonically decreases as the total number of episodes increases. This enables the agent to take highly precise gradient steps during the later stages of training, ensuring stable convergence to local optima rather than overshooting the optimal parameter space.

3.5.2 The PPO Update Loop

Upon the collection of the 3072 transitions, the data is partitioned into mini-batches of size 1024, resulting in three distinct mini-batches per update cycle. This mini-batching strategy significantly reduces the GPU memory footprint, processing only a third of the rollout buffer at any given time. Following standard PPO literature, which demonstrates that multiple optimization passes over the same rollout data enhance sample efficiency and final policy performance [16], the agent iterates over these mini-batches for six complete epochs.

Within each epoch, for every mini-batch, the objective function synthesizes three distinct loss components, each governed by specific hyperparameters tracked during the evaluation phase:

- **Policy Surrogate Loss (`loss_objective`):** The primary PPO clipped surrogate objective. This component relies on the hyperpa-

rameter `clip_range` (ϵ), which restricts the probability ratio of the new policy to the old policy within the interval $[1-\epsilon, 1+\epsilon]$. This clipping mechanism is critical; it fundamentally limits how much the policy can change per update step, thereby preventing catastrophic policy divergence and ensuring monotonic training stability.

- **Value Function Loss (`loss_critic`):** The Mean Squared Error (MSE) between the critic's predicted state-values and the computed empirical returns (advantages), driving the accuracy of the baseline estimation utilized by the actor.

- **Entropy Regularization (`loss_entropy`):** An entropy bonus that penalizes premature convergence. This is scaled by the `entropy_coeff` parameter (c_{ent}), which controls the quantitative weight of the regularization term. A higher entropy coefficient forces broader stochastic exploration of the state-action space, whereas setting it too low risks premature convergence to suboptimal, highly deterministic local minima.

Once the aggregate scalar loss is formulated from these three components, the total gradient is computed with respect to the network parameters. To counteract the exploding gradient problem frequently observed in deep RL architectures, an additional regularization step is applied during backpropagation: the computed gradients are subject to L_2 norm clipping at a strict threshold of 0.5 prior to the optimizer step. The formal sequence of this update logic is detailed in Algorithm 1.

Algorithm 1 PPO Mini-Batch Update Loop

for $epoch = 1, 2, \ldots, 6$ **do**
  for each $mini_batch \in \mathcal{B}$ {$\mathcal{B}$ contains 3 batches of size 1024} **do**
    $\mathcal{L}_{total} \leftarrow \text{ComputePPOLoss}(mini_batch, \epsilon, c_{ent})$
    $optimizer$.zero_grad()
    $\mathcal{L}_{total}$.backward()
    ClipGradientNorm(θ, 0.5)
    $optimizer$.step()
  end for
end for

3.5.3 Evaluation and Logging Protocol

To systematically monitor the agent's convergence and ensure numerical stability throughout the optimization process, rigorous intermediate and final evaluation protocols have been established. Comprehensive training telemetry is continuously recorded via TensorBoard. To facilitate granular analysis, these tracked metrics are fundamentally categorized into three operational domains, each explicitly defined in its respective table: episodic metrics (Table 3.2), PPO algorithmic optimization parameters (Table 3.3), and deterministic comparative benchmarks (Table 3.4).

The comparative routing decisions are also rendered via Gantt charts, facilitating a visual and operational analysis of the policy's structural evolution.

Table 3.2: Episodic Metrics (Aggregated per Rollout)

Metric Name	**Definition**	**Operational Purpose**
`Impact Percentage`	A distributional breakdown isolating the proportional impact of each discrete reward component on the episodic sum.	Facilitates precise reward shaping by identifying dominating or suppressed reward signals, critical for tuning penalty coefficients.
`Lateness`	The absolute mean temporal delay accumulated by all scheduled services within a single training episode.	Serves as the primary raw performance indicator to verify the minimization of logistical delays.
`Success Rate`	The fractional ratio of train services dispatched exactly at their scheduled departure time.	Tracks strict schedule adherence and punctuality optimization.
`Reward`	The cumulative scalar reward aggregated across the entire operational episode.	Acts as the fundamental objective metric to monitor overall monotonic policy improvement.

Following the completion of the training phase, a definitive evaluation is executed over 500 stochastic episodes utilizing isolated random seeds. A comprehensive suite of comparative metrics is computed against the baseline, and the final statistical distributions are systematically archived to facilitate hyperparameter tuning, debugging, and robust empirical validation of the final architecture.

Table 3.3: Proximal Policy Optimization (PPO) Parameters

Metric Name	Definition	Operational Purpose
`clip_fraction`	The empirical ratio of policy update samples that exceeded the surrogate objective clipping threshold (ϵ).	Diagnoses the integrity of the trust-region boundaries, ensuring the policy does not execute destructively large updates.
`entropy_loss`	The Shannon entropy of the continuous action distribution output by the actor network.	Quantifies the exploration-exploitation trade-off; expected to progressively decay as the policy converges to a deterministic strategy.
`approx_KL`	An approximation of the Kullback-Leibler (KL) divergence between the pre-update and post-update policies.	Monitors the magnitude of distributional shift during optimization to guarantee stability within the defined trust region.
`policy_loss`	The minimized scalar value of the clipped surrogate objective function.	Tracks the optimization convergence of the actor network.
`value_loss`	The Mean Squared Error (MSE) associated with the state-value predictions.	Tracks the convergence trajectory and baseline accuracy of the critic network.

Table 3.4: Comparative Benchmarks (Evaluated at 10-Episode Intervals)

Metric Name	Definition	Operational Purpose
`Delta Lateness`	A strict, deterministic comparative evaluation of lateness differentials between the agent and the baseline.	Provides a noise-free, standardized benchmarking metric to empirically validate superiority over the heuristic (target value < 0).
`Weight_Overview`	A time-series visualization representing the mean continuous preference outputs generated for specific locomotives.	Visualizes the agent's emergent operational strategies, isolating distinct combinatorial patterns in locomotive allocation.

3.6 Baseline Formulation

In Deep Reinforcement Learning (DRL) literature, evaluating an autonomous agent necessitates a comparative analysis against a reference policy, typically a standard operational heuristic, to establish an empirical performance lower bound. Throughout this thesis, the primary benchmark is defined as a *Stochastic Eager Dispatch Heuristic*. This baseline is particularly critical for evaluating the policy convergence and relative performance metrics of Model v1 and Model v2.

For Model v3, the comparative dynamic shifts. As the environmental complexity scales to include proactive asset management, the static rule-set of the heuristic inherently degrades. Consequently, in the context of Model v3, the baseline is utilized not merely as a competitive benchmark, but as a mechanism to quantify the performance degradation of static heuristics when subjected to complex, stochastic operational constraints.

The heuristic baseline is designed to aggressively clear pending schedules without foresight or state-value estimations. Its deterministic logic coupled with stochastic asset selection is formalized in Algorithm 2.

Algorithm 2 Stochastic Eager Dispatch Heuristic

Require: $\mathcal{T}$: Set of pending train services
Require: $\mathcal{L}$: Global set of available locomotives
Require: $\mathcal{L}_s \subseteq \mathcal{L}$: Subset of available locomotives currently located at station s

$\mathcal{A} \leftarrow \emptyset$ {Initialize action space}
$\mathcal{T}_{ready} \leftarrow \{t \in \mathcal{T} \mid \text{time} \geq t_{sched} \text{ and } t_{status} = \text{pending}\}$
$\mathcal{T}_{valid} \leftarrow \{t \in \mathcal{T}_{ready} \mid |\mathcal{L}_{t_{origin}}| \geq 2\}$ {$|\mathcal{L}_{t_{origin}}|$ denotes fleet size at t's origin}
if $\mathcal{T}_{valid} = \emptyset$ **then**
    return NO-OP
end if
$\mathcal{T}_{selected} \leftarrow$ Select up to 2 trains from $\mathcal{T}_{valid}$ with distinct destinations
for each $t \in \mathcal{T}_{selected}$ **do**
    $l_1, l_2 \leftarrow$ Sample without replacement 2 locomotives from $\mathcal{L}_{t_{origin}}$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{\text{Assign } (l_1, l_2) \text{ to } t\}$
end for
return $\mathcal{A}$

The fundamental limitations of this baseline are twofold. First, it is

strictly constrained to a fixed assignment parameter of exactly two locomotives per train, rendering it incapable of dynamically scaling motive power in response to payload requirements or directional imbalances. Second, the heuristic lacks the temporal foresight required to execute proactive maintenance, leaving it highly vulnerable to catastrophic intra-transit failures. The DRL architectures developed in Models v1 through v3 are explicitly designed to exploit these heuristic rigidities, leveraging superior state-space representation to dynamically allocate assets and proactively schedule maintenance interventions.

3.7 Observation and Action Space Formulation (Model v1)

The fundamental bridge between the simulated environment and the Deep Reinforcement Learning (DRL) agent is the mathematical formulation of the observation space ($\mathcal{O}$) and the action space ($\mathcal{A}$). The following subsections detail the state representations processed by the actor network and the continuous action vectors it outputs.

3.7.1 Observation Space

The observation space dictates the limits of the agent's environmental perception. To ensure stable gradient updates during the neural network's optimization, the entire observation vector is continuously normalized to the range $[-1, 1]$.

It is intrinsically modular, composed of global infrastructural metrics and specific asset constraints. A critical component of this space is the *Locomotive Mask*, a binary vector utilized to enforce constraint-aware action selection, effectively masking out non-feasible decisions (e.g., assigning a locomotive that is currently undergoing maintenance or actively in transit). The specific attributes of the observation space are synthesized in Table 3.5.

3.7.2 Action Space

While locomotive assignment is fundamentally a discrete combinatorial problem, directly formulating a discrete action space results in exponential state-action explosion as the fleet size scales. Furthermore, implementing a hybrid discrete-continuous distribution inherently destabilizes Proximal

Table 3.5: Observation Space Attributes and Dimensions for Model v1

Feature Group	**Description**	**Size**	**Range**
Global Features	Demand urgency, "ready" flags for the next 3 scheduled trains (Station A and B), temporal decay auxiliary variables, and the count of available locomotives per station.	15	$[-1, 1]$
Locomotive Mask	Binary validity matrix. A value of 1.0 indicates station availability; 0.0 indicates the asset is in transit or maintenance.	10	$\{0, 1\}$
Locomotive Status	Ten dimensional pairs of [health, station_id], encapsulating the localized physical state of the entire fleet.	20	$[-1, 1]$

Policy Optimization (PPO). Consequently, this thesis proposes a continuous relaxation of the action space. The agent outputs a continuous preference vector, which the environment subsequently discretizes into operational commands.

The action vector comprises preference scores for each locomotive and a strategic dispatch selector for each station, detailed in Table 3.6.

Table 3.6: Continuous Action Space Output and Downstream Mapping

Action Component	**Description**	**Size**	**Output Range**
Locomotive Preferences	Continuous preference score assigned to each locomotive.	10	$[-1, 1]$
Station Selectors	Continuous value dictating the volume of locomotives to dispatch from Station A and Station B.	2	$[-1, 1]$

The Station Selector output, denoted as $v \in [-1, 1]$, is deterministically mapped to the discrete number of locomotives to dispatch, $N_{dispatch}$, via the

following piecewise function:

$$N_{dispatch}(v) = \begin{cases} 0 & \text{if } v < -0.8 \quad \text{(Strategic waiting)} \\ 2 & \text{if } -0.8 \leq v < 0.2 \\ 3 & \text{if } v \geq 0.2 \end{cases} \tag{3.1}$$

During the environment step function, the simulation evaluates eligible, unmasked locomotives at a given origin station, identifies the most urgent pending train, ranks the available locomotives by their continuous preference scores, and selects the top-$N_{dispatch}$ assets to fulfill the composition requirements.

3.8 Neural Network Architecture (Model v1)

The weights of the actor neural network explicitly represent the parameterized policy, $\pi_\theta(a|s)$. To minimize computational complexity while retaining sufficient representational capacity, we utilize feed-forward Multi-Layer Perceptrons (MLP) rather than recurrent topologies.

Initial baseline testing utilized a standard Global MLP that ingested the entire observation space as a flattened vector. However, this architecture proved fundamentally flawed: the network overfitted to positional encodings. Because the fleet was passed as a fixed-length array, the network would hyper-fixate on specific array indices rather than learning the generalized operational attributes of the locomotives.

To rectify this and enforce **permutation invariance**, the actor is bifurcated into two specialized sub-networks. This structure, visualized in Figure 3.1, forces the agent to evaluate assets based strictly on their physical and temporal state.

The specific topological configurations of the bifurcated actor are strictly defined as follows:

- **Selector MLP:** Formulated to dictate macro-level dispatch strategy based on the global features which is a subset of the observation space ($\mathbb{R}^{15}$). The network consists of two fully connected (FC) hidden layers, each comprising 128 neurons. Non-linear transformations are applied via LeakyReLU activation functions with a negative slope of $\alpha = 0.03$.
- **Shared-Weight MLP:** Formulated to evaluate individual locomotive fitness. The network ingests a concatenated vector ($\mathbb{R}^{15}$) combining

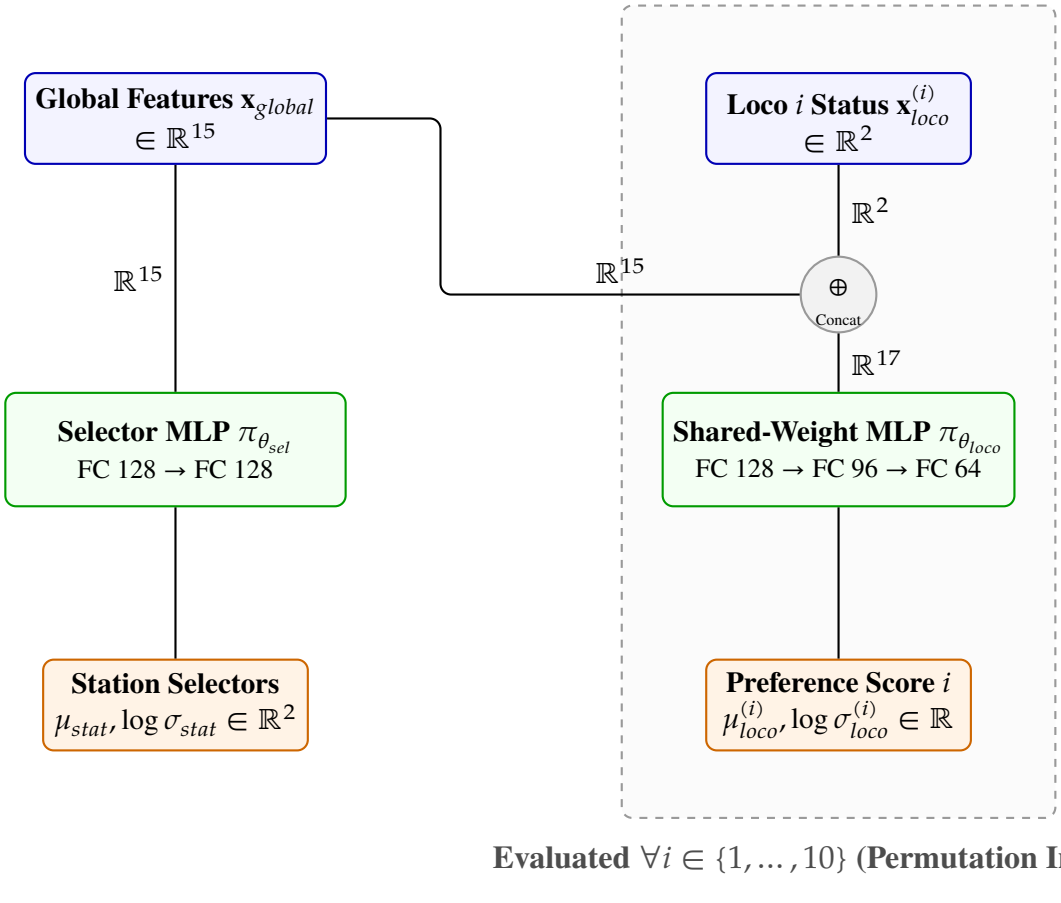


Figure 3.1: Architectural schematic of the bifurcated continuous Actor policy. The left pathway deduces macro-level dispatch strategies, while the right pathway establishes strict permutation invariance by independently evaluating each locomotive i through shared weights.

the global features and a single locomotive's features. It operates over three hidden layers with dimensions $[128, 96, 64]$, utilizing identical LeakyReLU ($\alpha = 0.03$) activations. This sub-network executes a forward pass ten independent times per environmental step—once for each asset—effectively evaluating the fleet using a strictly shared parameter space.

3.8.1 Stochastic Continuous Policy and Gaussian Sampling

Because PPO is an on-policy algorithm, the neural network does not output a deterministic action directly. Instead, to balance exploitation of known high-reward strategies with the exploration of the observation space, the actor outputs the parameters of a multivariate Gaussian (Normal) probability distribution, denoted as $\mathcal{N}(\mu, \sigma^2)$.

For each required action dimension (e.g., the preference score for locomotive i), the final hidden layers of both the Selector and Shared MLPs bifurcate into two distinct linear layers. The first linear layer outputs the *mean* (μ), which represents the network's deterministic intent or highest-confidence action. The second linear layer outputs the *log-standard deviation* ($\log \sigma$). The standard deviation dictates the variance or "exploration radius" around the mean. The log-scale is utilized exclusively for numerical stability during gradient descent, ensuring that the subsequently exponentiated standard deviation $\sigma = \exp(\log \sigma)$ remains strictly positive.

During the training phase, the actual continuous action a sent to the environment is sampled stochastically from this parameterized distribution:

$$a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)^2) \tag{3.2}$$

This stochastic sampling is critical; early in training, the network outputs a large σ, resulting in broad exploration of the action space. As training converges and the policy discovers optimal strategies, the network naturally decays σ, centralizing the sampling around the highly-optimized mean μ. During post-training evaluation and final deployment, the stochastic sampling is typically bypassed, and the agent executes the deterministic mean μ to ensure optimal operational consistency.

3.9 Reward Function Engineering (Model v1)

In Deep Reinforcement Learning (DRL), the reward function ($\mathcal{R}$) serves as the mathematical proxy for the overarching operational objectives. It is the sole mechanism through which the agent interprets the desirability of its actions. For this thesis, the reward formulation must strictly encapsulate GreenCargo's multi-objective logistical targets: ensuring absolute schedule completion, minimizing systemic departure lateness, maximizing on-time dispatch rates, and preventing the degradation of individual assets through equitable workload distribution.

Constructing this reward manifold requires meticulous hyperparameter scaling. A fundamental challenge in constraint-heavy DRL environments is the risk of policy collapse. If penalization terms are disproportionately severe, the agent may converge to a degenerate, risk-averse policy—frequently observed as a "no-op" (no-operation) strategy—where it deliberately avoids dispatching trains to prevent incurring massive latency or failure penalties.

To mitigate this, the architecture utilizes reward shaping, combining dense (step-wise and event-driven) signals to guide immediate decision-making, with sparse (terminal) signals to evaluate global episodic success.

3.9.1 Dense and Event-Driven Formulations

Dense and event-driven rewards are injected into the environment synchronously with agent actions or temporal steps. The specific components are detailed in Table 3.7.

3.9.2 Sparse and Terminal Formulations

Sparse rewards are strictly computed and applied at the termination of an episode. They are designed to evaluate the holistic quality of the agent's scheduling strategy, capturing long-term dependencies that cannot be accurately assessed during mid-rollout states. These components are detailed in Table 3.8.

3.9.3 Implementation Architecture and Hyperparameter Sensitivity

From a software engineering paradigm, the reward computation is strictly decoupled from the core environment logic and encapsulated within a

Table 3.7: Dense and Event-Driven Reward Components

Component Name	Type	Mathematical Logic & Description
START_BONUS	Positive	A fixed scalar reward applied immediately upon the successful dispatch of a train, incentivizing the agent to execute the schedule.
ON_TIME_BONUS	Positive	An auxiliary fixed scalar applied if the actual departure time (τ_{act}) is less than or equal to the scheduled departure time (τ_{sched}).
LATENESS_WEIGHT	Negative	A continuous, step-wise penalty proportional to the delta lateness (ΔL) accumulated over the current environmental timestep.
LATENESS_CONTINUOUS	Negative	An event-driven penalty applied upon dispatch, scaled proportionally ($-1 \times 10^{-3} \cdot L$) to the absolute accumulated lateness (L) of the specific train.

Table 3.8: Sparse and Terminal Reward Components

Component Name	Type	Mathematical Logic & Description
MISSED_PENALTY	Negative	A massive scalar penalty applied iteratively for every scheduled train that remains undispatched at the episode's conclusion.
VARIANCE_WEIGHT_P	Negative	Penalizes asymmetrical fleet utilization. Computed as $\lambda_{var} \frac{\mathrm{Var}(D)}{\mu_D + \epsilon}$, where D represents the set of total distances traveled by each locomotive. The variance is normalized with respect to the mean and then rescaled properly (ϵ is used to avoid 0 case)

dedicated, stateful object-oriented class. This centralization ensures that the complex reward state is systematically tracked. More importantly, it facilitates component-wise logging; each individual penalty and bonus is tracked independently and exported to TensorBoard, allowing for granular analysis of the agent's reward manifold during the optimization phase.

The precise scalar weights assigned to these components were derived through extensive empirical hyperparameter tuning, seeking a delicate equilibrium between task execution and constraint adherence. The most volatile parameter during training proved to be the `MISSED_PENALTY`. Empirical observations indicated a high sensitivity threshold: if the penalty magnitude was set too high, the agent exhibited catastrophic policy collapse, taking no actions to avoid the risk of downstream cascading delays. Conversely, if the magnitude was too low, the agent prioritized immediate, easy bonuses while readily sacrificing the completion of complex or late-stage scheduled services. The quantitative impact of these tuning iterations and their resulting behavioral policies are rigorously analyzed in the Results chapter.

3.10 Model v2

The secondary configuration of the agent framework, Model v2, maintains the identical action space, observation space, reward manifold, and neural network architecture defined in Model v1. The core contribution of this configuration is the introduction of environmental stochasticity via random locomotive failures.

To maintain the boundary constraints of the currentMDP, these stochastic breakdowns are restricted to inter-transit idle phases at terminal stations. Intra-transit failures are explicitly abstracted away; empirically, en-route locomotive faults frequently manifest as partial tractive degradation (e.g., speed restrictions) rather than catastrophic immobility, requiring complex spatial-temporal routing alterations that fall outside the scope of this specific model configuration. Furthermore, while the agent must now dynamically detect and reschedule around unexpected asset unavailability, it is not yet imbued with proactive maintenance capabilities; scheduled maintenance remains governed by a hard-coded cumulative distance threshold.

3.10.1 The Continuous Hazard Model

Initial prototype implementations of Model v2 utilized a priori sampling, where Y failure events were strictly pre-scheduled at the initiation of the

episode. This approach decoupled the temporal occurrence of a failure from the physical state of the fleet, resulting in boundary anomalies where a locomotive with 100% health could arbitrarily fail simply to satisfy the pre-sampled quota.

To rectify this, the architecture was fundamentally overhauled to utilize a *Continuous Hazard Model*. Breakdowns transitioned from pre-scheduled temporal events to real-time, probabilistic state transitions governed by asset utilization. The framework executes this stochastic degradation through four formal computational phases:

1. **Operational and Temporal Validity Constraints:** Prior to probability computation, the system enforces a strict validity filter. Stochastic failures are constrained within an operational "Danger Window" (excluding the extreme margins of the episodic schedule to prevent statistically anomalous initial or terminal failures). Furthermore, the specific asset must be actively idle at a station terminal, avoiding logical conflicts with active transit states.

2. **Baseline Hazard Formulation:** If a locomotive satisfies the operational constraints, the system evaluates its cumulative wear ratio, defined as the distance traveled since its last maintenance cycle relative to the maximum allowed threshold ($d_{max} = 12,000$ km). Utilizing a Weibull distribution, the system translates this wear ratio into an instantaneous hourly hazard rate, denoted as λ. Consequently, a recently serviced asset yields a near-zero hazard rate, while an asset approaching its maintenance limit exhibits an exponentially increasing probability of failure.

3. **Asynchronous Temporal Scaling:** Because the simulation utilizes an event-driven architecture that skips inert temporal blocks, the instantaneous hazard rate must be scaled to the discrete time elapsed between events (Δt). The probability of an asset failing during this elapsed idle period is calculated utilizing the standard exponential survival function:

$$P_{fail}(\Delta t) = 1 - \exp(-\lambda \cdot \Delta t) \tag{3.3}$$

   This continuous scaling ensures mathematical consistency; idling for five minutes yields a negligible failure probability, whereas idling for five hours exponentially compounds the cumulative risk.

4. **Stochastic State Transition:** Finally, the environment executes a Bernoulli trial utilizing the computed $P_{fail}(\Delta t)$ parameter. If the stochastic trial results in a failure, the locomotive's state is immediately transitioned to forced maintenance, rendering it unavailable, and the global quota for permitted random episodic failures is decremented.

3.10.2 Weibull Distribution and Reliability Engineering

To accurately model mechanical degradation, the baseline hazard rate λ is derived from the Weibull distribution. In classical reliability engineering, the Weibull hazard function $h(t)$, representing the instantaneous failure rate at time t, is defined as:

$$h(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} \tag{3.4}$$

where $\eta > 0$ represents the scale parameter and $\beta > 0$ represents the shape parameter. The selection of the shape parameter β is critical, as it dictates the degradation modality of the asset, mapping directly to the distinct phases of the theoretical "Bathtub Curve" of reliability [39]:

- $\beta < 1$ **(Infant Mortality):** The hazard rate monotonically decreases over time. This regime models early-life manufacturing defects or improper maintenance servicing, where an asset is most likely to fail immediately upon deployment.
- $\beta = 1$ **(Memoryless / Random Failure):** The hazard rate is entirely constant ($h(t) = 1/\eta$), rendering the failure probability independent of the asset's age. This models random, external operational shocks. Implementing $\beta = 1$ in this framework would critically destroy the DRL agent's incentive to optimize maintenance routing, as heavily utilized assets would possess the exact same failure probability as newly serviced assets.
- $\beta > 1$ **(Wear-Out Phase):** The hazard rate monotonically increases as utilization accumulates. This strictly models the cumulative physical fatigue of mechanical components, such as traction motors and braking systems.

For Model v2, the shape parameter is strictly defined as $\beta = 2$. Consequently, the exponent in the hazard function becomes linear ($\beta - 1 = 1$),

which models the rapidly increasing, cumulative wear-out rate characteristic of heavy rail locomotion as we can see in the figure 3.2. This mathematical formulation explicitly weights the stochastic sampling, ensuring that the DRL agent must learn to mitigate risk by avoiding the over-utilization of degraded assets.

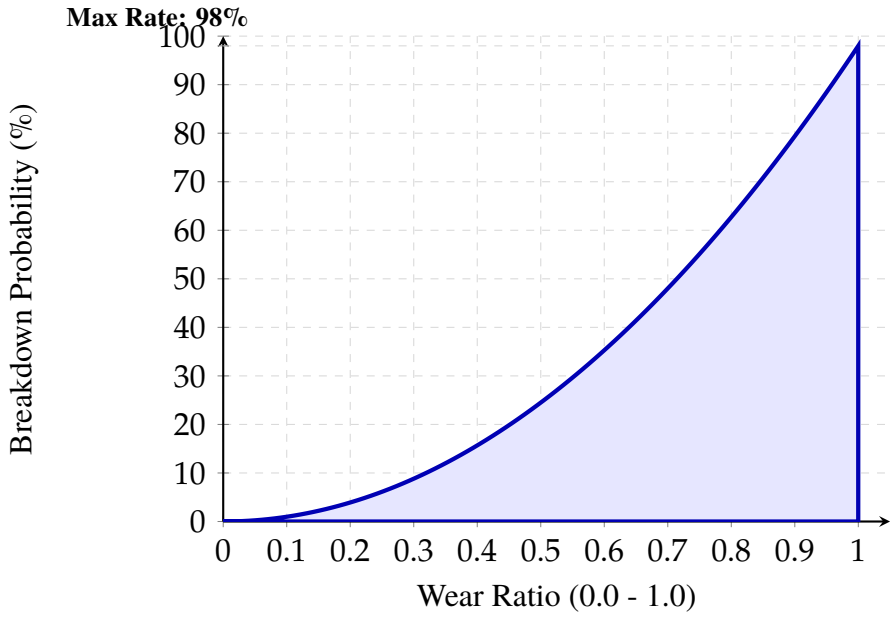


Figure 3.2: The implemented stochastic failure probability mapping. Driven by a shape parameter of $\beta = 2$, the breakdown probability scales quadratically with the locomotive's cumulative wear ratio, peaking at an artificially capped maximum hazard rate of 98%.

3.11 Model v3

The final architectural configuration, Model v3, elevates the agent from a reactive dispatcher to a proactive asset manager. While the deterministic forced maintenance threshold ($d_{max} = 12,000$ km) remains active as an environmental fail-safe, the agent is now imbued with the capacity to autonomously schedule early maintenance interventions. The core architecture remains an extension of the bifurcated Actor network (Shared-Weight and Selector MLPs) introduced in previous configurations, now operating within an expanded state-action space and a highly dynamic reward manifold.

3.11.1 Observation Space Augmentation

To facilitate proactive decision-making without exponentially inflating the computational complexity of the network, the observation space employs deliberate feature engineering. The global feature vector is augmented with six new pre-computed macroscopic statistics, detailing the fleet's health distribution at both Station A and Station B. These augmentations are detailed in Table 3.9.

Table 3.9: Augmented Global Features for Proactive Maintenance (Model v3)

Metric Name	**Description**	**Range**
Minimum Health	The absolute lowest degradation factor (h) currently present among the idle fleet at the respective station.	$[0, 1]$
Average Health	The mean degradation factor (μ_h) of the idle fleet at the respective station.	$[0, 1]$
Critical Count Ratio	The fraction of the local fleet designated as "critical." An asset is defined as critical if its health falls strictly below $h < 0.3$.	$[0, 1]$

While a sufficiently deep neural network possesses the theoretical representational capacity to deduce these metrics directly from the raw locomotive status arrays, explicitly providing these macro-state abstractions significantly flattens the learning curve. This state abstraction reduces the cognitive load on the agent, accelerating gradient convergence by directly exposing the critical thresholds required for optimal maintenance scheduling.

3.11.2 Action Space and Sequential Logic

To accommodate the new operational capabilities, the continuous action space is expanded to $N + 4$ elements. This vector comprises $N = 10$ continuous locomotive preference weights, two station-level dispatch selectors, and two newly introduced station-level *maintenance selectors*.

Because dispatching and maintenance draw from the same finite pool of assets, the environment's 'step()' function processes the continuous action vector in two strict sequential phases to guarantee operational safety:

1. **Maintenance Phase:** The environment first evaluates the maintenance selectors. If proactive service is triggered at a station, the system

scans the local idle fleet and selects the locomotive assigned the **lowest** preference weight by the agent. This cleverly inverts the preference logic: the agent assigns high weights to ”fit” locomotives for travel, naturally leaving the least fit (high-mileage) assets with the lowest scores, effectively designating them for maintenance. These assets are immediately transitioned to a maintenance state and masked out of the available pool.

2. **Dispatch Phase:** Following the extraction of maintenance assets, the environment evaluates the dispatch selectors and utilizes the remaining idle locomotives to compose train schedules based on the highest remaining preference weights.

To translate the continuous network outputs $v \in [-1,1]$ into discrete operational commands, the environment applies the piecewise mapping functions defined in Equations 3.5 and 3.6.

$$N_{dispatch}(v) = \begin{cases} 0 & \text{if } v < -0.8 \\ 2 & \text{if } -0.8 \leq v < 0.2 \\ 3 & \text{if } v \geq 0.2 \end{cases} \tag{3.5}$$

$$N_{maintenance}(v) = \begin{cases} 0 & \text{if } v < -0.2 \\ 1 & \text{if } -0.2 \leq v < 0.6 \\ 2 & \text{if } v \geq 0.6 \end{cases} \tag{3.6}$$

3.11.3 Critic Neural Network Architecture

In previous configurations, the Critic network ($\hat{V}_\phi(s)$) ingested a raw, flattened 1D array of the entire micro-state. Because the Critic lacked the permutation-invariant Shared-Weight structure of the Actor, it struggled to map these raw positional encodings to accurate state-value predictions, resulting in high variance during advantage estimation.

To rectify this, the Critic was redesigned to mirror the ”macro-awareness” of the Actor. Rather than processing the raw fleet array, the updated Critic evaluates the environment strictly through state abstractions. As visualized in Figure 3.3, the Critic concatenates the augmented global features with the total scalar count of available locomotives. This vector is processed through a two-layer MLP (128 neurons each, utilizing LeakyReLU activations with $\alpha = 0.03$), outputting a single scalar state-value. This architectural shift

from micro-state to macro-state evaluation drastically stabilizes the value prediction.

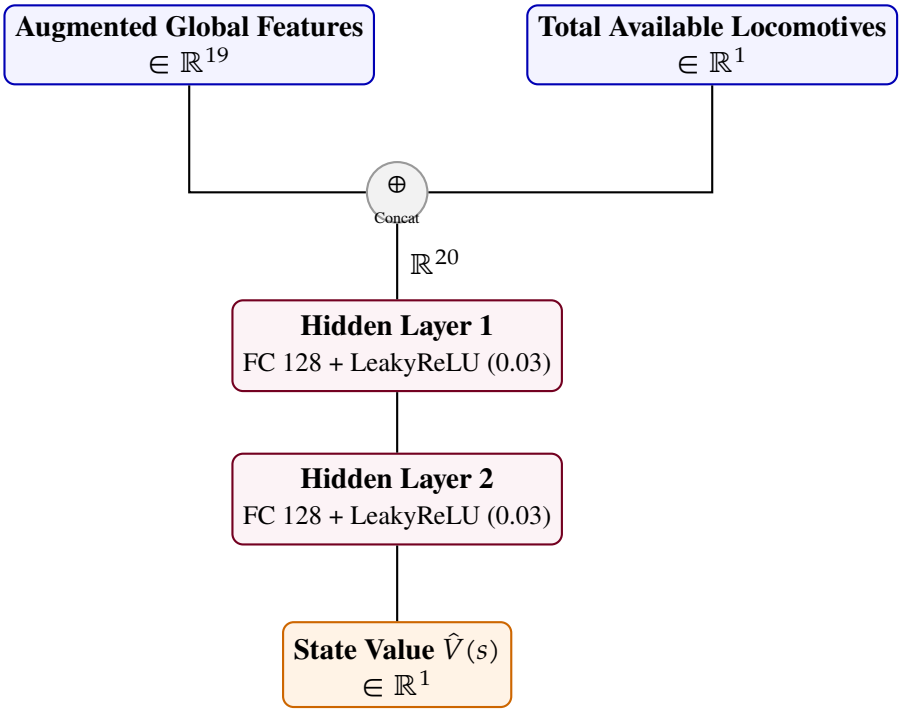


Figure 3.3: The upgraded Critic Network architecture for Model v3. By exclusively processing macro-state abstractions rather than raw positional micro-states, the network achieves highly stable baseline value estimations.

It is crucial to note that unlike the Actor's Shared-Weight MLP, which ingests a 15-dimensional vector by concatenating global features with a specific locomotive's 2-dimensional micro-state, the Critic completely discards individual asset features. Instead, it processes a strictly 20-dimensional macro-state: the 19 augmented global features plus the scalar count of available locomotives.

3.11.4 Reward Shaping for Maintenance

The introduction of proactive maintenance necessitates a fundamental restructuring of the reward function to guide the agent's interaction with the three distinct maintenance paradigms:

- **Stochastic Failures:** These intra-transit breakdowns remain purely random environmental dynamics. Consequently, they do not trigger a

direct penalty, forcing the agent to simply adapt and reschedule around the stochastic disruption.

- **Deterministic Forced Maintenance:** If an asset reaches the hard 12,000 km limit, the agent is heavily penalized via the `FORCED_MAINTENANCE_PENALTY`. In real-world logistics, pushing an asset to its absolute operational limit drastically increases the probability of catastrophic, structural failure, requiring significantly longer repair durations. This penalty serves as a proxy for that operational risk, strongly disincentivizing reactive management.

- **Proactive Maintenance:** To prevent the agent from exploiting the system by continuously cycling fully operational assets through maintenance, a dynamic `MAINTENANCE_PENALTY` is applied whenever a proactive service is triggered. The penalty is derived via a non-linear polynomial function based on the asset's current health $h \in [0, 1]$:

$$P_{\text{maint}}(h) = -15.0 \cdot h^3 \tag{3.7}$$

  This cubic formulation is a deliberate architectural choice. It exponentially punishes the servicing of highly operational assets ($h \to 1$), while flattening to near-zero for severely degraded assets ($h \to 0$). This mathematically aligns the agent's objective with corporate fiscal realities, where maintenance incurs distinct capital expenditures and must be optimized against actual physical degradation.

3.12 Explainability and Decision Support (XAI)

A fundamental challenge in deploying Deep Reinforcement Learning (DRL) within critical industrial infrastructure is the inherent opacity of the agent's decision-making process. To foster operational trust and provide actionable decision support, this thesis introduces a novel visualization framework. This framework is explicitly designed to emulate the graphical topologies of established, industry-standard dispatching tools utilized by GreenCargo's human planners, thereby bridging the gap between algorithmic optimization and human-centric interpretability.

3.12.1 Emulating Industry-Standard Topologies

Modern professional dispatching software (e.g., Plata) typically partitions operational data into three distinct, hierarchical horizontal zones. The proposed XAI interface draws heavy structural inspiration from this paradigm to ensure cognitive familiarity for the end-user:

1. **Macro-Level Fleet Activity (Train Granularity):** This upper tier tracks the aggregated activity of specific locomotive classes (e.g., Series 9124). In industry tools, dual-layered bars represent forecasted versus actual data. A critical feature of this tier is the visualization of capacity deficits (often rendered as red blocks). A deficit indicates a mathematical shortage of required motive power for a scheduled departure. In a real-world scenario, this visually prompts the human dispatcher to execute a manual override , either delaying the service, canceling it, or sub-classing (borrowing an alternatively certified locomotive).

2. **Shift and Routing Allocation:** The middle tier translates abstract schedules into concrete logistical routing. It differentiates between active freight hauling and ”light engine” movements (deadheading for repositioning). Crucially, this tier exposes *temporal buffer thresholds* , the idle gaps between active tasks. For a human planner, evaluating these gaps is essential for assessing schedule fragility; a densely packed schedule with negligible buffer time possesses a high risk of cascading delays.

3. **Micro-Level Asset Tracking:** The foundational tier provides granular tracking of specific physical assets (e.g., locomotive ID ‘4017.MB1E‘). It visualizes static geographical assignments, tentative heuristic allocations, and critical technical metadata (e.g., winterization equipment or heating capabilities), ensuring the dispatcher is aware of localized hardware constraints.

While the proposed DRL visualization heavily maps to these concepts, fundamental architectural differences remain due to the nature of simulated algorithmic evaluation. These distinctions are synthesized in Table 3.10.

Table 3.10: Architectural Comparison: DRL Visualization vs. Industry-Standard Interfaces

Feature Domain	Proposed DRL Visualization	Industry-Standard Interface
Logical Structure	**Decoupled Entity Tracking:** Train services and locomotive assets are segregated into distinct horizontal blocks for post-hoc analysis.	**Integrated Workflow:** Views are vertically stacked, allowing dispatchers to interactively ”drag and drop” assets into schedule shifts.
Primary Purpose	**Algorithmic Evaluation:** Designed to passively visualize reward accumulation, policy drift, and relative lateness scores.	**Operational Manipulation:** Designed as an active, interactive sandbox for human-driven logistical execution.
Temporal Anchoring	**Synthetic Horizons:** Evaluated over abstracted, zero-indexed temporal horizons (e.g., Hour 0 to Hour 140).	**Empirical Horizons:** Strictly anchored to real-world, calendar-based temporal data.

3.12.2 Iterative Refinement of the Visualization Framework

The initial configuration of the evaluation interface (visualized in Figure 3.4 and Figure 3.5) suffered from severe visual scalability limitations. In this legacy format, the temporal X-axis was tracked in absolute hours, and each individual train dispatch generated a distinct, isolated row. Consequently, scaling the simulation to a standard operational parameter (such as 6 daily trains and 12 locomotives evaluated over a 10-day horizon) resulted in catastrophic vertical bloat. As demonstrated in the legacy views, evaluating highly populated episodes became visually intractable, obscuring the agent's strategic intent.

To resolve these UI/UX bottlenecks, a heavily optimized charting framework was developed (Figure 3.6). The refined architecture introduces several critical enhancements designed to maximize visual density and operational clarity:

- **Consolidated Train Topologies:** Instead of generating infinite vertical rows, train services are now grouped and consolidated. A single row pairs the planned schedule with the actual agent-driven execution. Delays are explicitly rendered as yellow buffer extensions prior to the green transit blocks, instantly highlighting where the agent's policy drifted from the optimal timetable.

- **Categorical Maintenance Rendering:** Within the locomotive section, maintenance events are now distinctly categorized utilizing a combination of color theory and categorical hatching. Normal proactive maintenance is rendered in yellow (‘///‘), deterministic forced maintenance in orange (‘//‘), and stochastic random breakdowns in red (‘/‘).

- **Directional Glyphs:** Text-heavy destination logging has been replaced with clean, directional glyphs (e.g., ‘> B‘), immediately conveying the spatial trajectory of the asset.

- **Standardized Localization and Scaling:** To align with international academic standards, all UI localization was transitioned from Swedish to English. Furthermore, the X-axis was rescaled from absolute hourly tracking to discrete daily increments, providing a much cleaner macro-level perspective of the multi-day rolling horizon.

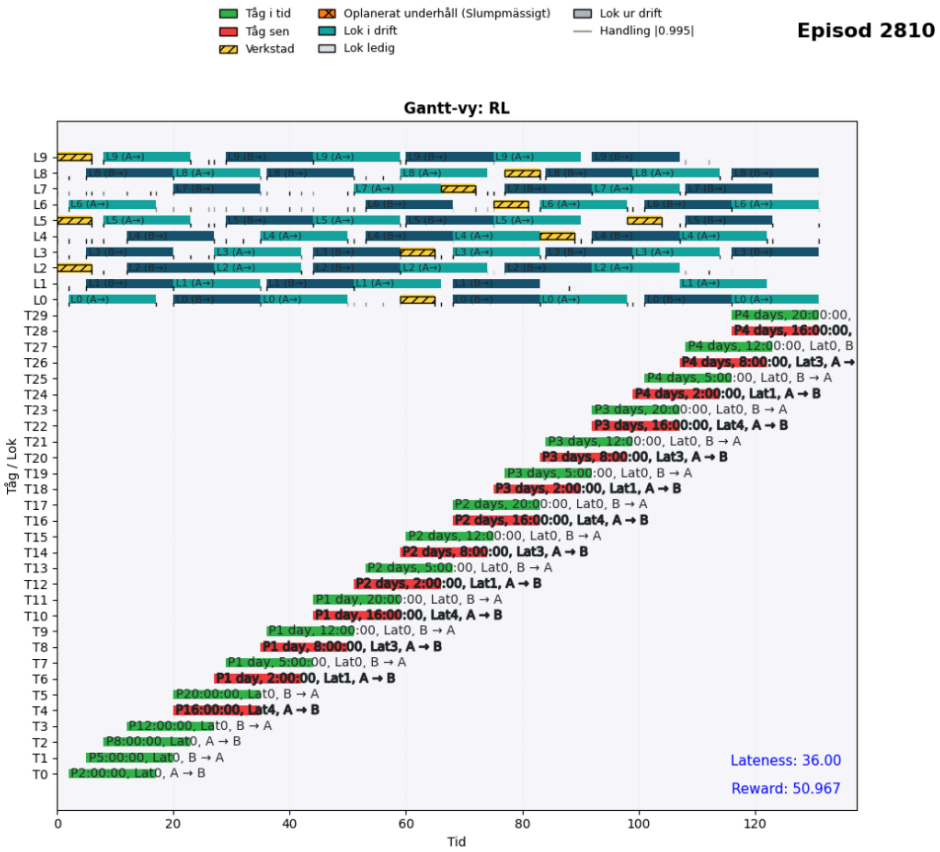


Figure 3.4: The legacy (V1) Gantt chart formulation. Utilizing an hourly X-axis and Swedish localization, the chart isolates every entity, leading to fragmented visual tracking.

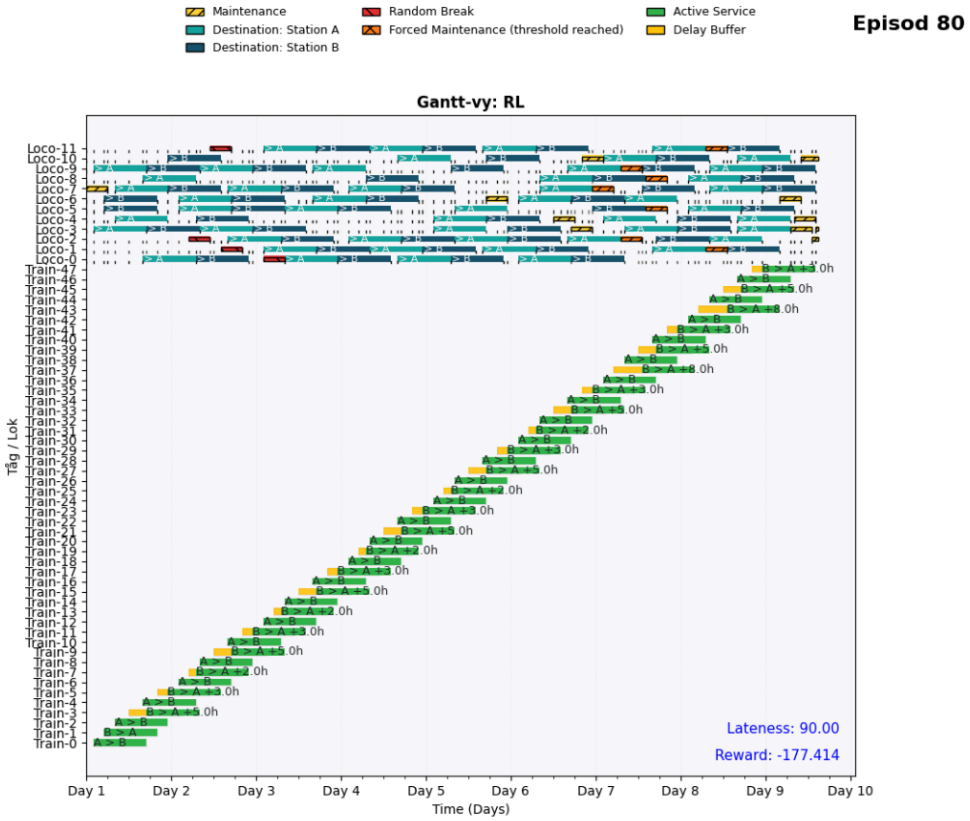


Figure 3.5: The legacy (V2) Gantt chart demonstrating severe vertical scaling issues. As the episode progresses, the strict one-row-per-train logic creates visual bloat, rendering long-horizon evaluations intractable for human planners.

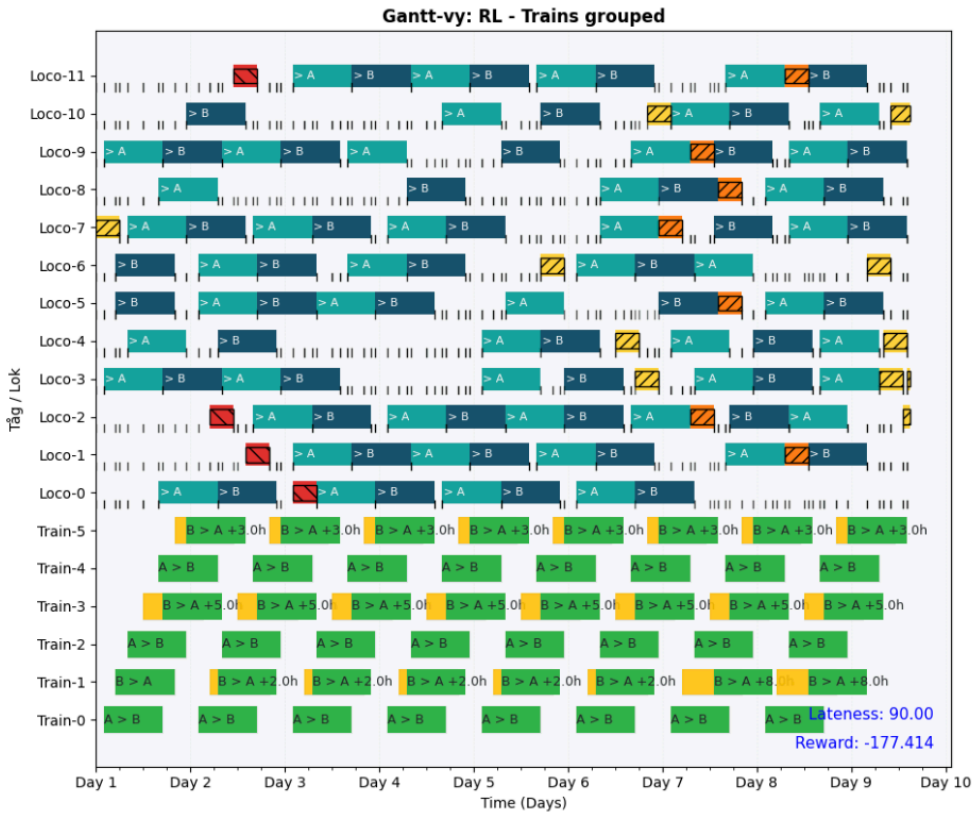


Figure 3.6: The refined DRL evaluation interface. Trains are consolidated into grouped rows detailing actual versus scheduled transit, maintenance is categorically hatched, and the temporal axis is scaled to multi-day increments, drastically enhancing operational interpretability.

Chapter 4

Results and Analysis

This chapter presents a comprehensive empirical evaluation of the three proposed Deep Reinforcement Learning models. The analysis begins by formalizing the evaluation framework and defining the core performance metrics derived from GreenCargo's operational objectives. Subsequently, the quantitative results, performance differentials, and emergent policy behaviors across Model v1, v2, and v3 are systematically analyzed.

Due to the significant computational overhead associated with the DRL pipeline (where individual training runs necessitated between 15 and 35 hours of continuous execution on the RISE computing clusters) the experimental phase was constrained to approximately 40 total discrete trials. These runs were distributed across the three architectural configurations, with the foundational Model v1 receiving the highest concentration of trials to establish a robust baseline hyperparameter configuration prior to advancing environmental complexity.

4.1 Evaluation Framework and Empirical Protocols

To rigorously quantify the agent's performance, the evaluation framework must align seamlessly with GreenCargo's overarching logistical mandates: minimizing schedule lateness, ensuring strict timetable adherence, distributing asset workload equitably, and preventing forced mechanical interventions. To this end, an isolated, post-training validation protocol was established.

Following the conclusion of the training phase, the optimized neural network parameters are frozen. The agent is then deployed across an

evaluation horizon of 500 distinct, randomly seeded episodes. Crucially, to assess the true capacity of the learned policy, the stochastic exploration noise inherent to the PPO training loop is disabled. Rather than stochastically sampling actions from the full parameterized Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, the environment executes a deterministic policy by directly extracting and applying the mean μ produced by the Actor network. The final performance metrics are subsequently averaged across these 500 evaluation episodes to mitigate seed-based environmental variance.

The primary validation metrics utilized to benchmark the agent against the deterministic baseline heuristic are formally defined in Table 4.1.

Table 4.1: Core Evaluation Metrics Computed over the 500-Episode Validation Phase

Metric	**Operational Definition**
Model Win Rate	The fractional frequency across the 500 episodes wherein the DRL agent successfully achieved a lower cumulative lateness than the heuristic baseline.
Model Lose Rate	The fractional frequency wherein the agent yielded a higher cumulative lateness than the heuristic baseline.
Delta Lateness Average	The mean difference in accumulated schedule lateness between the proposed agent and the baseline.
Delta Success Rate Average	The mean differential ratio of train services dispatched exactly at their scheduled departure time when comparing the agent's execution against the baseline.

These quantitative indicators serve as direct mathematical proxies for the corporate logistical objectives outlined in the introduction. The *Delta Lateness Average* and the *Model Win/Loose Rate Average* directly evaluate the agent's capability to minimize logistical delays in complex routing scenarios. Concurrently, the *Delta Success Rate Average* strictly measures the efficiency of schedule completion, ensuring that the maximization of on-time departures is achieved without triggering unnecessary forced maintenance events.

4.2 Model v1 Results

The empirical evaluation of Model v1 progressed through a rigorous phase of hyperparameter optimization and architectural refinement. During the initial

nineteen experimental trials, the DRL agent failed to converge on an optimal policy, demonstrating operational parity with the random baseline heuristic. In these early configurations, utilizing a Global MLP, the network suffered from severe positional overfitting; the agent struggled to evaluate individual asset fitness, effectively operating without a coherent dispatch strategy.

A critical performance inflection occurred at Trial 20 following the integration of the Shared-Weight actor architecture. By enforcing permutation invariance, the agent transitioned from memorizing array indices to genuinely evaluating locomotive state parameters. This architectural paradigm shift enabled the agent to dynamically allocate motive power (such as assigning three locomotives when payload or directional balancing dictated) thereby strictly outperforming the rigid baseline heuristic. The improvements described can be seen in the Figure 4.1 which plots the Delta Lateness Average and the Model Win Rate from Trial 10 to Trial 23.

The exact granular results from the core developmental phase of Model v1 are synthesized in Table 4.2. This tabular data details the monotonic improvements in delay reduction, reward acquisition, and dispatch success rates, serving as empirical validation for the architectural decisions implemented during the experimental loop.

4.3 Model v2 Results

Building upon the stable policy established in Model v1, the second architectural iteration introduced environmental stochasticity via the Continuous Hazard Model. The objective was to evaluate policy robustness when subjected to real-time, Weibull-distributed mechanical failures ($\beta = 2.0$), which dynamically transition locomotives into forced maintenance based on cumulative wear ratios.

Table 4.3 details the performance of the agent during Trials 24 and 25 under these adversarial conditions. While the introduction of 4 to 5 random failures per episode predictably decreases the overall Win Rate compared to the peak of Model v1, the agent still maintains a dominant operational advantage, successfully mitigating cascading delays far more effectively than the deterministic heuristic. Figure 4.2 provides a direct comparison between the peak performance of Model v1 (Trial 22, deterministic environment) and the best-performing iteration of Model v2 (Trial 25, stochastic environment). While the introduction of random asset failures predictably degrades both the win rate and the absolute lateness reduction the core Shared-Weight architecture is robust enough to handle unexpected capacity deficits without

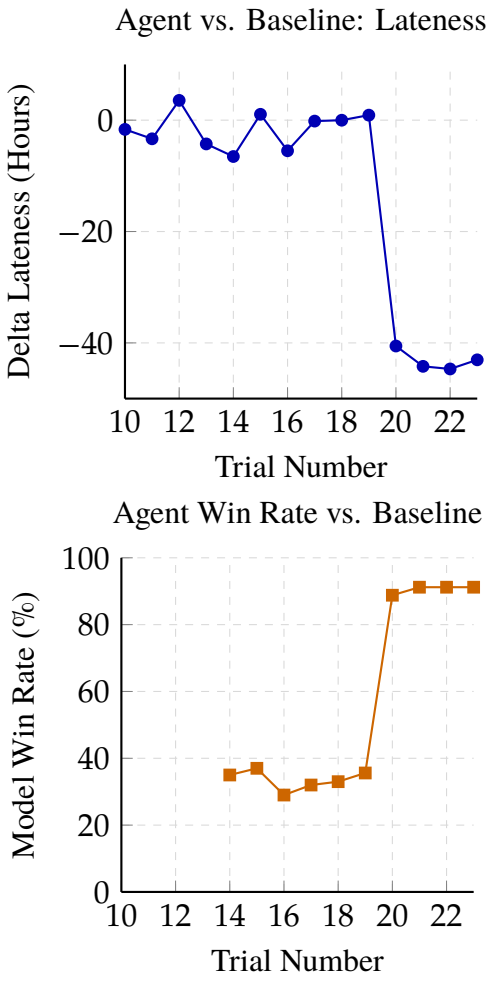


Figure 4.1: Performance evolution of Model v1 across intermediate trials. The charts highlight the period of operational parity (Trials 10–19) followed by a definitive phase shift in policy performance (Trial 20+) resulting from the Shared-Weight architectural upgrade.

Table 4.2: Comprehensive performance metrics for the development of Model v1 (Trials 10–23). Missing values (–) indicate metrics that were either not tracked or not relevant during early experimental phases prior to framework stabilization.

Trial	Delta Lateness (h)	Win Rate (%)	Lose Rate (%)	Delta Success Rate (%)
10	−1.66	–	–	0.40
11	−3.35	–	–	0.70
12	3.55	–	–	0.30
13	−4.27	–	–	−0.30
14	−6.53	35.00	35.00	−0.20
15	1.05	37.00	30.00	0.60
16	−5.50	29.00	42.00	0.50
17	−0.17	32.00	32.80	−0.10
18	−0.01	33.00	33.00	0.50
19	0.90	35.60	34.00	−0.10
20	−40.56	88.80	7.80	16.20
21	−44.21	91.20	5.80	18.40
22	−44.68	91.20	5.80	19.10
23	−43.04	91.20	5.80	17.00

Note: All delta values are calculated as strictly relative to the baseline heuristic. A negative *Delta Lateness* denotes fewer accumulated delay hours (improved performance). The remaining percentage points for the Win/Lose rates (to reach 100%) account for ties where the agent and baseline achieved identical outcomes.

Table 4.3: Performance metrics for Model v2 under stochastic failure conditions (Trials 24–25). The table highlights the agent's robustness when subjected to unexpected mechanical breakdowns.

Trial	Random Failures	Delta Lateness (h)	Win Rate (%)	Lose Rate (%)	Delta Succe
24	5	−36.13	81.40	18.60	1
25	4	−38.12	81.60	17.80	1

Note: Random Failures denotes the discrete number of unexpected stochastic breakdowns injected per evaluation episode. Despite these targeted capacity deficits, the DRL agent consistently preserved a significant Delta Lateness advantage over the baseline.

suffering from policy collapse.

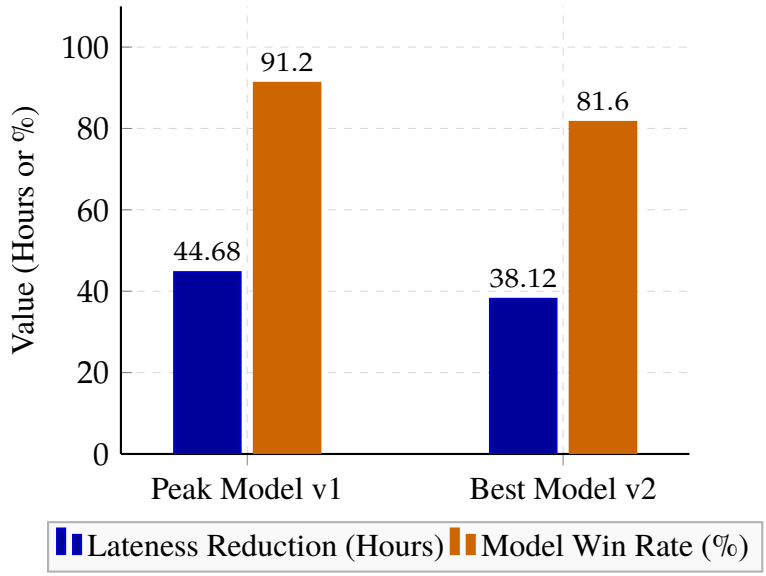


Figure 4.2: Direct performance comparison between the peak configuration of Model v1 (deterministic environment) and the most robust configuration of Model v2 (stochastic environment with 4 random mechanical failures). The chart illustrates the minor but expected performance degradation caused by real-time asset breakdowns.

4.4 Model v3 Results

Model v3 represents the culmination of the proposed framework, augmenting the agent with autonomous proactive fleet management capabilities via Inverse Weight Selection and a macro-state processing Critic network. In this configuration, the primary operational objective transitioned from reactive delay mitigation to predictive maintenance scheduling.

The expansion of the continuous action space to include maintenance selectors initially introduced high variance within the Proximal Policy Optimization (PPO) optimization loop. Consequently, Trials 26 through 34 were dedicated to rigorous hyperparameter tuning, specifically targeting the learning rate, the entropy coefficient (`ent_coef`), the trust region clipping bounds (`clip_range`), and the total episodic training horizon.

As illustrated in Figure 4.4 and Figure 4.3, Trials 26 to 30 focused primarily on establishing a stable hyperparameter configuration capable of mitigating gradient variance and stabilizing the core PPO metrics.

Following policy stabilization, subsequent trials systematically extended the computational training horizon. The experimental sequence was deliberately terminated at Trial 34; extending the optimization beyond 60,000 episodes yielded diminishing returns without monotonic performance improvements, confirming the empirical performance ceiling of the proposed architecture.

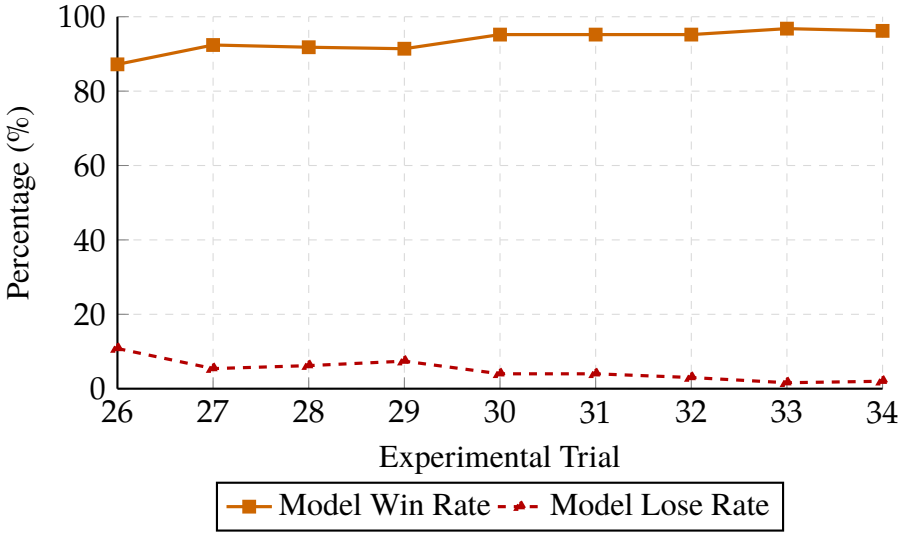


Figure 4.3: Comparative progression of the Model Win Rate and Lose Rate across the evaluation of Model v3. By autonomously managing fleet health, the agent nearly eliminates scenarios where the heuristic out-performs the network, converging to a strict operational dominance at Trial 33.

As the network optimization progressed, the comparative reliability metrics demonstrated substantial improvement. The agent's Win Rate steadily escalated beyond 96%, concurrently compressing the Lose Rate to a statistically negligible fraction ($< 2\%$). By Trial 33, the policy reached its absolute peak, systematically mitigating over 51 hours of accumulated logistical delay per episode relative to the baseline heuristic.

Figure 4.5 isolates the Delta Success Rate Average, measuring the proportional increase in completely on-time train dispatches achieved by the agent versus the baseline. The progressive upward trend correlates directly with the integration of early maintenance logic; by servicing locomotives prior to critical failure thresholds, the agent mathematically ensures a higher volume of perfectly executed dispatches.

A comprehensive synthesis of the evaluation metrics recorded during this developmental phase is presented in Table 4.4.

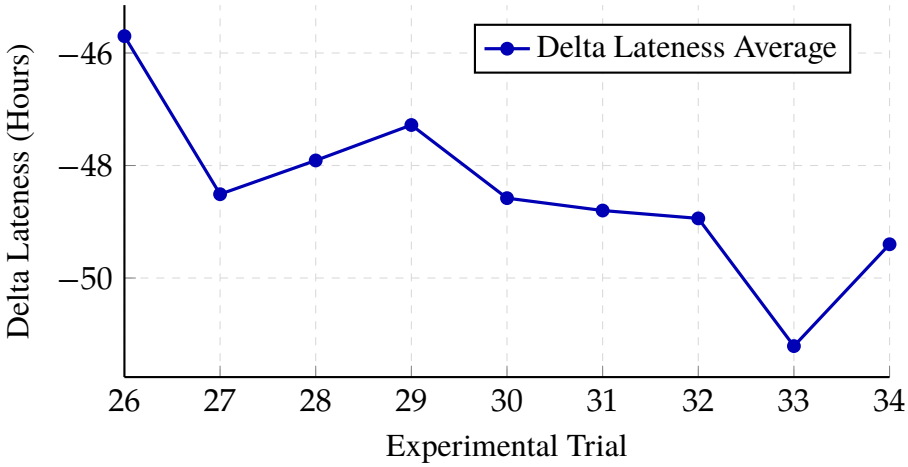


Figure 4.4: Evolution of the Delta Lateness Average during the training of Model v3. The steady downward trajectory indicates the agent's increasing proficiency in utilizing proactive maintenance to eliminate structural delays, peaking at an unprecedented −51.21 hours in Trial 33.

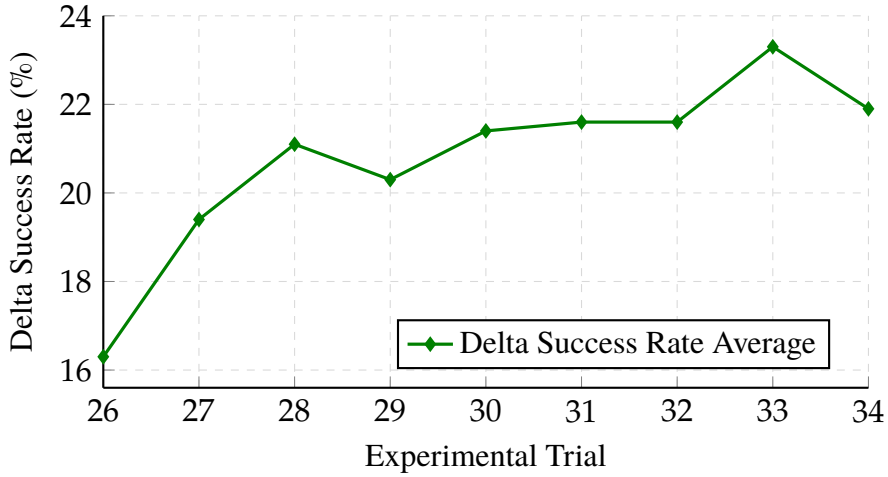


Figure 4.5: The Delta Success Rate Average for Model v3. The graph quantifies the agent's absolute superiority in adhering strictly to the scheduled departure timetable, culminating in a +23.3% operational advantage over the baseline heuristic.

Table 4.4: Comprehensive performance metrics for the development and optimization of Model v3 (Trials 26–34). The table encapsulates the final performance ceiling of the proactive fleet management architecture.

Trial	Delta Lateness (h)	Win Rate (%)	Lose Rate (%)	Delta Success Rate (%)
26	−45.70	87.20	10.80	16.30
27	−48.51	92.40	5.40	19.40
28	−47.91	91.80	6.20	21.10
29	−47.28	91.40	7.40	20.30
30	−48.58	95.20	4.00	21.40
31	−48.80	95.20	4.00	21.60
32	−48.94	95.20	3.00	21.60
33	−51.21	96.80	1.60	23.30
34	−49.40	96.20	2.00	21.90

Note: All delta values are calculated relative to the baseline heuristic. Negative *Delta Lateness* represents superior delay mitigation. The gap between combined Win/Lose Rates and 100% accounts for operational parity (ties).

4.5 Model v3: Evaluation under Stochastic Hazard Conditions

The final experimental phase evaluates Model v3 under stochastic operational conditions, representing a full architectural synthesis of the continuous hazard framework (Model v2) and the proactive maintenance architecture (Model v3). This phase rigorously tests the agent's reliability, structural resilience, and dynamic re-planning efficacy. The primary objective is to quantify the performance differential when the agent is granted the autonomy to schedule proactive maintenance while concurrently navigating unpreventable, probabilistic mechanical failures.

Prior deterministic evaluations demonstrated that the isolated introduction of proactive maintenance increased the Model Win Rate by over 5 percentage points while reducing the Model Lose Rate by approximately 4 percentage points. While these deterministic gains are substantial, this section expands the empirical scope to assess the policy's robustness when subjected to true environmental uncertainty via the Weibull-distributed continuous probability model.

Figure 4.6 plots the Delta Lateness Average for Model v3 under these stochastic failure conditions. While the absolute peak of delay reduction observed in the strictly deterministic Trial 33 (−51.21 hours) naturally

degrades when the environment injects unpreventable mechanical failures, the agent consistently maintains a performance delta of approximately -45 hours against the heuristic baseline. This sustained latency reduction empirically validates the robustness of the macro-state Critic architecture and the agent's capacity for rapid schedule recovery.

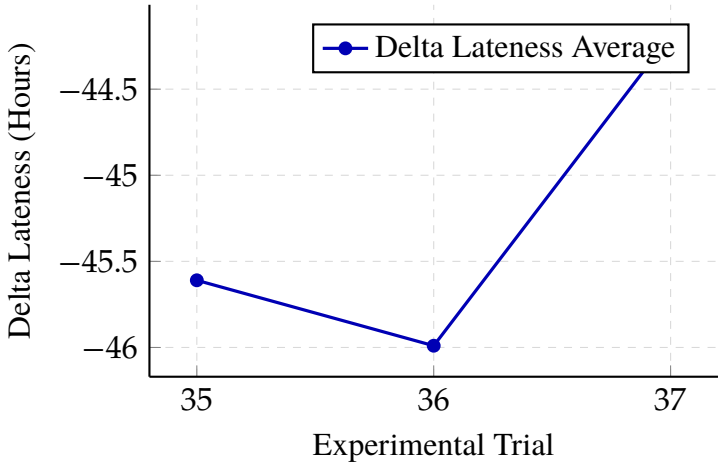


Figure 4.6: Evolution of the Delta Lateness Average during the stochastic evaluation of Model v3 (Trials 35–37). The data demonstrates the agent's capacity to preserve superior delay mitigation even when subjected to dynamic, Weibull-distributed mechanical failures.

By juxtaposing the Model Win Rate against the Model Lose Rate in Figure 4.7 during the stochastic evaluation phase, it becomes evident that the DRL framework does not suffer from policy collapse when unpredicted capacity deficits occur. Even in Trial 37 (where the base hazard rate and Weibull shape parameter were aggressively scaled to induce a higher frequency of random breakdowns) the agent successfully navigated the localized motive power shortages to secure a Win Rate exceeding 92%.

Furthermore, Figure 4.8 visualizes the agent's schedule adherence relative to the heuristic under these stochastic conditions, confirming that the reduction in total lateness does not come at the expense of dispatch punctuality.

Finally, Table 4.5 consolidates the comprehensive empirical results for the proposed framework operating under true environmental uncertainty.

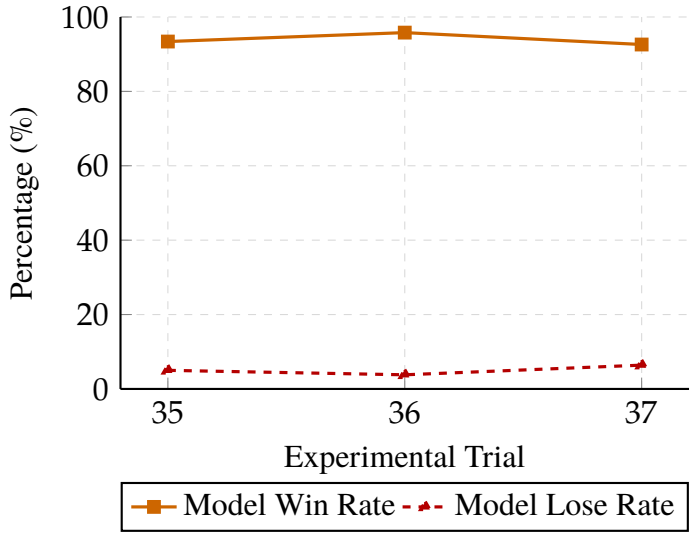


Figure 4.7: Progression of the Model Win Rate and Model Lose Rate metrics for Model v3 evaluated under continuous mechanical hazard constraints. The variance highlights the agent's structural resilience to unpredicted fleet degradation.

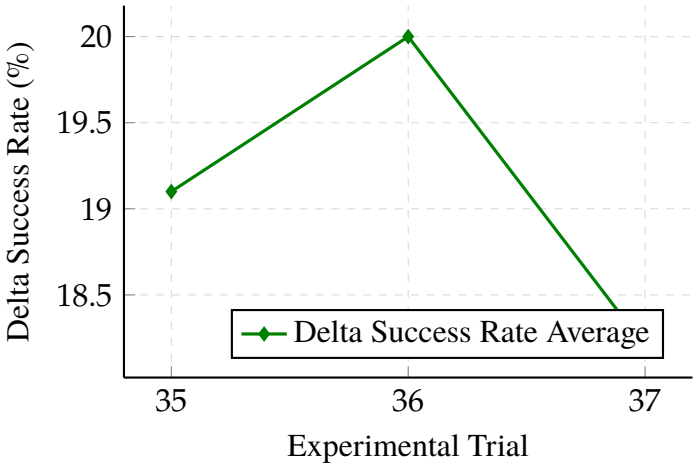


Figure 4.8: The Delta Success Rate Average across the final stochastic trials. The metric isolates the agent's superior capacity to maintain strict timetable punctuality despite randomly injected asset unavailability.

Table 4.5: Comprehensive performance metrics for Model v3 evaluated under the Continuous Hazard Model (Trials 35–37). The metrics quantify the framework's ultimate capability to synthesize proactive maintenance scheduling with reactive crisis management.

Trial	**Delta Lateness (h)**	**Win Rate (%)**	**Lose Rate (%)**	**Delta Success Rate (%)**
35	−45.61	93.40	5.00	19.10
36	−45.99	95.80	3.80	20.00
37	−44.19	92.60	6.40	18.20

Note: All evaluation metrics are calculated relative to the baseline heuristic over a multi-episode horizon. Trial 37 explicitly scales the Weibull shape and hazard rate parameters to induce maximum adversarial complexity.

Chapter 5

Discussion

The empirical results in Chapter 4 make it possible to address the research questions introduced in Section 1.1.

For *RQ T.1*, the results indicate that the rolling horizon DRL formulation improves planning quality and operational adaptability when compared with the sequential baseline heuristic. In particular, Model v3 demonstrates the strongest performance, since it is able to coordinate locomotive dispatching with proactive maintenance. In the deterministic evaluation, this model achieves a win rate of 96.8% and reduces average episodic lateness by more than 51 hours relative to the baseline. When stochastic locomotive failures are introduced, the policy remains stable, with a win rate above 92% and an average lateness reduction of approximately 44–46 hours. These results suggest that the learned policy is not only effective in nominal conditions, but also capable of adapting to disruptions.

For *RQ T.2*, the results show that explainability can be partially addressed by translating the agent's decisions into visual and operationally meaningful representations. The proposed interface allows human planners to inspect locomotive assignments, delay patterns, and maintenance actions in a format closer to existing dispatching tools. While this does not make the internal neural policy fully transparent, it improves the interpretability of the system at the decision-support level. Therefore, the contribution of the XAI component is mainly practical: it helps planners understand and evaluate the agent's recommendations without treating the DRL model as a fully autonomous black box.

5.1 Limitations

While the proposed agent demonstrated superiority over the greedy baseline, the current research is bounded by several deliberate experimental constraints that must be acknowledged.

First, the evaluation relies on a heuristic baseline rather than an industry-standard exact solver (e.g., Mixed-Integer Linear Programming). To definitively quantify the absolute optimality of the agent's policy, future evaluations must benchmark the DRL framework against traditional Operations Research methodologies.

Second, a primary limitation of this study is the reliance on synthetic data. The schedules, demand urgencies, and fleet degradation models were synthesized rather than derived from empirical historical logs. Consequently, the agent's policy, while sound within the simulation, lacks exposure to the nuanced, latent complexities of real-world rail logistics.

Third, the simulated environment represents a high degree of environmental abstraction. The spatial topography is restricted to a dual-node network (two stations), which vastly underrepresents the complex, multi-node routing challenges inherent to commercial freight networks. Furthermore, the simulation currently abstracts away critical operational variables, including meteorological impacts, heterogeneous freight material constraints, varying maintenance tier durations, and the localized temporal latencies required for mechanical shunting and locomotive coupling.

Finally, the scope of the empirical validation was bounded by computational resource constraints. Due to the significant processing time required to train the continuous PPO architecture in highly stochastic environments, the framework was subjected to a restricted number of experimental trials, precluding an exhaustive hyperparameter grid search.

Chapter 6

Conclusions and Future Work

This final chapter synthesizes the core findings of the thesis, directly addressing the foundational research questions, and outlines strategic pathways for advancing the proposed framework toward real-world industrial deployment.

6.1 Conclusions

This research aimed to determine the viability of utilizing DRL to solve the LAP under both deterministic and stochastic conditions. Based on the architectural development and empirical evaluations, the following core conclusions are drawn:

- **Robustness to Stochasticity:** DRL can effectively optimize scheduling operations not only under rigid deterministic conditions but also when subjected to continuous, Weibull-distributed mechanical failures, outperforming naive heuristic dispatching.
- **Proactive Asset Management:** The framework successfully demonstrates that a single DRL agent can concurrently manage multi-objective tasks—specifically, balancing immediate dispatch requirements with the strategic execution of proactive maintenance.
- **Architectural Necessity:** While DRL networks demand significant episodic interactions to converge, the system scales effectively with the number of assets. However, achieving this scalability requires strict architectural discipline, specifically the implementation of permutation-invariant Shared-Weight MLPs and macro-state Critic evaluations.

- **Mitigating the Black Box:** The inherent opacity of neural networks can be effectively mitigated. By translating the agent's continuous output vectors into industry-standard, human-readable Gantt visualizations, the framework transitions from an autonomous "black box" to a highly interpretable decision-support tool for human planners.

6.2 Future Work

While this thesis establishes a viable DRL methodology for the LAP, the framework requires substantial expansion to meet the rigorous demands of commercial deployment at GreenCargo. Future research trajectories are divided into near-term architectural extensions and long-term operational integration.

Near-Term Architectural Extensions: Immediate future work should focus on elevating the benchmarking standards by evaluating the agent against exact mathematical solvers to quantify the optimality gap. Additionally, the neural network topology could be advanced. Integrating Recurrent Neural Networks (e.g., LSTMs) alongside the existing MLPs could provide the agent with an implicit temporal memory, allowing it to capture sequential operational dependencies without relying solely on instantaneous observation representations. Expanding the available computational hardware to conduct exhaustive hyperparameter optimization will also be critical for refining policy stability.

Long-Term Operational Integration: To bridge the gap between simulation and reality, future improvements must systematically strip away the current environmental abstractions. The temporal resolution of the environment should be increased from hourly intervals to 15-minute operational steps to capture micro-latencies such as shunting and coupling. The spatial network must be expanded to encompass a highly connected, multi-node topography representing the true Swedish rail network. Most importantly, the model must be trained and evaluated using empirical, historical operational data from GreenCargo. Exposing the agent to real-world datasets will allow it to discover and exploit latent operational patterns, ultimately transitioning the framework from an academic proof-of-concept to a deployable industrial asset.

References

[1] F. Piu and M. G. Speranza, "The locomotive assignment problem: a survey on optimization models," *International Transactions in Operational Research*, vol. 21, no. 3, pp. 327–352, 2014. doi: https://doi.org/10.1111/itor.12062. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12062

[2] R. K. Ahuja, J. Liu, J. B. Orlin, D. Sharma, and L. A. Shughart, "A multi-exchange neighborhood search algorithm for the locomotive assignment problem," *Operations Research*, vol. 53, no. 6, pp. 903–914, 2005. doi: 10.1287/opre.1050.0219

[3] B. Vaidyanathan, R. K. Ahuja, and J. B. Orlin, "A resource-directive decomposition approach for the locomotive assignment problem," *Operations Research*, vol. 56, no. 2, pp. 420–432, 2008. doi: 10.1287/opre.1070.0452

[4] G. P. Basharin, A. N. Langville, and V. A. Naumov, "The life and work of A. A. Markov," *Linear Algebra and its Applications*, vol. 386, pp. 3–26, 2004. doi: 10.1016/j.laa.2003.12.041. [Online]. Available: https://doi.org/10.1016/j.laa.2003.12.041

[5] H. Jiang, R. Gui, Z. Chen, L. Wu, J. Dang, and J. Zhou, "An improved sarsa(λ) reinforcement learning algorithm for wireless communication systems," *IEEE Access*, vol. PP, pp. 1–1, 08 2019. doi: 10.1109/ACCESS.2019.2935255

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, Massachusetts: The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[7] R. A. Howard, *Dynamic programming and Markov processes*. MIT Press, 1960.

[8] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[9] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," University of Cambridge, Department of Engineering, Tech. Rep. CUED/F-INFENG/TR 166, 1994.

[10] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems (NIPS)*, 2000, pp. 1057–1063.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[12] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 1889–1897. [Online]. Available: http://proceedings.mlr.press/v37/schulman15.html

[13] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004. ISBN 978-0521833783. [Online]. Available: https://web.stanford.edu/~boyd/cvxbook/

[14] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep., 1994. [Online]. Available: https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf

[15] J. Achiam, "Spinning up in deep reinforcement learning," https://github.com/openai/spinningup, 2018.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[17] Y. Tang and D. Ha, "The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning," *CoRR*, vol.

abs/2109.02869, 2021. [Online]. Available: https://arxiv.org/abs/2109.02869

[18] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *CoRR*, vol. abs/2006.14171, 2020. [Online]. Available: https://arxiv.org/abs/2006.14171

[19] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of Artificial Intelligence Research*, vol. 13, pp. 227–303, 2000.

[20] A. Kanervisto, C. Scheller, and V. Hautamäki, "Action space shaping in deep reinforcement learning," 2020. [Online]. Available: https://arxiv.org/abs/2004.00980

[21] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1587–1596.

[22] M. Florian, G. Bushell, J. Ferland, G. Guérin, and L. Stewart, "The engine scheduling problem in a railway network," *INFOR: Information Systems and Operational Research*, vol. 14, no. 2, pp. 121–138, 1976. doi: 10.1080/03155986.1976.11731632

[23] A. Scholz, "Locomotive scheduling with maintenance constraints at the swedish state railways," *International Transactions in Operational Research*, vol. 7, no. 4-5, pp. 329–350, 2000. doi: 10.1111/j.1475-3995.2000.tb00204.x

[24] W. B. Powell, K. Papadaki, and M. K. Spivey, "A multiagent, context-specific approach to the locomotive assignment problem," *Transportation Science*, vol. 35, no. 4, pp. 329–358, 2001. doi: 10.1287/trsc.35.4.329.10435

[25] Y. Yang, H. Huo, N. Bešinović, Y. Sun, and S. Ni, "An advanced learning environment and a scalable deep reinforcement learning approach for rolling stock circulation on urban rail transit line," *Transportation Research Part C: Emerging Technologies*, vol. 171, p. 104976, 2025. doi: https://doi.org/10.1016/j.trc.2024.104976. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X24004972

[26] X. Zhao, D. Li, and X. Bao, "A deep reinforcement learning approach for integrated optimization of train scheduling and rolling stock

circulation planning," *Computers & Industrial Engineering*, vol. 213, p. 111784, 2026. doi: https://doi.org/10.1016/j.cie.2025.111784. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835225009301

[27] X. Su, Y. Yue, B. Guo, Z. Cui, and B. Li, "Solving high-speed rolling stock planning with an improved lstm-ppo reinforcement learning algorithm," *Transportmetrica A Transport Science*, 2026. doi: https://doi.org/10.1080/23249935.2026.2653703. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2324993526000369

[28] M. Bröchin, R. Ramseier, and K. Schüpbach, "Rolling stock planning with maintenance constraints by a rolling horizon approach," 04 2023. [Online]. Available: https://www.researchgate.net/publication/391914974_Rolling_Stock_Planning_with_Maintenance_Constraints_by_a_Rolling_Horizon_approach

[29] P. Habibi, M. Tamannaei, and H. Zarei, "Dynamic wagon assignment and reassignment for pick-up and delivery requests: A deep reinforcement learning framework to address uncertainty in locomotive availability," *Applied Soft Computing*, vol. 187, p. 114351, 2026. doi: https://doi.org/10.1016/j.asoc.2025.114351. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494625016643

[30] C. shuo Ying, A. H. Chow, and K.-S. Chin, "An actor-critic deep reinforcement learning approach for metro train scheduling with rolling stock circulation under stochastic demand," *Transportation Research Part B: Methodological*, vol. 140, pp. 210–235, 2020. doi: https://doi.org/10.1016/j.trb.2020.08.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0191261520303829

[31] Department for Transport (DfT), "Tag unit 3.5.6 values of time and vehicle operating costs, transport analysis guidance data book," Department for Transport, United Kingdom, Technical Report, 2014.

[32] A. H. F. Chow and A. Pavlides, "Cost functions and multi-objective timetabling of mixed train services," *Transportation Research Part A*, vol. 113, pp. 335–356, 2018. doi: 10.1016/j.tra.2018.04.027

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org

[34] Y. Liu, J. Wu, and Q. Wu, "Research on optimization of c4 repair operation of harmony electric locomotive based on preventive maintenance," *PLOS ONE*, vol. 20, no. 7, pp. 1–22, 07 2025. doi: 10.1371/journal.pone.0328399. [Online]. Available: https://doi.org/10.1371/journal.pone.0328399

[35] D. Hu, Z. Ouyang, Q. Chen, and H. Fan, "Optimization method of special vehicle maintenance strategy based on reliability and cost-effectiveness ratio," *Military Automation*, vol. 40, no. 7, pp. 72–77, 2021.

[36] H.-y. Zhu, J.-l. Li, L.-m. Wang, Z. Wang, and J.-c. Liu, "Reliability and maintenance cost evaluation of long-period service system," *Modern Defense Technology*, vol. 49, no. 2, pp. 86–94, 2021. doi: 10.3969/j.issn.1009-086x.2021.02.013

[37] H. Dong, Q. Liu, C. Ye, and W. Liu, "Periodic preventive maintenance optimization and a model for high-speed rail brake based on the limitation of reliability," *Journal of University of Shanghai for Science and Technology*, vol. 42, no. 4, pp. 384–389, 2020. doi: 10.13255/j.cnki.jusst.20190520003

[38] W. Weibull, "A statistical distribution function of wide applicability," *Journal of Applied Mechanics*, vol. 18, no. 3, pp. 293–297, 1951.

[39] P. O'Connor and A. Kleyner, *Practical reliability engineering*, 5th ed. John Wiley & Sons, 2012.