Adaptive Distributed Event-Driven Reinforcement Learning for the Dynamic Flexible Job Shop Scheduling Problem

Rui Wu

Supervised by Dr. Jin Yang

Submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy

School of Engineering

College of Science and Engineering

University of Glasgow

Aug-2025

Abstract

Manufacturing systems face growing complexity in the era of Industry 4.0, where competitiveness depends on real-time responsiveness, efficiency, and reliability. Disruptions such as random job arrivals, machine breakdowns, and sequence-dependent setups challenge traditional scheduling. We address the Dynamic Flexible Job Shop Scheduling Problem (DFJSP) by proposing a distributed, event-driven reinforcement learning (RL) framework that enables real-time, multi-objective decision-making. The adaptive policies improve throughput, reduce delays, and enhance system resilience, demonstrating reinforcement learning's potential as a foundation for next-generation industrial scheduling.

This thesis investigates three increasingly complex scheduling scenarios, each reflecting key challenges faced in dynamic manufacturing environments. The first scenario addresses a baseline problem of decentralized scheduling. In this problem, multiple work centers operate independently. And each work center must select job and machine priority rules without global coordination. This setting captures the reality of distributed decision-making in modern production systems, where rapid local responses are crucial. A Double Deep Q-Network (DDQN)-based agent is employed, achieving the lowest mean tardiness and the highest win rate, outperforming Short Processing Time (SPT) with statistical significance.

The second scenario introduces machine breakdowns, representing the uncertainty and disruptions inherent to physical manufacturing systems. Such breakdowns not only increase decision complexity but also require adaptive scheduling strategies that can reallocate resources in real time. Here, a Proximal Policy Optimization (PPO)-based agent with feature-weighted prioritization is used, achieving superior performance compared to PPO-RS and SPT.

The third scenario expands the problem to a multi-objective setting with sequence-dependent setup times, reflecting real-world trade-offs between performance metrics such as tardiness and changeover efficiency. A Universal Value Function Approximators (UVFA)-enhanced DDQN agent is applied to learn across different reward preferences, with the baseline-referenced reward (Set 4) achieving the best Pareto front.

SHapley Additive exPlanations (SHAP) analysis shows that the agent adaptively shifts attention across features based on the reward structure, prioritizing due-date and waiting-time features under tardiness objectives, and setup-related features when the goal is to minimize setup impact. The number of waiting jobs consistently remains one of the most influential indicators across all settings.

Overall, this thesis contributes a robust and generalizable RL-based scheduling architecture that effectively adapts to real-time disturbances and multi-objective trade-offs. By integrating distributed control, event-driven decision mechanisms, and interpretable learning, the proposed frameworks pave the way for scalable, intelligent scheduling systems for next-generation smart manufacturing.

Acknowledgements

I would like to express my sincere gratitude to my current supervisor, Dr Yang, for his guidance and support during the final year of my doctoral research. His constructive feedback, insightful advice, and encouragement have been invaluable in bringing this thesis to completion.

I would like to thank the School of Engineering at the University of Glasgow for providing an excellent research environment, outstanding facilities, and consistent administrative support.

My heartfelt thanks go to my peers and friends, especially my fellow PhD colleagues in the 74 Oakfield Avenue office, for their encouragement, stimulating discussions, and the camaraderie that made my doctoral journey both productive and enjoyable. Finally, I am deeply indebted to my family for their unwavering love, patience, and belief in me; without their support, this work would not have been possible.

Table of Contents

List of Figures

List of Tables

Abbreviations

2D-FND	2D-Folded Normal Distribution
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
ACO	Ant Colony Optimization
AI	Artificial Intelligence
ATC	Apparent Tardiness Cost
CEXSPT	Cumulative Expected Shortest Processing Time
CNC	Computer Numerical Control
COVERT	Cost Over Time
CPS	Cyber-Physical System
CPU	Central Processing Unit
CR	Critical Ratio
CRsetup	Critical Ratio with Setup Consideration
DAG	Directed Acyclic Graph
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Network
Dec-POMDP	Decentralized Partially Observable Markov Decision Process
DFJSP	Dynamic Flexible Job Shop Problem
DJS	Dynamic Job-Shop Scheduling
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EDD	Earliest Due Date
ERP	Enterprise Resource Planning
FIFO	First In, First Out
FJS	Flexible Job-Shop
FJSSP	Flexible Job-Shop Scheduling Problem
FMS	Flexible Manufacturing System
FSP	Flow-Shop Scheduling Problem
GA	Genetic Algorithm
GAE	Generalized Advantage Estimation
GD	Generational Distance
GP	Genetic Programming
IGD	Inverted Generational Distance
IoT	Internet of Things
JSP	Job-Shop Scheduling Problem
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
MES	Manufacturing Execution System

MF	Most Work Remaining
MILP	Mixed-Integer Linear Programming
MLP	Multi-Layer Perceptron
MOD	Modified Operation Due Date
MOOP	Multi-Objective Optimization Problems
MRP	Material Requirements Planning
MSE	Mean Squared Error
MTBF	Mean Time Between Failures
MTTR	Mean Time to Repair
NP-hard	Non-Deterministic Polynomial-Time Hard
PPO	Proximal Policy Optimization
PSO	Particle Swarm Optimization
RAM	Random Access Memory
RL	Reinforcement Learning
SARSA	State–Action–Reward–State–Action
SHAP	SHapley Additive exPlanations
SI	Slack per Operation
SLACK	Slack Time
SPT	Shortest Processing Time
SST	Shortest Setup Time
SSAPT	Shortest Setup Adjusted Processing Time
TD	Temporal Difference
TRPO	Trust Region Policy Optimization
TWC	Total Work Content
UVFA	Universal Value Function Approximators
VLSI	Very-Large-Scale Integration
WIP	Work-in-Progress

Chapter 1. Introduction

In the era of Industry 4.0 and the emerging Industry 5.0 paradigm, collectively known as smart manufacturing, the integration of Cyber-Physical Systems (CPS) and the Internet of Things (IoT) connects equipment, sensors, and products throughout the shop floor. This seamless connectivity enables real-time data acquisition, bidirectional communication, and continuous feedback, thereby supporting intelligent, sequential, and data-driven decision-making across all stages of manufacturing operations [1].

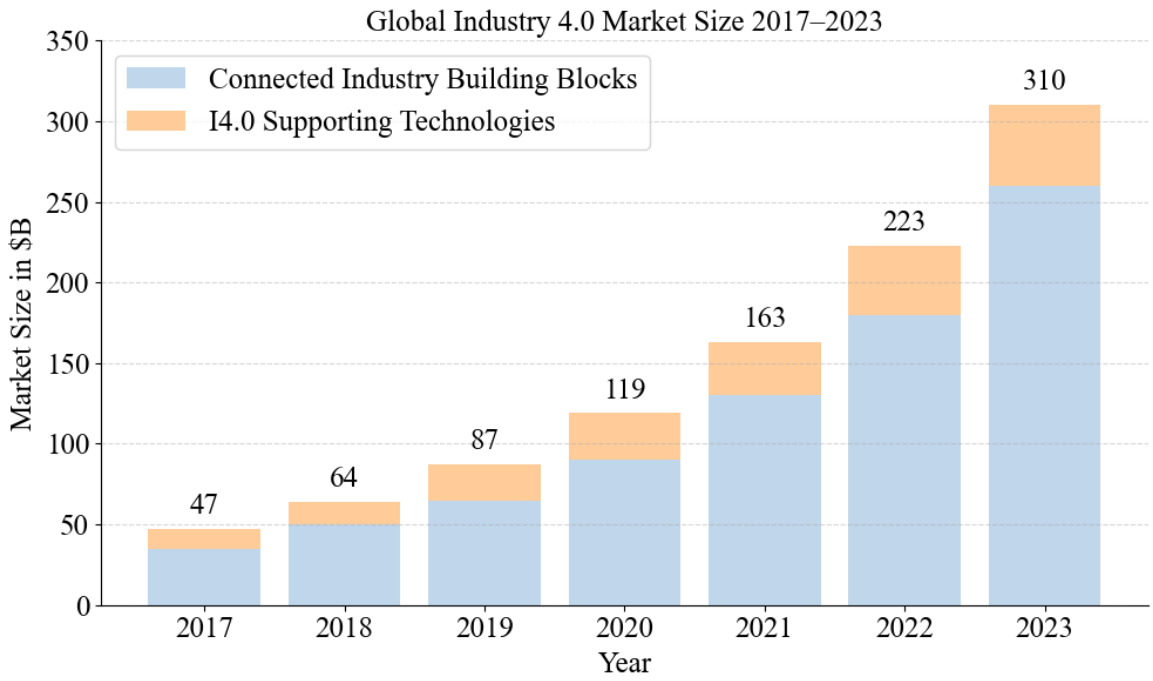


Figure 1.1. Global Industry 4.0 market size from 2017 to 2023. Data source: IoT Analytics [2].

The rapid adoption of these technologies has significantly expanded the global Industry 4.0 market. As shown in **Figure 1.1**, the market is segmented into Connected Industry Building Blocks (e.g., hardware, connectivity, analytics, cybersecurity) and I4.0 Supporting Technologies (e.g., additive manufacturing, AR/VR, collaborative robots, drones, and

SDVs), with the overall size growing from $47 billion to $310 billion over six years (CAGR: 37%). The overall market size increased from $47 billion in 2017 to $310 billion in 2023, reflecting a compound annual growth rate (CAGR) of approximately 37%. The market is composed of two major subsets: *Connected Industry Building Blocks*, which include core domains such as hardware, connectivity, cloud analytics, cybersecurity, and system integration; and *I4.0 Supporting Technologies*, encompassing emerging innovations such as 3D printing, AR/VR, collaborative robots, machine vision, drones, and autonomous vehicles.

This rapid growth highlights the increasing industrial emphasis on flexibility, automation, and responsiveness—the core objectives of Industry 4.0. Meanwhile, Industry 5.0 extends this paradigm by introducing a human-centric approach that enhances the synergy between human creativity and artificial intelligence. This collaborative model aims to deliver highly personalized products at scale while preserving efficiency, sustainability, and social well-being in future manufacturing ecosystems [3].

Despite these advances, production scheduling remains a critical operational bottleneck in smart factories. Modern manufacturing is characterized by real-time data streams, dynamic job arrivals, machine unreliability, and personalized product mixes. These features introduce unprecedented levels of uncertainty, which render traditional scheduling methods obsolete. While reinforcement learning (RL) has shown promise for adaptive decision-making, existing RL-based schedulers are often impractical: they require global system state (infeasible in large-scale shops), operate as black boxes (undermining human trust in Industry 5.0), and cannot flexibly balance competing objectives like tardiness and setup cost without retraining.

This thesis addresses this gap by developing a decentralized, interpretable, and multi-objective reinforcement learning framework that operates under local observability. The approach enables each work center to make transparent and adaptive scheduling decisions in dynamic, real-world manufacturing environments.

1.1. Recent Developments in Smart Manufacturing

The rise of personalized manufacturing is a critical response to customer demand for unique and customized products, moving away from traditional mass production to customized production. For example, BMW's MINI Cooper offers extensive customization options, allowing customers to personalize their cars, thereby enhancing customer satisfaction and loyalty [4]. However, this personalization strategy significantly increases complexity and leads to frequent production changes, which raises setup times and production costs. Similarly, the Very Large-Scale Integration (VLSI) circuit production faces complicated scheduling issues due to re-entrant flows, unpredictable job arrivals, high capital costs, batch operations, sequence-dependent setups, and significant automation [5]. These complexities underscore the need for flexible and dynamic scheduling methods that can rapidly adjust to frequent changes while maintaining profitability and efficiency.

These challenges reflect a fundamental shift in production logic—from predictable and linear to adaptive and nonlinear. As the demand for customization and real-time responsiveness intensifies, traditional static scheduling systems, which rely on predefined sequences and centralized decisions, become inadequate. This motivates the development of dynamic, distributed, and intelligent scheduling frameworks that can adjust decisions on the fly in response to disturbances and demand shifts. Among the promising solutions, RL stands out for its ability to learn from interaction with the environment, enabling adaptive scheduling strategies that balance multiple competing objectives in complex shop-floor environments.

1.2. Problem Statement

Scheduling is a fundamental decision-making process widely applied in many manufacturing and service industries. It deals with the allocation of resources to tasks over

given periods, and its goal is to optimize one or more objectives [6]. Traditional static scheduling methods assume a predictable and stable production environment, making them insufficient when faced with unexpected disruptions such as sudden job arrivals or machine breakdowns. Such disruptions often lead to delays, inefficient resource utilization, and increased costs, necessitating more adaptive and responsive scheduling methods.

Dynamic scheduling more accurately reflects real-world manufacturing due to its capability for real-time responsiveness. In particular, the DFJSP captures key complexities including flexible routing, dynamic arrivals, and resource uncertainty. However, three interrelated challenges persist:

- **Scalability under decentralization:** Centralized RL approaches require full system observability, which is impractical in large-scale shops.
- **Robustness under uncertainty:** Machine failures and stochastic events demand policies that adapt without catastrophic performance drops.
- **Multi-objective trade-offs:** Objectives like minimizing tardiness and reducing setup costs are often conflicting, yet most RL schedulers optimize a single fixed reward.

To address these challenges in a structured manner, this thesis investigates the DFJSP across three progressively realistic problem settings:

- **Problem 1:** Dynamic job arrivals (baseline dynamism),
- **Problem 2:** Adding stochastic machine breakdowns (resource unreliability),
- **Problem 3:** Adding sequence-dependent setup times and explicit multi-objective optimization (operational realism).

1.3. Research Objectives

Achieving flexibility, automation, and real-time responsiveness requires dynamic scheduling

methods that are adaptive, distributed, and robust. Such methods must address key challenges related to scalability and adaptability within complex production environments. Therefore, this research explores the development of dynamic scheduling techniques designed to efficiently manage multiple dynamic scenarios commonly encountered in manufacturing, including unexpected job arrivals, machine breakdowns, and variability in setup times. This research targets the following objectives:

- To design a decentralized and event-driven scheduling architecture that enables distributed decision-making at the work center level, supporting responsiveness to real-time events such as job arrivals and machine availability.
- To develop an RL-based continuous prioritization mechanism that overcomes the limitations of fixed rule selection by dynamically generating feature-based priority scores for sequencing decisions.
- To extend the scheduling framework to support multi-objective optimization, particularly in scenarios involving sequence-dependent setup times, by allowing flexible trade-offs between competing goals such as tardiness and setup cost.
- To ensure interpretability and transparency of the learned policies, explainable AI techniques are used to uncover how the RL agent adapts under different conditions and objective preferences.

1.4. Main Contributions

This study specifically focuses on the production scheduling layer, excluding components related to supply chain coordination or process design. By maintaining this scope, the research investigates mechanisms and strategies that support effective operational decision-making and improve scheduling performance under dynamic and uncertain conditions.

The core contributions of this thesis lie in the development and progressive enhancement of a distributed, event-driven reinforcement learning framework for solving the DFJSP under

Industry 4.0/5.0 conditions. The contributions evolve across three increasingly complex problems, resulting in a generalizable, interpretable, and multi-objective scheduling system.

The primary contributions are:

- **A distributed, event-driven RL architecture for dynamic scheduling under random job arrivals**

Addressing **Problem 1** (**Chapter 4)**, this contribution introduces a decentralized framework where each work center independently triggers scheduling decisions upon local events (e.g., job arrival, machine idle). Using DDQN trained on locally observable state features, the system achieves scalable, real-time responsiveness without global coordination.

- **A feature-weighted continuous prioritization mechanism for robust scheduling under machine failures**

Addressing **Problem 2** (**Chapter 5**), this work replaces discrete rule selection with a PPO-based policy that dynamically computes priority scores from weighted job features. This enables fine-grained, adaptive sequencing under stochastic disruptions while supporting post-hoc interpretability through feature attribution.

- **A Universal Value Function Approximators (UVFA)-enhanced multi-objective RL framework for balancing tardiness and setup cost with sequence-dependent constraints**

Addressing **Problem 3** (**Chapter 6**), this contribution integrates UVFA into the RL agent, allowing it to generalize across arbitrary linear combinations of competing objectives. A baseline-referenced reward design further stabilizes learning and promotes Pareto-efficient solutions without retraining.

- **Cross-problem interpretability analysis via SHapley Additive exPlanations**

Spanning **Chapters 4–6**, this contribution applies SHAP to decode how learned policies

prioritize state features under different uncertainties and objective weights. This provides actionable insights into agent behavior and bridges the gap between black-box RL and human-understandable scheduling logic.

A modular, open-source simulation environment for the DFJSP under dynamic events (job arrivals, machine breakdowns, sequence-dependent setups) is developed in Python, facilitating reproducibility and extension by the research community.

1.5. Outline of the Thesis

This thesis is structured into seven chapters, each contributing to a comprehensive investigation into distributed, event-driven reinforcement learning for dynamic scheduling problems in manufacturing systems. The content and purpose of each chapter are as follows:

Chapter 1: Introduction

This chapter introduces the background and motivation for addressing dynamic scheduling problems in modern manufacturing systems. It outlines the research objectives, presents the main contributions, and describes the structure of the thesis.

Chapter 2: Literature Review

This chapter reviews existing studies on dynamic scheduling, focusing on key classifications such as event triggers, uncertainty types, decision-making structures, and solution methods. The chapter identifies research gaps in scalability, interpretability, and multi-objective handling with reinforcement learning.

Chapter 3: Formulation of the Dynamic Scheduling Problem

This chapter formulates the dynamic flexible job shop scheduling problem across three

increasingly complex scenarios. It begins with random job arrivals as the baseline dynamic setting (**Problem 1**). The second stage incorporates machine breakdowns to account for resource uncertainty (**Problem 2**). Finally, it introduces multi-objective scheduling with sequence-dependent setups to reflect real-world trade-offs between performance metrics (**Problem 3)**. Each scenario is formally modeled, laying a unified foundation for the learning-based methods developed in later chapters.

Chapter 4: Problem 1 – Baseline Dynamic Scheduling under Random Job Arrivals with RL

This chapter addresses the challenge of dynamic job arrivals in a flexible job shop environment, where traditional rules struggle to maintain performance under uncertainty. To solve this, a decentralized event-driven scheduling framework is developed based on DDQN, where each work center independently selects job and machine priority rules. The proposed approach significantly improves tardiness performance over conventional dispatching rules, demonstrating the effectiveness of RL in handling real-time job flow.

Chapter 5: Problem 2 – Scheduling under Machine Reliability Constraints with RL

Building on **Chapter 4**, this chapter adds stochastic machine breakdowns, which are a second layer of uncertainty that disrupts both resource availability and schedule continuity. To handle this, the discrete rule-selection approach is replaced with a PPO-based continuous prioritization mechanism that dynamically weights job features (e.g., remaining processing time, due date urgency) to generate fine-grained sequencing decisions. This represents a fundamental shift from choosing rules to constructing priorities, improving robustness, and enabling post-hoc interpretability via feature attribution.

Chapter 6: Problem 3 – Multi-Objective Scheduling with Sequence-Dependent Setup Times with RL

Extending **Chapter 5,** this chapter incorporates sequence-dependent setup times and explicit trade-offs between tardiness and setup cost. This adds both operational constraints and competing objectives to the scheduling problem. The RL agent is enhanced with UVFA, allowing a single policy to generalize across arbitrary linear combinations of objectives without retraining. The evaluation now shifts from single-metric performance to Pareto efficiency and solution diversity, reflecting real-world needs for flexible preference adaptation.

Chapter 7: Conclusion and Future Work

This chapter summarizes the key findings, theoretical and practical implications, and limitations of the research. It concludes with suggestions for future work, including expanding the framework to more complex real-world applications.

Chapter 2. Literature Review

Scheduling plays a critical role in manufacturing systems, especially under the dynamic and uncertain conditions of Industry 4.0. The Dynamic Flexible Job Shop Problem (DFJSP) captures this complexity, where job arrivals are unpredictable, machine breakdowns are possible, and production conditions evolve in real time. Traditional static scheduling methods, which rely on fixed job information and stable machine availability, often fail to cope with such variability.

In response, researchers have developed more adaptive strategies, including heuristic, metaheuristic, and RL-based methods. RL agents, in particular, can learn effective scheduling policies by interacting with the environment and making real-time, data-driven decisions—enhancing responsiveness, efficiency, and resource utilization.

The chapter is structured as follows. **Section 2.1** introduces key scheduling concepts, constraints, and performance objectives. **Section 2.2** presents a conceptual framework of scheduling environments and compares their complexity. **Section 2.3** highlights the dynamic and uncertain nature of modern manufacturing scheduling. **Section 2.4** reviews traditional solution methods, including exact, heuristic, and metaheuristic algorithms. **Section 2.5** focuses on RL for dynamic scheduling, including value-based and actor-critic methods. **Section 2.6** discusses multi-objective scheduling problems and trade-off management using Pareto theory. **Section 2.7** introduces SHAP (SHapley Additive exPlanations) as an explainability technique in RL and outlines its use as an empirical tool to interpret learned scheduling policies. **Section 2.8** identifies current challenges and research gaps. Finally, **Section 2.9** summarizes the review and connects it to the research direction of this thesis.

2.1. Theoretical Foundations

2.1.1. Fundamental Scheduling Concepts and Terminology

Scheduling is a critical decision-making process in manufacturing systems, involving allocating resources (machines) to tasks (jobs) over specified time periods, to optimize one or multiple objectives [7]. Scheduling plays a crucial role in improving production efficiency and effectiveness across different industrial engineering contexts. Resources can vary widely from airport runways [8], [9], [10], port cranes [11], [12], and computer processing units [13] to machines in manufacturing systems. Tasks also vary significantly, such as aircraft take-offs and landings [14], cargo loading on ships [15], job executions in manufacturing systems, or task scheduling in computer processors and cloud computing environments.

Scheduling problems in manufacturing are not only defined by their objectives but also shaped by a variety of constraints. These constraints directly affect the feasibility and performance of a schedule, especially in dynamic environments. Typical restrictions and constraints in the scheduling problem are:

- **Release dates:** Times when jobs become available for processing. Release dates prevent the processing of the jobs from being available.
- **Preemptions:** Imply that it is not necessary to keep a job on a machine once started until its completion. Preemptions are important in real-time environments, while urgent tasks must be prioritized.
- **Precedence constraints:** Require that one or more jobs may have to be completed before another job is allowed to start its processing. These constraints reflect the complexity of a multi-stage production environment, such as the automobile or steel furnace industry.
- **Sequence-dependent setup times:** Additional preparation time is required between different jobs, varying based on the order of the tasks. These types of setup times reflect

the industry with frequent product changes, such as pharmaceuticals and food processing, especially for mass customized production, where the setup duration variation can lead to significant production inefficiencies.

- **Batch processing:** Indicates that a machine may be able to process several jobs. Batch processing is common in chemical processing or the semiconductor industry, which adds complexity due to the batch sizes and compatibility.
- **Breakdowns:** Imply that a machine may not be continuously available. The robust and flexible scheduling strategies are required to maintain efficiency and minimize downtime.
- **Recirculation:** A job may visit a machine or work center more than once. Recirculation is considered in two different situations; one is in the complex manufacturing operations where multiple processing stages are considered. Another is that the failure rate is considered in processing. Both circumstances increase scheduling complexity.

Some common possible objective functions in the manufacturing problems are:

- **Makespan:** defined as the completion time of the last job to leave the system. Shorter makespans are associated with increased efficiency and reduced costs.
- **Total weighted tardiness**: represents the total waited delay relative to due dates. Excessive tardiness leads directly to financial penalties, lost customer goodwill, and damaged market reputation.
- **Flowtime:** represents the total time jobs spend in production. Minimizing flowtime is important for reducing inventory costs, improving responsiveness, and enhancing customer service.
- **Total setup time:** defined as the cumulative time spent preparing machines for different jobs. Reducing the setup time could improve the production efficiency and cut down the tooling cost, while frequently changing the tools would affect the health of the machines.
- **Machine utilization:** defined as the percentage of total available time during which

machines are actively processing jobs. High utilization rates signify optimal use of resources, though excessively high rates may risk equipment fatigue and failures.

Real-world scheduling problems often involve conflicting objectives, such as minimizing makespan, cost, energy consumption, and tardiness. A clear understanding of these trade-offs lays the groundwork for developing advanced scheduling strategies that can adapt to diverse operational goals and dynamic constraints, as explored in the subsequent chapters.

2.1.2. The Role of Scheduling in Enterprise

As highlighted in **Section 1.2** of Pinedo's "Scheduling: Theory, Algorithms, and Systems (5th ed.)" [6], scheduling acts as a critical coordination mechanism that integrates various managerial and operational activities within an enterprise ecosystem. Particularly in manufacturing, scheduling interacts closely with upstream systems such as Enterprise Resource Planning (ERP) and Material Requirements Planning (MRP), aligning production activities with organizational goals. **Figure 2.1** depicts the information flow diagram in a manufacturing system.

Scheduling systems receive key data, including customer orders, due dates, production quantities, and capacity forecasts, from ERP and MRP modules. These inputs are translated into detailed shop-floor schedules, determining machine assignments, processing sequences, and operation timings. This process effectively bridges long-term strategic plans with day-to-day operational tasks.

Effective integration between scheduling and ERP/MRP enables the enterprise to manage disruptions such as equipment failures, material shortages, or shifts in customer demand, maintaining responsiveness and agility. Simultaneously, scheduling provides operational instructions to shop-floor control systems, facilitating real-time execution and continuous performance monitoring. Thus, scheduling acts as an essential link connecting long-term

strategic planning and immediate operational decision-making.

Industry 4.0 and digital transformation highlight the importance of real-time data exchange and rapid responsiveness. Traditional ERP-based scheduling systems struggle to adapt quickly enough to highly dynamic manufacturing environments. Modern cyber-physical systems and smart factories require advanced scheduling methods capable of quickly adapting to changing conditions. Emerging decentralized scheduling approaches based on reinforcement learning are increasingly explored to address these challenges [16].

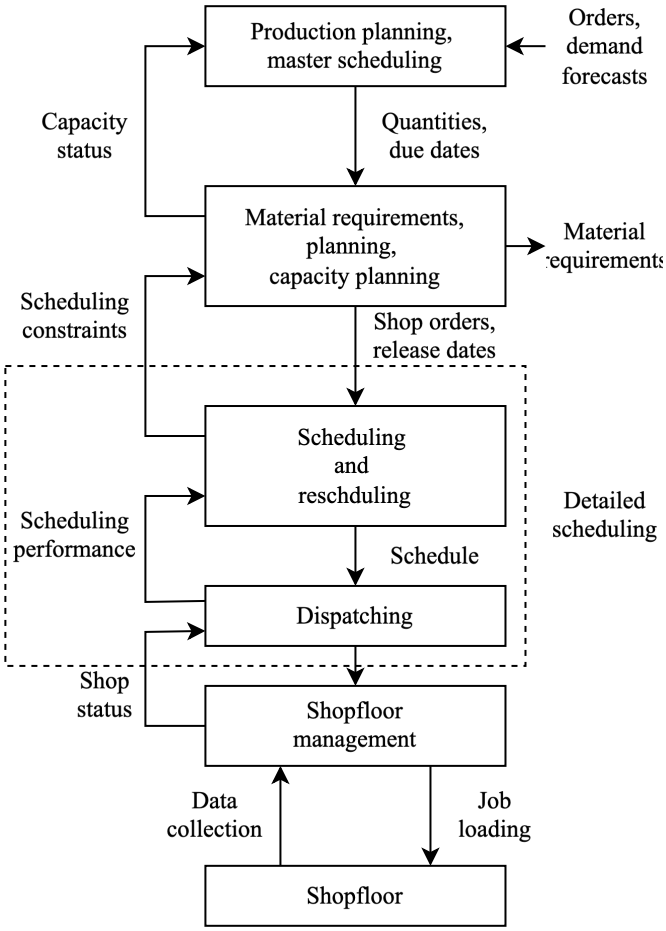


Figure 2.1. The information flow diagram in a manufacturing system.

The scheduling approach presented in this research aims to enhance interoperability between strategic planning functions (ERP/MRP) and operational execution on the shop floor. This research leverages decentralized and event-driven approaches to enhance decision quality,

system responsiveness, and operational performance. Accordingly, the thesis highlights scheduling as a vital integrative mechanism for aligning strategic goals with operational efficiency in modern manufacturing.

2.2. A Conceptual Framework of Scheduling Problems

The conceptual framework proposed in this section is systematically derived from a critical synthesis of foundational and contemporary literature in dynamic scheduling. The framework is organized around four core dimensions: triggering mechanisms, environment uncertainties, decision architectures, and solution methods. Each of these reflects a structuring principle that has been consistently emphasized in the literature over decades.

Specifically, the distinction in triggering mechanisms, such as periodic versus event-driven rescheduling, builds on the seminal framework by Vieira, Herrmann, and Lin [17], who classified reactive scheduling policies based on disturbance types. This perspective was later echoed in surveys including Ouelhadj and Petrovic [18]. Empirical studies by Sabuncuoğlu and Bayiz [19] further validated these rescheduling strategies under conditions such as machine breakdowns and unexpected job arrivals.

The treatment of environmental uncertainties is grounded in the comprehensive uncertainties established by Ouelhadj and Petrovic [18], including machine failures, stochastic job arrivals, and sequence-dependent setup times. This view has been extended by Zandieh and Gholami [20] and Sotskov and Werner [21], and complemented by Aytug et al.'s systematic review of approaches to handling uncertainty in production execution [22].

The axis of decision architecture, which distinguishes centralized from decentralized control, draws directly from research on multi-agent and holonic manufacturing systems. Key contributions include the work of Shen, Norrie, and Barthès [23], as synthesized in Kletti [24], and Aissani et al. [25], both of which contrast global optimization with local autonomy

in smart factory environments.

Finally, the evolution of solution methods, from simple dispatching rules to advanced deep reinforcement learning techniques, is traced through key milestones. These include Pan and Yang [26] on transfer learning, Wang et al. [27] on curriculum learning, and recent applications to the dynamic flexible job shop scheduling problem by Liu et al. [28], Zhang et al. [16], and Luo et al. [29].

This framework integrates insights from Pinedo's authoritative textbooks [6], [7] and more than 120 peer-reviewed studies published between 2000 and 2024, all drawn from the reference corpus of this thesis. To validate its coherence, every cited work in **Chapter 2** was mapped onto the framework, confirming that each study fits unambiguously within one or more of its dimensions. Furthermore, the framework aligns with recent survey articles, including Shojaeinasab et al. [30] on intelligent manufacturing execution systems and Tian et al. [31] on dynamic multi-objective learning, which underscores its relevance to current research trends.

Figure 2.2 visually synthesizes this conceptual framework, presenting the four dimensions as hierarchical axes that jointly define the space of dynamic scheduling problems. Each branch corresponds to a category discussed above, enabling researchers to locate any study or design any new approach within a unified classification system. The diagram thus serves not only as a summary of existing knowledge but also as a structured guide for future investigation.

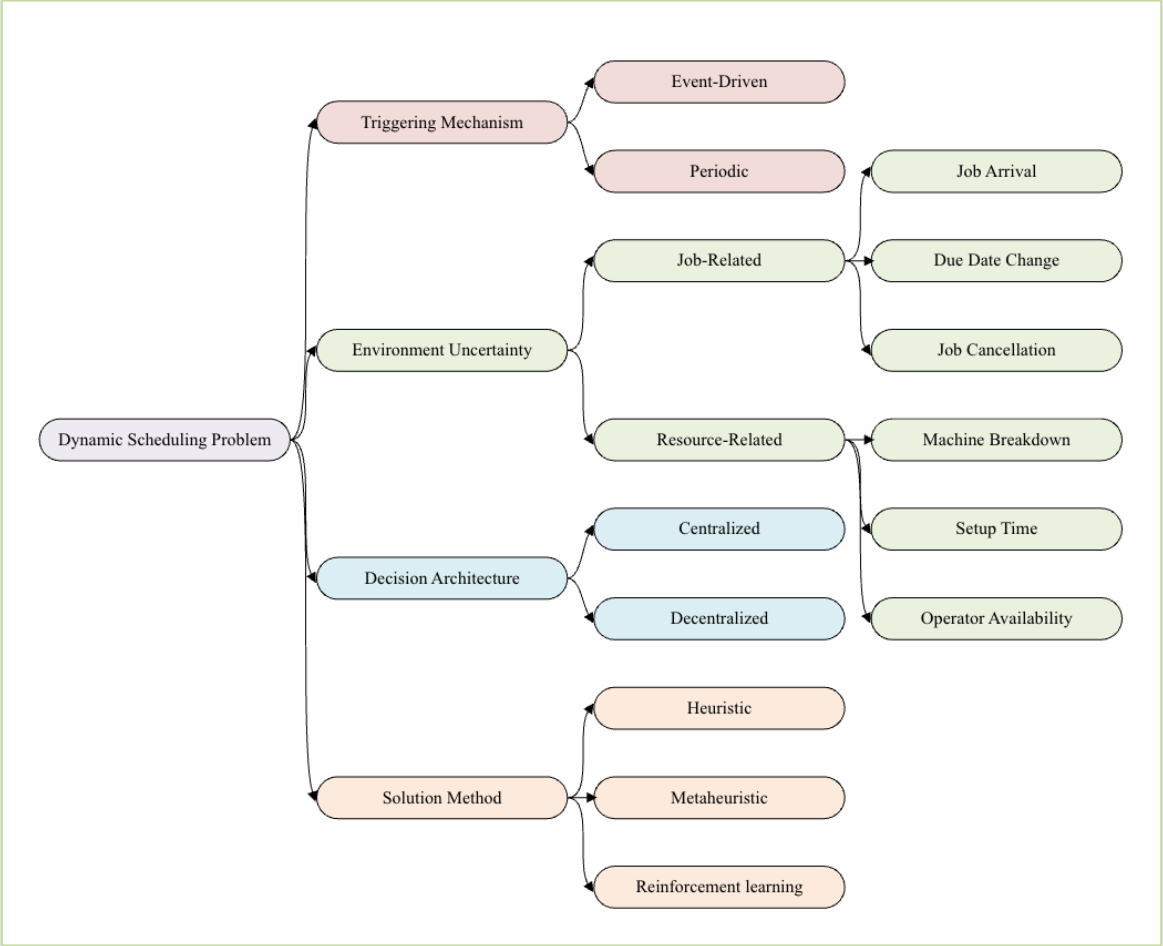


Figure 2.2. Conceptual framework of dynamic scheduling problem components.

2.2.1. Machine Environment Classifications

Based on the machine environment, the manufacturing problem can be categorized basically as the single-machine scheduling problem, the parallel-machine problem, the flow-shop scheduling problem, the job-shop scheduling problem, and the open-shop scheduling problem.

- **Single-machine scheduling problem** is the simplest form, involving sequential processing on a single resource. It is often applied in specialized manufacturing, like custom jewelry or precision watchmaking.
- **Parallel-machine scheduling problem** extends this by enabling simultaneous processing across multiple resources. It is commonly observed in printing industries or

batch processing in food industries.

- **Flow-shop scheduling problem (FSP)** is further generalized by imposing a fixed machine sequence common to all jobs, introducing constraints on processing order. A typical example is an automobile assembly line, where all the production follows the same processing route.
- **Job-shop scheduling problem (JSSP)** expands complexity by permitting distinct routing paths for each job. Some common real-world applications include aerospace part fabrication, custom mold manufacturing, and semiconductor wafer production, while custom production often includes increasing the flexibility of this scheduling problem.
- **Open-shop scheduling problem** embodies the highest level of generality by removing routing restrictions altogether, significantly enlarging the feasible solution space and intensifying the complexity inherent in the scheduling problem. Though this problem is theoretically intriguing, it is rarely practiced in real manufacturing due to its demand for machines capable of multiple functions.

Among these categories, the job-shop scheduling problem has garnered substantial attention since Johnson's seminal work in 1954, primarily because of its direct alignment with real-world manufacturing complexities [32]. The flexible job-shop scheduling problem, an extension of JSSP allowing operations on multiple alternative machines, further enhances practical applicability, finding significant use in advanced manufacturing sectors such as semiconductor fabrication, chemical production, pharmaceutical manufacturing, and precision mold industries [33].

2.2.2. Complexity of Scheduling Problems

The complexity of scheduling problems increases with system flexibility. While single-machine problems can be polynomially solvable under certain conditions, open-shop problems are generally NP-hard, meaning no known polynomial-time algorithm guarantees optimality. This has led to the development of advanced heuristic and metaheuristic

approaches, such as genetic algorithms, simulated annealing, tabu search, and hybrid methods, to obtain practically feasible solutions.

Table 2.1. Comparison of scheduling problem types by complexity, applicability, and Industry 4.0 relevance.

Problem type	Complexity	Applicability	Industry 4.0 Relevance
Single machine	**Low** (Polynomial-time solutions for many cases)	Basic manufacturing, small-scale workshops, and low-volume production.	Low: Limited flexibility and scalability; incompatible with smart factories requiring parallel workflows.
Parallel-Machine	**Moderate** (NP-hard for many variants)	High-volume production (e.g., electronics assembly), batch processing.	Moderate: Enables scalability but lacks adaptability to real-time disruptions or complex workflows.
Flow-Shop (FSP)	**Moderate-High** (NP-hard for >2 machines)	Assembly lines (e.g., automotive, consumer goods), sequential production stages.	Moderate: Fixed sequences limit flexibility, but IoT integration can enhance predictability.
Job-Shop (JSSP)	**High** (Strongly NP-hard)	Customized manufacturing (e.g., aerospace, semiconductors), complex routing with unique job sequences.	High: Aligns with flexible manufacturing systems (FMS) and smart factories needing adaptive routing.
Open-Shop	**Very High** (NP-hard)	Rare in traditional manufacturing (e.g., maintenance scheduling, lab experiments).	Emerging Potential: Maximizes flexibility (no fixed routing), ideal for reconfigurable smart factories.

2.2.3. Flexible Job Shop Scheduling Problem

A notable extension of JSSP is the FJSSP, which integrates features from JSSP and parallel-machine scheduling. A flexible job shop (FJS) consists of multiple work centers. Each work

center contains several parallel machines that can process different operations independently. Jobs are categorized into several types, and each job type is composed of a sequence of operations. These operations must follow a predefined order and can be processed on any capable machine within the assigned work center. FJSSP generalizes JSSP by allowing routing flexibility and sequencing flexibility. Routing flexibility allows jobs to choose from multiple available machines within a work center, significantly enhancing adaptability and responsiveness. Sequencing flexibility allows jobs to follow varied operation sequences across different machines, improving customization and efficiency. These flexibilities considerably increase both the practical relevance and complexity of the FJSSP.

DFJSSP has become increasingly relevant with the advancement of flexible manufacturing systems (FMS), which align with Industry 4.0 objectives of increased adaptability, efficiency, and responsiveness to market changes. These modern manufacturing systems heavily rely on advanced optimization algorithms, simulation techniques, and real-time decision-making tools to effectively address this complex problem.

2.2.4. Centralized vs. Decentralized Scheduling

Centralized Scheduling involves a single global scheduler responsible for generating schedules for the entire system, where decisions flow from the top down. This hierarchical approach simplifies initial management but limits the autonomy and adaptability of local controllers, potentially causing delays in responding to dynamic changes and reducing scalability.

Decentralized Scheduling decomposes the overall scheduling problem into smaller subproblems handled independently by local decision-making agents. These agents may have conflicting local objectives but coordinate their decisions through communication mechanisms to achieve global system efficiency. Decentralized scheduling enables swift, localized decision-making based on up-to-date system information, enhancing adaptability

and responsiveness to dynamic events, such as unexpected job arrivals or machine breakdowns.

The increasing significance of decentralized scheduling correlates with advancements in modern manufacturing environments, such as Industry 4.0 and smart factories [24]. These environments prioritize adaptability, responsiveness, and robustness, making decentralized scheduling more suitable despite the requirement for sophisticated coordination mechanisms and carefully designed local agents.

2.2.5. Evolution of Scheduling Methodologies

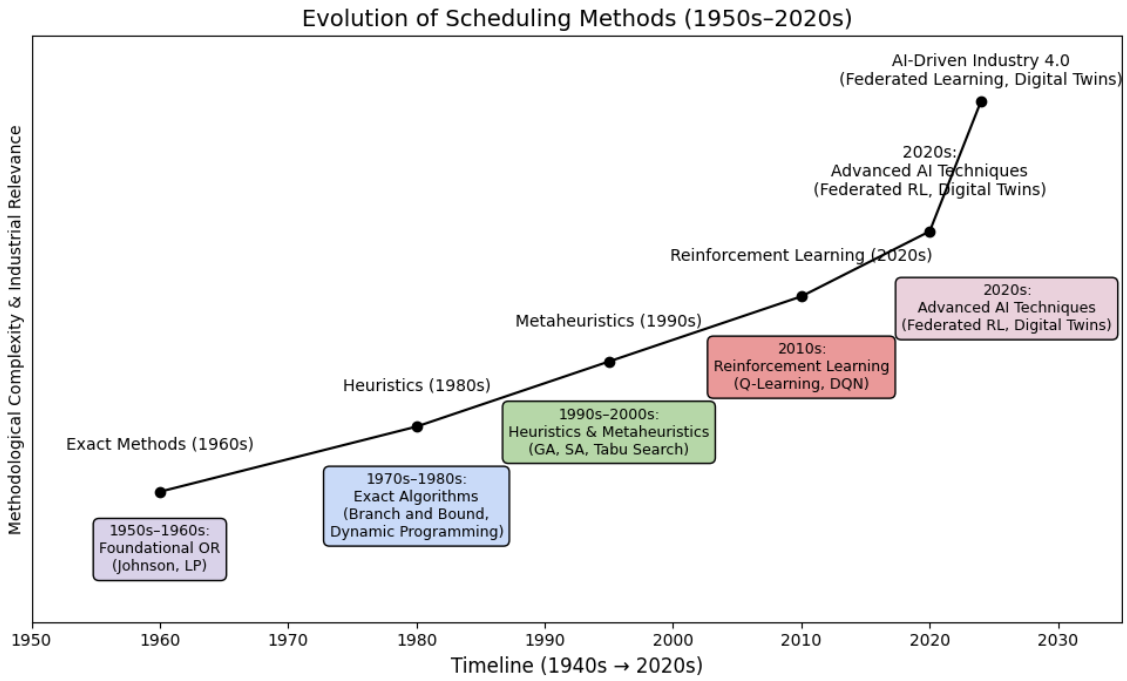


Figure 2.3. Chronological development of scheduling methodologies: from exact algorithms to AI-driven frameworks (1950s–2020s).

Figure 2.3 illustrates the evolution of scheduling methodologies from the foundational methods of Operations Research (1950s) to modern AI-driven decentralized scheduling techniques aligned with Industry 4.0. This historical progression underscores the increasing methodological complexity and industrial relevance, highlighting the transition from exact

methods and heuristics towards reinforcement learning-based approaches, including decentralized RL and federated learning. Such advanced methods reflect the contemporary demands for real-time responsiveness, adaptability, and decentralized decision-making capabilities essential for modern smart manufacturing environments.

2.3. The Dynamic Nature of Modern Manufacturing Scheduling

With the development of information technology, especially cyber-physical systems and the IoT, real-time event collection has become possible. Meanwhile, the improvement in computing power enables real-time processing of dynamic events, thereby promoting the advancement of modern industrial scheduling technologies. However, those dynamic events challenge the classical scheduling system. Modern scheduling systems emphasize real-time responsiveness, flexibility, and automation. Considering that these events add complexity to the scheduling problem, developing effective scheduling strategies has become increasingly important.

2.3.1. Static vs. Dynamic Scheduling: A Comparative Framework

Scheduling problems are often divided into static and dynamic categories due to varying levels of uncertainty and information availability inherent in different manufacturing environments:

- **Static Scheduling:** Assumes deterministic conditions where all job details and machine availabilities are known in advance, enabling fixed schedules without further modification during execution.
- **Dynamic Scheduling:** Explicitly addresses uncertainties such as unpredictable job arrivals, disruptions, or machine breakdowns, requiring real-time decision-making and continuous schedule adaptation. Dynamic scheduling methodologies leverage advanced computational methods like heuristics, machine learning, and reinforcement learning.

In most real-world environments, scheduling is a continuous and reactive process driven by frequent, unforeseen disruptions. These disruptions demand ongoing adjustments, as static approaches often fail to remain practical when exposed to real-time changes. As a result, dynamic scheduling methods are essential for ensuring robustness and sustaining optimal efficiency in modern manufacturing systems.

2.3.2. Dynamic Scheduling Policies

In a dynamic scheduling problem, decision-making strategies can be categorized by their triggering mechanisms. Rather than focusing on how decisions are made, this level considers when scheduling actions are initiated, either through event-driven triggers such as job arrivals or machine availability, or through periodic triggers at fixed time intervals.

Event-Driven Policy

In event-driven policy, the decision-making process is triggered in response to events that will change the status of the dynamic scheduling problem. Events can be either unexpected, such as job arrivals or machine breakdowns, or predictable, such as a machine becoming idle or the completion of a job's processing stage.

Shahrabi et al. [34] proposed an event-driven approach in dynamic job shop scheduling, with rescheduling triggered by new job arrivals or machine breakdowns. Similarly, Gu et al. [35] adopt an event-driven rescheduling strategy for dynamic events like job insertions, machine breakdowns, and reprocessing of unqualified operations. In a pharmaceutical factory case study, Said et al. [36] implemented online scheduling triggered by disruptions that significantly alter system states and performance. In semiconductor manufacturing, Kim et al. [37] described a two-stage scheduling process, "machine targeting" and "lot dispatching", initiated by the number of available machines and the volume of jobs in the buffer. In the context of smart factories optimizing flexible job shop problems, Chang et al. [38] used new

job arrivals as triggers for scheduling decisions. Luo et al. [29] proposed a deep reinforcement learning solution for dynamic multi-objective problems with partial-no-wait constraints, where scheduling is initiated by new job insertions or machine breakdowns.

Periodic Policy

In a periodic policy, the decision-making process is triggered at regular time intervals, which gathers all the information from the systems. This method will change the dynamic scheduling problem into a static form in decision-making. The schedule produced in the time horizon will be executed but not revised until the next time horizon.

In a chemical production scheduling scenario, Hubbs et al. [39] used reinforcement learning under a periodic receding horizon approach, with product demands generated via statistical models. This approach naturally represents system uncertainty and demonstrates superior performance compared to short receding horizon Mixed-Integer Linear Programming (MILP) schedulers. Extending forecast horizons using a shrinking horizon MILP model further enhances scheduling effectiveness. Zhu et al. [40] employed deep reinforcement learning (DRL) in cloud manufacturing, utilizing periodic updates of demand and resource matrices. Qu et al. [41] applied Q-learning for dispatching decisions in multistage batching processes, making periodic assessments at each time interval.

2.3.3. Sources of Uncertainty in Modern Manufacturing

In a dynamic manufacturing environment, disruptions can be categorized into resource-related and job-related events [18].

Resource-related events include unforeseen issues affecting machinery, tools, or materials, such as machine breakdowns, defective inputs, delayed material deliveries, and capacity constraints. Examples include supply chain disruptions due to transportation delays or geopolitical events. These events significantly impact operational flow, requiring rapid

adjustments to schedules. Common modeling techniques for these disruptions include exponential distributions for breakdown intervals and repair times [20], [42], non-homogeneous Poisson processes [43], and Gamma stochastic processes to represent machine deterioration [44]. Weibull distributions are also used to model machine failure patterns realistically, as failure probabilities typically increase over time [45], [46].

Job-related events stem from fluctuations in orders or job specifications, including unexpected orders, rush requests, last-minute priority changes, revised processing times, or additional mid-production operations. Such disruptions necessitate significant schedule adjustments to meet changing customer demands. There are three main approaches to modelling stochastic job-related events. The first approach treats events as entirely unpredictable until their occurrence. Zhang and van de Velde [47] address a non-clairvoyant online scheduling problem, where job release times, processing times, and required time lags are all unknown until execution. Both studies exemplify online scheduling models that assume events are entirely unpredictable until they occur. Hopf et al. [48] address a multistage online scheduling problem where jobs arrive sequentially and must be assigned immediately without any knowledge of future arrivals. Their work exemplifies the class of online models that assume events are entirely unpredictable until they occur. The second approach utilizes historical data and machine learning techniques for forecasting uncertainties [49], [50]. The third approach employs probability distributions, notably the Poisson process, to define event likelihood [42], [51]. Beyond the Poisson framework, various probability and uncertainty models have also been adopted to simulate dynamic behaviors in scheduling environments. Exponential and geometric distributions are commonly used to model job arrivals and processing times. For example, Thürer et al. [52] applied exponential distributions to simulate job durations in stochastic environments. Germs and van Foreest [53] employed geometric distributions in a discrete-time setup to represent stochastic inter-arrival times. Uniform distributions have been utilized for simulating dispatching rules and comparing performance under varying objectives [54]. In

addition, fuzzy logic-based models have been introduced to address ambiguity and vagueness in production information. Ali and El-Baz [55], for instance, proposed a fuzzy logic scheduling algorithm tailored for non-permutation flow shops under both static and dynamic conditions. In this research, the Poisson process is selected for modeling job-related uncertainties due to its effectiveness in capturing random arrival patterns common in dynamic manufacturing scenarios, enabling responsive scheduling strategies.

2.4. Traditional Scheduling Methodologies

To cope with the complexity and variability of modern scheduling problems, various algorithms have been developed to enhance production efficiency and effectiveness. These scheduling algorithms can generally be divided into three categories: exact algorithms, heuristics, and metaheuristics. Each category offers different trade-offs between solution quality and computational effort, and their applicability often depends on the specific characteristics of the scheduling environment.

2.4.1. Exact Algorithms: Strengths and Limitations

Exact algorithms, such as dynamic programming [56], [57], branch-and-bound [58], integer/mixed-integer programming [59], and constraint programming [60], [61], guarantee optimal solutions by exhaustively exploring the entire solution space. For instance, branch-and-bound systematically evaluates subproblems, pruning those that cannot lead to better solutions, ensuring optimality in smaller or less complex scheduling problems. Similarly, integer/mixed-integer programming provides mathematically proven optimal solutions by precisely defining and solving linear constraints and objectives.

However, these methods encounter significant challenges when dealing with larger, more complex, or highly dynamic scheduling environments. As problem size grows, the computational complexity of most practical scheduling problems increases exponentially

because they are Non-deterministic Polynomial-time hard (NP-hard), rendering exact solution methods impractical or infeasible for large-scale cases. Therefore, while exact algorithms are effective for smaller or simplified instances, their limitations in scalability and responsiveness justify the need for heuristics and metaheuristics to handle more realistic, dynamic, and complex scenarios efficiently.

2.4.2. Heuristic and Metaheuristic Approaches

A common class of heuristic algorithms adopts basic priority rules to construct schedules efficiently. Common dispatching rules used in reactive scheduling systems include SPT, FIFO, SLACK, visibility rule, EDD, CR, SI, MOD, MF, COVERT, ATC, CEXSPT, CRsetup, and others [18]. These rules consist of two key components: an attribute function, which defines the job or machine's priority based on specific criteria, and a sorting algorithm, which ranks and selects the best candidate for allocation. Scheduling decisions are made by choosing the job or machine that minimizes or maximizes the attribute value. Due to their low computational complexity, dispatching rules enable rapid decision-making, making them well-suited for real-time scheduling. However, their effectiveness can significantly decline in more complex or multi-objective scheduling scenarios, and they are sensitive to rule selection and changing system conditions. Therefore, while simple dispatching rules offer fast solutions, they may not consistently yield optimal outcomes across diverse or unpredictable environments [62].

Metaheuristics, on the other hand, are higher-level algorithms designed to navigate complex and large solution spaces, guiding searches toward high-quality solutions without ensuring optimality. Common examples include Genetic Algorithms (GA), Simulated Annealing (SA), Tabu Search (TS), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO). A brief description of selected metaheuristics is presented next.

- **Genetic Algorithms** [63], inspired by evolutionary biology, encode scheduling solutions as chromosomes. Through iterative processes involving selection, crossover, mutation, and elitism, GA progressively refines populations of solutions. While GA has advantages in handling complex combinatorial scheduling problems, it typically requires substantial computational resources and iterative evaluations, limiting its applicability to real-time adjustments [64], [65].
- **Simulated Annealing** mimics the physical annealing process, probabilistically accepting both better and worse solutions to escape local optima [66], [67], [68]. Its strength lies in finding high-quality solutions with fewer parameters than GA. However, the iterative cooling schedule required for convergence limits its responsiveness to immediate disruptions, making it less suitable for real-time scheduling.
- **Tabu Search** utilizes memory structures to systematically explore solutions and avoid cycling through recently visited options [69], [70]. Its adaptive memory approach efficiently handles complex problems, but it requires considerable computation for managing and updating the tabu list, thus reducing real-time responsiveness.
- **Ant Colony Optimization**, inspired by ant behavior in nature, constructs solutions by probabilistically following pheromone trails [71], [72]. Its iterative improvement process effectively tackles complicated optimization problems but typically demands extensive computation, making it less viable for immediate decision-making in dynamic environments.
- **Particle Swarm Optimization** simulates collective behavior to search the solution space [73], [74], [75]. It balances global and local exploration effectively but generally requires multiple iterations for convergence. Consequently, its applicability to instantaneous real-time scheduling adjustments is limited.

In summary, while metaheuristics offer robust methods for addressing complex scheduling problems, their reliance on iterative refinements and substantial computational resources makes them predominantly suitable for offline or periodic optimization tasks. In contrast,

simpler heuristic dispatching rules provide faster yet potentially suboptimal solutions, underscoring the value of more adaptive approaches, such as reinforcement learning, for dynamic scheduling scenarios.

2.5. Modern AI-Driven Scheduling Solutions

2.5.1. Summary of RL-based Scheduling Literature

RL significantly differs from classical optimization methods as it does not rely on predefined mathematical models or explicit solutions. Instead, RL learns through a trial-and-error process within an environment, incrementally improving policies by maximizing accumulated rewards from actions. With the development of machine learning methods, especially the occurrence of deep learning methods, RL has been successfully applied to complex decision-making problems, such as the game of GO [76] and robot control [77]. Considering the dynamic scheduling problem as a decision-making environment for RL, the RL agent can be trained to select the optimal policy through a large amount of simulation, especially for uncertain values such as processing time and unpredictable events.

2.5.1.1. Introduction to Reinforcement Learning

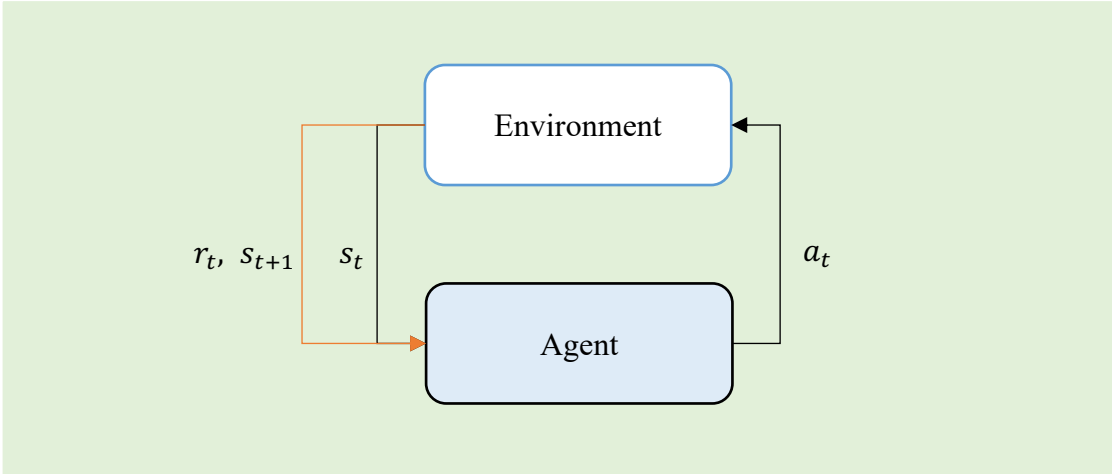


Figure 2.4. The diagram of reinforcement learning

RL is one of the three basic machine learning paradigms, alongside supervised learning and unsupervised learning. Unlike the supervised learning method, it does not need pairs of input and output data or the optimal solutions to correct the learning algorithms. By focusing on finding the balance of exploitation (policy under the learned knowledge) and exploration (using untried policy), RL selected the actions to achieve the maximum accumulated rewards. The environment setting of the RL is stated as the Markov Decision Process (MDP). An MDP is formally defined by a tuple $(\boldsymbol{S}, A, P, R, \gamma)$, where $\boldsymbol{S}$ is the state space, A the action space, $P(\boldsymbol{s}'|\boldsymbol{s}, a)$ the transition probability, $R(\boldsymbol{s}, a, \boldsymbol{s}')$ the reward function, and $\gamma \in [0,1]$ the discount factor that weighs future rewards. In scheduling contexts, these components are instantiated as follows: the state $\boldsymbol{s_t}$ typically encodes real-time shop floor information (e.g., machine status, job queues, due dates); the action a_t corresponds to a dispatching decision (e.g., assigning a job-operation to a machine); and the reward r_t is often designed as a negative of a scheduling objective (e.g., $-$tardiness or $-$makespan increment) to encourage minimization. As shown in **Figure 2.4**, the agent gets the states $\boldsymbol{s_t}$ from the environment and selects the actions a_t based on the exploitation and exploration procedure. The environment transitions to the next state $\boldsymbol{s_{t+1}}$ under the action. A reward r_t is fed back into the agent. The agent's goal is to learn a policy $\pi(a|\boldsymbol{s})$, which is a mapping from states

to actions. This policy π maximizes the expected return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, which represents the discounted sum of future rewards. This is typically achieved by estimating value functions: the state-value function $V^{\pi}(\boldsymbol{s})$ predicts the expected return from state $\boldsymbol{s}$ under policy π, while the action-value function $Q^{\pi}(\boldsymbol{s}, a)$ evaluates taking action a in state $\boldsymbol{s}$ and then following π.

The agent learns and optimizes the policy based on the experience set ($\boldsymbol{s_t}$, a_t, r_t, $\boldsymbol{s_{t+1}}$). Under the trial-and-error process, the agent learns to select the appropriate action a^* that will maximize the cumulative rewards under state $\boldsymbol{s}^*$. Since the true transition dynamics $P(\boldsymbol{s}' \mid \boldsymbol{s}, a)$ are generally unknown in real-world scheduling, most applications adopt model-free RL methods, which learn directly from interaction data without requiring a system model.

2.5.1.2. Value-based Reinforcement Learning Algorithms

(1) Q-learning

Q-learning is a model-free reinforcement learning algorithm that learns the optimal action to take in each state. It does not require a model of the environment. The "Q" refers to the expected accumulated return by applying certain actions under a certain state. In the Q-learning algorithm, the Q-value matrix $Q(\boldsymbol{s}, a)$ is the crucial component that will be updated during the training. Q-learning is utilized to interact with the factory with stochastic state transitions for the derivation of optimal policies as

$$Q'(\boldsymbol{S_t}, A_t) = Q(\boldsymbol{S_t}, A_t) + \alpha[R_{t+1} + \gamma \max_{A_{t+1}} Q(\boldsymbol{S_{t+1}}, A_{t+1}) - Q(\boldsymbol{S_t}, A_t)], \qquad (2.1)$$

where γ is a discount factor for future rewards; α is the learning rate. $Q(\cdot)$ is a state-action value function, which is sequentially updated during the training period; A_t denotes a chosen action from the action space at time t; R_{t+1} is a reward for the state transition from $\boldsymbol{S_t}$ to $\boldsymbol{S_{t+1}}$. A scheduling experience of the AI scheduler is denoted by $(\boldsymbol{S_t}, A_t, R_{t+1}, \boldsymbol{S_{t+1}})$,

and all the experiences are stored for training purposes.

In the pattern-driven dynamic approach, Wei et al. [78] defined the state space uniformly. The number of the state is determined by the complexity of the problem. The uniformly divided state space and candidate dispatching rules create the Q-value matrix with limited state-action pairs. In the dynamic job shop scheduling problem, while the jobs come randomly, M. Aydin and E. Öztemel [79] applied Q-learning to select the most appropriate rules at a different decision point. Based on the range of the objective value mean slack time, the choice of the different rules returns different rewards. In a real-time dynamic flow shop scheduling problem, Wang et al. [80] proposed an RL approach based on the Q-learning method to support the decision-making process at every time step. The Q-value is an innovation in the RL learning methods, which quantifies the good and bad of the experience. However, for the large-scale problem, the Q-value matrix is not feasible. Moreover, the state space in real-time scheduling is fundamentally continuous, encompassing dynamic variables such as time, workload, and resource availability. Consequently, discrete tabular approaches like the Q-value matrix cannot adequately model the problem and are computationally impractical.

(2) Deep Q-network

To address more complex decision-making tasks, Deep Q-Networks (DQN) leverage deep neural networks to approximate the action-value (Q) function over continuous or high-dimensional state spaces [81]. Unlike tabular Q-learning, which stores a separate value for every state–action pair, this method uses a function approximator to generalize across states, enabling it to handle large or continuous state spaces. By storing past experiences in a replay buffer and sampling them randomly during training, experience replay further stabilizes learning and reduces temporal correlations. Together, these mechanisms enable DQN to scale effectively to complex scheduling environments and adapt robustly across diverse and dynamic conditions. Hu et al. [82] employed the DQN, which is a successful Deep

Reinforcement Learning (DRL) method, to solve the dynamic scheduling problem of FMSs involving shared resources, route flexibility, and stochastic arrivals of raw products. In a real-time and concurrent optimization of scheduling and reconfiguration for dynamic flow shop problems, Yang et al. [83] applied double-layer DQN with a "softmax" policy. The double-layer DQN contains two layers, the online layer and the offline layer, of critic functions. This structure separates the training process and exploitation of the policy; hence, it increases the efficiency of the algorithm. The "softmax" policy, rather than random selection in the exploration process, chooses the candidate based on the probability values, which can accelerate convergence while maintaining exploration.

Overall, value-based DRL methods like DQN are well-suited to scheduling tasks. They generalize across high-dimensional, stochastic environments without requiring full state enumeration, enabling real-time and adaptive decision-making.

2.5.1.3. Actor-Critic Network

In value-based methods, the Q-value pair $Q(\boldsymbol{s}, a)$ estimates the expected return obtained by executing action a in state $\boldsymbol{s}$ and subsequently following a given policy. For small-scale problems with discrete and finite state–action spaces, these Q-values can be represented exactly using a tabular format and stored directly in memory. However, value-based methods like Double DQN (DDQN) are inherently limited to discrete action spaces, as they rely on explicit maximization over $Q(\boldsymbol{s}, a)$ for all actions a. This makes them unsuitable for scheduling problems that require fine-grained or continuous control decisions, even if the state representation is handled via deep networks. To overcome this, policy gradient methods directly optimize the policy parameters θ by ascending the gradient of expected return, enabling stochastic or continuous actions. Actor-critic algorithms combine the strengths of both approaches: an actor (policy network) selects actions, while a critic (value network) evaluates them by estimating $V(s)$ or $Q(\boldsymbol{s}, a)$. The critic's low-variance feedback stabilizes policy updates, making actor-critic methods particularly effective for

high-dimensional scheduling tasks.

In the real-time flow shop scheduling with dynamic job arrivals, Yang et al. [84] compared the DQN, double-layer DQN, and the actor-critic algorithm Advantage Actor-Critic (A2C). A2C achieved shorter CPU times while maintaining performance comparable to that of value-based reinforcement learning algorithms. In the multi-agent manufacturing system for dynamic job shop scheduling, Zhang et al. [85] compared the performance of dispatching rules, Genetic Programming-based (GP-based) algorithms, DQN-based algorithms, and PPO algorithms. The proposed actor-critic-based PPO algorithm makes the AI scheduler obtain better scheduling results. Lin and Chen [86] adopted an actor-critic framework for the RL agent used in the two-stage flow shop problem. The RL agent is based on the PPO algorithm, where the actor is responsible for interacting with the environment to collect samples, and the critic is responsible for judging the action of the actor. Liu et al. [87] designed the Asynchronous Advantage Actor-Critic (A3C) algorithm agent according to the LeNet-5 network structure. To reduce the difficulty of training, the actor-network and critic-network are generally adopted to share part of the network structure and parameters.

2.5.1.4. Graph Convolutional Network

In dynamic scheduling problems, the scheduling process can be represented using graph structures such as disjunctive graphs. Hence, the graph convolutional network combined with the RL method can be applied to these scheduling problems. Jing et al. [88] developed an RL method to solve the decentralized flexible job shop problem. The problem is described by a probabilistic directed acyclic graph (DAG). In this formulation, nodes represent jobs or operations, and edges encode probabilistic precedence dependencies. Graph Convolutional Networks (GCNs) are then used to aggregate and transform node features by leveraging the DAG's topology, thereby capturing structural dependencies and contextual relationships among jobs.

2.5.2. Multi-Agent Methods and Decentralized Scheduling Systems

(1) Motivation for Decentralization

Large-scale or geographically distributed factories often require decentralized scheduling due to prohibitive communication latency between sites, limited bandwidth for centralized data aggregation, computational intractability of global optimization, and the need for local decision-making under partial observability. Centralized approaches may become bottlenecks due to communication delays or data overload. Multi-Agent Reinforcement Learning (MARL) addresses this by deploying multiple agents that operate based on local observations and execute decentralized policies, enabling scalable coordination without reliance on global information. This architecture enhances responsiveness to dynamic events, improves scalability across heterogeneous production units, and increases resilience to localized disruptions such as machine failures or stochastic job arrivals.

(2) Conflict Resolution and Coordination

Effective decentralized scheduling requires agents to resolve resource conflicts and coordinate decisions despite operating under partial observability, formally modeled as a decentralized partially observable Markov decision process (Dec-POMDP). To improve joint performance, agents may exchange limited local information, experiences, or learned intentions; however, such communication must be designed carefully to avoid excessive overhead and to mitigate the non-stationarity induced by concurrently learning peers. While global reward signals are sometimes shared across agents to promote cooperation, this approach alone often fails to resolve the credit assignment problem, for example, determining which agent's actions contributed to success or failure. Consequently, more sophisticated mechanisms, such as counterfactual baselines, difference rewards, or learned reward decomposition, are typically required to meaningfully align individual policies with

system-level objectives without undermining local autonomy.

In dynamic scheduling problems with multiple stages, such as flexible job shops or flexible flow shops, the manufacturing process involves a sequence of operations for each job, where each operation can be processed on one of several eligible machines. This structure naturally leads to a decomposition of the scheduling decision into two interrelated subproblems: machine assignment (routing), which determines which machine processes each operation, and local sequencing, which orders the operations assigned to each machine's queue. Multi-agent systems significantly reduce scheduling complexity and computational requirements. Liu et al. [89] proposed a modified version of the Deep Deterministic Policy Gradient (DDPG) algorithm that enables simultaneous training across multiple dynamic job shop scheduling tasks. In their approach, agents share common actor and critic networks, with experiences collected in parallel from distributed environment instances. This design introduces a training scheme that resembles asynchronous methods. It differs from standard DDPG, which relies on synchronized, off-policy updates from a centralized replay buffer. Additionally, Liu et al. [28] divided the scheduling problem into a routing agent for assigning jobs to machines and a sequencing agent for job ordering, employing asynchronous transition techniques and parameter sharing to simplify the multi-agent training.

Furthermore, Aissani et al. [25] introduced a multi-agent adaptive scheduling model tailored for multi-site operations, featuring observer, stock, and resource agents that dynamically collaborate to determine optimal pricing and resource allocation using the State–Action–Reward–State–Action (SARSA) reinforcement learning algorithm. Similarly, Qu et al. [41] utilized distributed RL in large-scale manufacturing networks, allowing agents to independently select dispatching rules based on local information while sharing learned parameters to enhance global performance. Zhou et al. [90] demonstrated that multi-agent RL enhances scheduling efficiency through agents' mutual interactions and cooperative learning within smart factories.

2.5.3. Case Studies: RL in Industry 4.0 Applications

In Industry 4.0, especially in dynamic manufacturing contexts such as semiconductor production, pharmaceutical manufacturing, and smart factories. For instance, Waschneck et al. [91] deployed cooperative deep Q-learning agents in a semiconductor fab to dynamically select dispatching rules based on real-time conditions like machine status and job urgency. This led to a dramatic reduction in late deliveries (e.g., delayed lots dropping from 17% to 1.3%) and a 50% decrease in cycle time spread. Similarly, in the pharmaceutical sector, Said et al. [36] applied an online Q-learning algorithm to manage dynamic disruptions in flexible job-shop environments, including machine breakdowns and urgent job arrivals. The proposed method achieved a 33% average reduction in makespan over a baseline schedule across various problem instances, highlighting its potential to improve operational efficiency in real-time scheduling contexts.

Crucial prerequisites for effectively implementing RL systems include access to real-time data and robust computational infrastructures. Real-time data streams are vital because they enable RL algorithms to continuously learn from and adapt to the most current operational conditions. This capability is particularly important in dynamic environments where rapid adjustments are necessary to maintain efficiency and productivity, such as in manufacturing or autonomous driving. Meanwhile, advanced computational resources facilitate complex simulations and extensive training phases, which are indispensable for developing accurate and reliable RL models. High-performance computing allows for the processing of large datasets and the execution of computationally intensive tasks, such as deep neural network training, ensuring that the RL models can generalize well from the training data to unseen situations, thereby enhancing their reliability and applicability in real-world scenarios.

Nevertheless, significant challenges persist. The interpretability of RL decision-making remains limited because learned policies, especially those based on deep neural networks, typically operate as black-box functions that map observations to actions without providing

explicit rationale or intermediate reasoning. This lack of transparency complicates efforts to understand, validate, or justify individual scheduling decisions, particularly in safety-critical or highly regulated manufacturing environments.

2.6. Multi-Objective Optimization Problem in Manufacturing

2.6.1. Definition

Multi-objective optimization involves optimizing two or more objectives simultaneously, and their corresponding set of constraints, aiming to find balanced solutions. Unlike single-objective optimization, which targets a single criterion, multi-objective problems consider multiple practical factors, especially relevant in manufacturing processes. As such, the mathematical representation of multi-objective optimization problems (MOOP) is shown in **Equations (2.2)** to **(2.5)** [92].

$$\min f_n(x_i); n = 1,2,3, \dots, N \tag{2.2}$$

Subject to:

$$g_j(x_i) \geq 0; j = 1,2,3, \dots, p, \tag{2.3}$$

$$h_j(x_i) = 0; j = p+1, \dots, J, \tag{2.4}$$

$$x_i^{min} \leq x_i \leq x_i^{max}; i = 1,2,3, \dots, I, \tag{2.5}$$

where:

$f_n(x_i)$: Objective function n of decision variable x_i

$g_j(x_i)$: Inequality constraint j as a function of decision variable x_i

$h_j(x_i)$: Equality constraint j as a function of decision variable x_i

x_i: Decision variable

x_i^{min}: Minimum value of decision variable x_i

x_i^{max}: Maximum value of decision variable x_i

N: Total number of objective functions

J: Total number of inequality and equality constraints

I: Total number of decision variables

The feasible solution space vector $\boldsymbol{C}$ is defined that fulfill all the constraints stated in the optimization problem:

$$\boldsymbol{C} = \begin{cases} x_i \in \mathbb{R}: g_j(x_i) \geq 0; j = 1,2,3, \dots, p \\ h_j(x_i) = 0; j = p+1, \dots, J \\ x_i^{min} \leq x_i \leq x_i^{max}; i = 1,2,3, \dots, I \end{cases} \tag{2.6}$$

Pareto optimality is central to multi-objective optimization, defining solutions where no objective can be improved without worsening another. Mathematically, for a given set of feasible solutions $X_1 \in \boldsymbol{C}$, the objective vector $F(X_1) = \{f_1(X_1), \dots, f_N(X_1)\}$ is Pareto optimal if there is no vector $X_2 \in \boldsymbol{C}$ such that:

- $f_n(X_2) \leq f_n(X_1)$ for all $n = 1,2,3, \dots, N$
- There exists at least one $n \in \{1,2,3, \dots, N\}$ with $f_n(X_1) < f_n(X_2)$

The Pareto front represents the set of these optimal solutions. As mentioned above, X_1 dominates X_2. Normally, at least one objective is lower in this solution, while others are not higher than another solution in the manufacturing scheduling problem. **Figure 2.5** illustrates these concepts of MOOP with two minimization conflicting objective functions, where all the points between $(f_2(\hat{x}), f_1(\hat{x}))$ and $(f_2(\tilde{x}), f_1(\tilde{x}))$ define the Pareto front. These points are called non-dominated points.

In manufacturing contexts, multiple competing factors typically must be considered, including production cost, makespan, flow time, quality, and energy efficiency. In multi-objective optimization problems, objectives may align or complement each other.

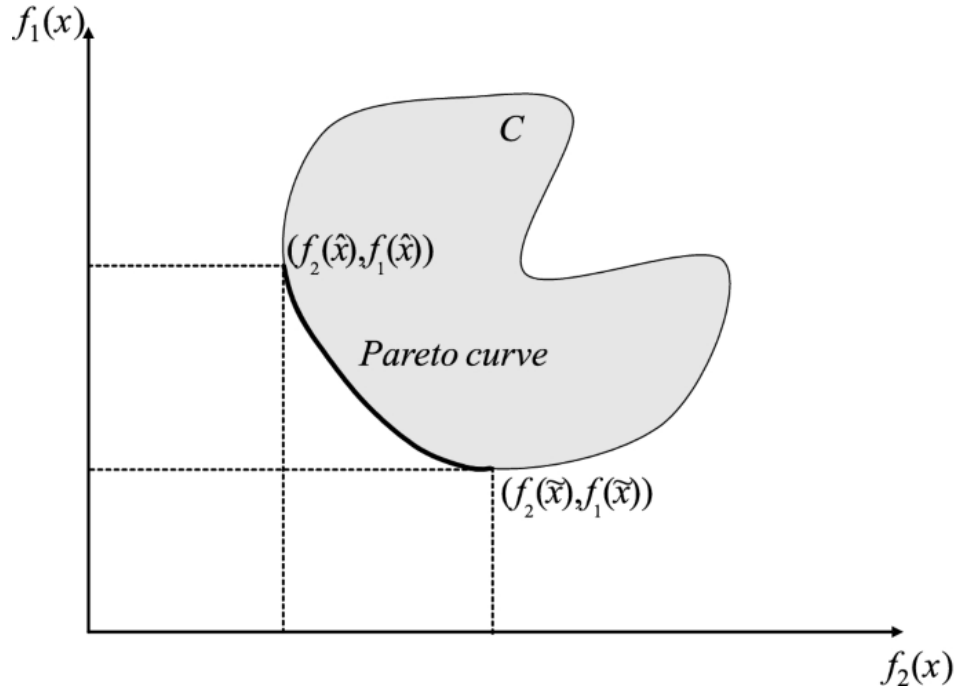


Figure 2.5. The diagram of a multi-objective optimization problem with two minimization conflicting objective functions (reproduced from [92]).

For instance, shorter makespan often requires running machines at higher speeds or employing more equipment simultaneously [93], [94]. Another example is the trade-off between total production cost and delivery reliability [95], [96].

2.6.2. Common Objectives in Manufacturing

Manufacturing environments often involve multiple interrelated objectives. The following notations present the common optimality criteria by [56].

C_j Completion time

L_i/L_{max} Lateness/maximum lateness

T_i/T_{max} Tardiness/maximum tardiness

U_i Unit penalty

C_{max} Makespan

$\sum C_j$ Total completion time

$\sum E_i$ Total earliness

$\sum T_i$ Total tardiness

$\sum U_i$ Number of late jobs

$\sum w_i C_i$ Total weighted completion time

$\sum w_i U_i$ Weighted number of tardy jobs

$\sum w_i T_i$ Total weighted tardiness

$\sum w_i(d_i - C_i)$ Total weighted earliness

Makespan is the most used completion-time-based objective in both static and dynamic scheduling problems, though its definition can vary. In static scheduling, the maximum completion time of all jobs serves as a performance indicator, illustrating the manufacturing process's goal of finishing all orders as quickly as possible.

Flow time is the time a job spends within the system. Lead time, which is more commonly used in supply chain problems, is defined as the interval between the initiation and completion of a production process and often includes transportation and delivery times. Because the working times of different jobs vary, average flow time is widely used as an objective function in dynamic scheduling problems, e.g., in references [97], [98], [99], [100], [101], [102], [103], and [104].

Due date–related objectives have practical significance because missing a due date incurs penalty costs in real production settings. Tardiness T_i, earliness E_i , and lateness L_i are three basic forms of due date–related objectives, but many variations appear in the literature. Thürer et al. [52] and Sels et al. [97] applied the basic form of tardiness; Ebrahimi et al. [105] used a weighted sum of tardiness; Shnits [101], as well as Baykasoğlu and Karaslan [98] used the average form of tardiness; Wang et al. [106] and Choi et al. [107] considered maximum lateness as an objective that must be minimized; Additionally, Rafiee et al. [108], Chongwatpol and Sharda [94], Georgiadis and Michaloudis [109], and Rossi et al. [96] optimized the number of tardy jobs; Weng and Fujimura [110] employed just-in-time (JIT)

as their objective, and Sels et al. [97] used the proportion of tardy jobs as another objective function in the scheduling system.

Machine-related objectives are often introduced when dynamic scheduling aspects come into play. Machine status is crucial in some research, as machine breakdowns must be prevented. For example, Sheikhalishahi et al. [111] incorporated machine availability as one objective in an open shop scheduling problem that includes preventive maintenance. Machine-related objectives can also be used to evaluate system efficiency instead of focusing solely on job-related metrics. Pradhan et al. [112] used throughput and work-in-progress (WIP) to measure performance in a scheduling system with production failures. Chongwatpol and Sharda [94] and Wong et al. [113] used utilization to represent machine usage. Georgiadis and Michaloudis [109] adopted average WIP to indicate the long-term performance of the system.

Economic objectives are the most used performance indicators in the reviewed literature, reflecting the pursuit of economic benefits. Net profit and cost are two basic types of economic objectives. Both capture the system's overall performance in comparison to objectives such as makespan, flow time, or tardiness. Although cost and profit are often used as simplified indicators of a manufacturing system's economic performance, real-world applications typically involve a variety of cost components. These may include setup cost, production cost, transportation cost, storage cost, and reconfiguration cost, especially in cellular manufacturing. Additionally, due date–related costs such as lateness, earliness, and tardiness, as well as maintenance and repair costs, must also be considered. Applying economic objectives requires detailed information on the costs associated with different processes. However, in real markets, material prices are not static, and product prices may fluctuate as well.

2.6.3. Reinforcement Learning in Multi-Objective Manufacturing Scheduling Problems

Recent work by Li et al. [114] demonstrates how reinforcement learning can address multi-objective scheduling in resource-constrained environments. In their study of flexible job shops with limited AGVs, the authors formulate a hybrid deep Q-network that jointly minimizes makespan, total tardiness, and workload imbalance while maximizing transportation efficiency. Their reward function explicitly balances these competing objectives through weighted aggregation, and their action space is designed to coordinate machine assignment and material handling decisions in real time. This integrated approach underscores a key principle in RL-based manufacturing scheduling: effective policies must account for multiple performance criteria and physical system couplings, rather than optimizing a single metric in isolation. Similarly, Zhou et al. [115] applied a weighted sum approach in a job shop scheduling scenario, targeting reduced makespan, lowered production costs, and improved machine utilization. Although effective, these weighted-sum methods simplify multi-objective scenarios into single-objective frameworks.

Luo et al. [116] present a multi-objective deep reinforcement learning approach for dynamic flexible job shop scheduling, modeling the problem as an MDP with a composite reward function that jointly minimizes makespan, tardiness, and workload imbalance. By embedding these objectives directly into the reward design, their deep Q-network learns dispatching policies that balance competing performance criteria in real time.

In real-world manufacturing environments, objectives often shift dynamically, such as prioritizing energy efficiency during peak electricity hours. Traditional multi-objective metaheuristics, which typically rely on static or predetermined objective weights, struggle with such flexibility. In contrast, reinforcement learning methods can dynamically incorporate priority shifts through adaptive weighting schemes or scalarization techniques. Furthermore, Pareto-based reinforcement learning methods, such as Pareto Q-learning,

enable the capturing of multiple trade-offs without collapsing them into a single metric. However, effectively translating Pareto optimization into scalable and interpretable RL reward structures remains an open challenge, representing a critical frontier for developing more adaptive, transparent, and decision-supportive multi-objective scheduling systems in real-world manufacturing.

2.7. Explainable Reinforcement Learning and the Role of SHAP in Scheduling

As RL systems are increasingly applied to complex operational domains such as manufacturing, logistics, and energy management, the need for interpretable policies has become paramount. This has led to the emergence of Explainable Reinforcement Learning, which seeks to render agent decisions transparent to human stakeholders without compromising performance [117]. Among post-hoc explanation techniques, SHAP (SHapley Additive exPlanations) has gained widespread adoption due to its theoretical grounding in cooperative game theory and model-agnostic nature [118].

SHAP assigns each input feature an importance value for a given prediction by computing Shapley values, defined as the average marginal contribution of a feature across all possible subsets of other features:

$$\phi_i = \sum_{S \subseteq F\{i\}} \frac{|S|!\,(|F| - |S| - 1)!}{|F|!} [f(S \cup \{i\}) - f(S)], \tag{2.7}$$

where F is the full set of input features, S is any subset excluding feature i, and $f(S)$ denotes the model output when only features in S are active (others replaced by reference values). This formulation satisfies key axiomatic properties, including local accuracy, consistency, and missingness, making it theoretically appealing for attribution tasks.

In practice, SHAP has been successfully deployed in high-stakes domains: identifying clinical biomarkers in medical diagnosis [119], justifying credit approvals in finance [120], and generating saliency maps in computer vision [121]. However, a well-documented limitation of the standard SHAP implementation is its reliance on the feature independence assumption, which underpins the use of marginal expectations when estimating Shapley values [118]. This assumption is routinely violated in scheduling contexts, where state variables such as job slack, due date, remaining processing time, and queue length are inherently interdependent due to temporal, logical, or physical constraints. Despite this, empirical studies have shown that SHAP often produces stable relative rankings of feature importance when dependencies are consistent with the data distribution encountered during training and evaluation [122]. Consequently, many applied explainable reinforcement learning works treat SHAP not as a tool for computing exact Shapley values, but as a pragmatic diagnostic instrument for identifying dominant decision drivers and validating behavioral plausibility.

For instance, Julian and Kochenderfer [123] use SHAP to verify that a PPO-based autonomous driving policy bases throttle decisions on physically meaningful features like headway distance, rather than spurious correlations, thereby confirming behavioral plausibility in longitudinal control. Alzantot et al. [124] apply SHAP post-hoc to a DDPG agent for vehicle speed control, focusing on the relative ranking of traffic-related inputs (e.g., relative velocity) to identify dominant decision factors under stochastic conditions. Van Wyk et al. [125] employ SHAP in aerospace RL to audit guidance policies, using feature attributions as a diagnostic check to ensure the agent relies on legitimate flight state variables (e.g., Mach number, altitude) during high-speed maneuvers. Similarly, Miller et al. [126] leverage SHAP in building energy control to interpret an RL policy for mixed-mode ventilation in tropical climates. By identifying dominant environmental drivers such as outdoor enthalpy and indoor CO_2 levels, they translate complex action choices into human-understandable operational rules, effectively bridging machine decisions and domain

expertise.

To mitigate the independence violation more rigorously, methodological extensions such as conditional SHAP [122] and causal SHAP [127], [128] have been proposed. These approaches replace marginal with conditional expectations or incorporate causal structural models to define valid feature interventions. However, they require explicit modeling of feature relationships through generative models, copulas, or domain-specified graphs, which is often infeasible in high-dimensional RL state spaces with dozens of correlated variables [117].

Given these considerations, the use of SHAP in dynamic flexible job shop scheduling can be applied as a justifiable pragmatic choice, provided its limitations are acknowledged, and interpretations focus on comparative trends rather than absolute attribution magnitudes. While direct industrial deployments of SHAP in semiconductor manufacturing or predictive maintenance remain undocumented in the public literature, the demonstrated utility of this approach in other safety-critical domains (e.g., autonomous control and building energy management) provides a methodological precedent for its adoption in complex scheduling environments. Therefore, SHAP is leveraged to generate interpretable, feature-level narratives that can support human operators in understanding and trusting RL-driven scheduling decisions, even when the underlying attributions are approximate.

2.8. Challenges and Evolutionary Perspectives

2.8.1. Bridging Theory and Practice

The adoption of dynamic job shop scheduling (DJS) methodologies, particularly RL-based solutions, confronts substantial practical barriers. Among these barriers, data availability remains critical. Real-world manufacturing environments frequently lack comprehensive, real-time data streams, constraining the effective training and operation of RL algorithms

[129]. Moreover, model fidelity presents an additional challenge, as theoretical models may fail to capture critical shop-floor dynamics. This discrepancy between modeled and real conditions can significantly undermine performance when deployed [130]. Legacy systems often lack compatibility with modern AI-driven solutions, requiring significant infrastructural adjustments and additional resources, thereby posing substantial implementation barriers [131].

Validation in actual industrial environments is a crucial step towards bridging theoretical advancements with practical applications. Employing digital twins, which are virtual replicas of real-world manufacturing systems, allows for robust, risk-free exploration of reinforcement learning scheduling algorithms under realistic operating conditions, significantly enhancing the efficiency of transitioning from simulation to shop-floor implementation [132]. Additionally, pilot projects conducted in controlled settings within the factory offer practical insights and facilitate iterative adjustments, ensuring algorithm performance closely aligns with operational demands before broader adoption.

2.8.2. The Future of Scheduling in Smart Factories

Emerging technologies such as federated learning and edge computing represent transformative directions in smart manufacturing scheduling. Federated learning allows multiple geographically dispersed factories to collaboratively train sophisticated scheduling models without sharing sensitive raw data. This approach can also extend to individual machines located at the network edge, for example, decentralized devices that generate and process data locally rather than relying on a central cloud system. This decentralized approach not only enhances privacy and data security but also enables faster, more robust learning across diverse manufacturing environments, potentially revolutionizing inter-factory coordination and efficiency [133].

The growing interest in human-AI collaboration highlights another critical evolutionary

trend in smart factory scheduling. Explainable RL and user-in-the-loop systems are becoming increasingly vital for fostering trust and facilitating acceptance among human operators. Transparent and interpretable decision-making frameworks enable operators to understand and, if necessary, override or adjust AI-generated schedules. This collaborative interaction ensures that operational expertise complements algorithmic efficiency, promoting smoother adoption and more reliable shop-floor operations [134].

Looking ahead, the ultimate vision for scheduling in smart factories involves the development of extensive decentralized frameworks that continually learn from high-volume sensor data streams in near-real time. Such frameworks would dynamically adjust scheduling strategies in response to subtle shifts or major disruptions, aiming to enhance productivity, flexibility, and resilience. This high-level roadmap points toward an integrated, intelligent future where scheduling transcends static optimization to become an inherently dynamic and adaptive facet of manufacturing excellence [135].

2.8.3. Critical Reflections and Research Gaps

Despite extensive research efforts in scheduling—including conventional heuristics, metaheuristics, and artificial intelligence-driven techniques—key limitations remain inadequately addressed within current literature. Firstly, existing studies frequently prioritize the introduction of novel algorithms or frameworks, often overlooking practical constraints such as limited sensor data availability, real-time decision latency requirements, and integration challenges with legacy ERP systems. Traditional heuristics, such as FIFO, SPT, and EDD, demonstrate computational efficiency; however, these approaches typically struggle to capture complex interactions within highly dynamic environments due to incomplete state representations and the absence of real-time feedback, leading to suboptimal decisions when disruptions occur. Conversely, advanced RL methods, although promising for adaptive scheduling, can produce policies whose decision logic is opaque to human operators. This shortcoming restricts practical acceptance, as practitioners and shop-

floor managers may hesitate to rely on or adjust policies lacking a transparent rationale.

Secondly, despite the increased application of multi-objective optimization approaches, such as simultaneously minimizing makespan and costs or energy consumption, many existing studies overly simplify complex decision scenarios through aggregated weighted-sum objectives. This approach frequently neglects intricate trade-offs inherent in actual manufacturing contexts, including balancing efficiency, reliability of delivery dates, and resource consumption. Although more nuanced Pareto-based optimization methods are gradually emerging, additional exploration is necessary to demonstrate their practical effectiveness, particularly within decentralized or distributed scheduling frameworks where global coordination presents significant complexities.

Thirdly, there remains a scarcity of comprehensive comparative analyses that explicitly evaluate new scheduling methods across critical dimensions such as scalability, robustness to disruptions, and computational or implementation cost. While metaheuristic and RL methodologies commonly outperform basic dispatching rules under benchmark scenarios, systematic evaluations concerning constraints—such as scalability, computational demands, and resilience against machine failures—are notably lacking. Additionally, few studies rigorously assess the transferability of these methods from controlled simulation environments to actual production scenarios, leading to persistent gaps concerning data availability, model accuracy, and compatibility with existing ERP/MRP systems. These gaps significantly affect real-world applicability and feasibility.

Considering these critical assessments, two primary research gaps require further investigation. The first involves developing more interpretable RL-based scheduling solutions capable of delivering transparent and justifiable decisions in dynamic production environments. The opaque nature of current "black-box" RL algorithms impedes widespread industrial adoption, as stakeholders require understandable and transparent systems for effective risk management and regulatory compliance.

The second identified gap pertains to empirical exploration into effectively managing multiple and dynamically evolving objectives, including costs, adherence to due dates, and product quality, especially within real-time, decentralized, or multi-agent scheduling contexts. Facilitating robust and efficient coordination among multiple agents, each optimizing distinct local objectives, remains challenging. Synchronizing and effectively communicating Pareto-optimal solutions across various work centers continues to represent an unresolved issue in the literature.

Collectively, these identified limitations highlight the necessity of adopting a holistic research approach, emphasizing not only algorithmic advancements but also addressing interpretability, scalability, and the alignment with realistic manufacturing constraints. Explicit acknowledgment of these gaps, particularly those related to transparency, multi-agent coordination, and empirical validation in real-world conditions, will enable researchers to better illustrate the originality and practical relevance of proposed scheduling frameworks.

2.9. The Contribution of this Thesis

This thesis contributes a series of original advancements in dynamic scheduling for modern manufacturing environments, directly addressing critical gaps in existing approaches to the DFJSP under Industry 4.0 conditions. As identified in the literature review, current methods suffer from three key limitations: (1) reliance on centralized or rigid architectures that hinder scalability; (2) dependence on discrete dispatching rules that lack adaptability under disruption; and (3) treatment of scheduling as a single-objective problem, ignoring real-world trade-offs such as tardiness versus setup cost. To address these gaps, this research proposes a distributed, event-driven reinforcement learning framework that evolves across three stages of increasing complexity and realism—each validated through rigorous simulation with quantifiable improvements over strong baselines.

First, to overcome the scalability and responsiveness bottlenecks of global optimization and

heuristic coordination, a decentralized DDQN architecture is introduced, where each work center schedules autonomously using only local observations. This design eliminates communication overhead while enabling real-time reaction to stochastic job arrivals and machine unavailability. In benchmark tests (**Cases 1A**–**1C**), the DDQN agent consistently achieves the lowest average tardiness and highest win rate among all rule-based and learning-based comparators, establishing a robust foundation for distributed intelligence.

Second, discrete rule selection is a known limitation in both classical and early reinforcement learning–based schedulers, as it restricts adaptability in dynamic environments. To overcome this inflexibility, this thesis introduces a continuous prioritization mechanism using PPO. Two variants are evaluated: PPO with weighted features (PPO-WF) and PPO with raw state inputs (PPO-RS). As shown in **Section 5.5** (**Table 5.2** and **Figure 5.4**), PPO-WF achieves a win rate of 40.0%, significantly outperforming PPO-RS (30.0%), the best traditional heuristic SPT (27.0%), and other rules like EDD (2.0%) and LOR (1.0%). Moreover, PPO-WF attains the lowest average tardiness of 653.65, compared to 671.65 (PPO-RS) and 672.51 (SPT), while exhibiting a lower standard deviation, confirming its superior robustness under operational uncertainty. This represents a conceptual and practical advance: by moving from rule switching to continuous, feature-weighted decision-making, the scheduler gains both performance and interpretability.

Third, real production systems require balancing competing goals, such as minimizing job tardiness and reducing sequence-dependent setup times. To reflect this multi-objective nature, this work extends the DFJSP into a dual-objective formulation. A novel integration of UVFA with DDQN enables a single agent to generalize across arbitrary preference weights without retraining. Critically, a baseline-referenced reward design is introduced to stabilize learning by measuring improvement relative to rule-based policies. As quantified in **Table 6.4**, the modified reward scheme (tardiness-only variant) yields dramatically improved Pareto approximation quality, achieving GD = 22.42, IGD = 5.30, and Spread =

0.727—substantially better than origin-reward baselines (e.g., IGD reduced from 161.28 to 5.30). This demonstrates that the agent not only learns high-quality trade-off policies but does so with enhanced convergence and coverage, enabling near-Pareto solutions across dynamic business priorities.

Across all three stages, transparency is ensured through SHAP-based interpretability analysis, which identifies the most influential scheduling features (e.g., job slack, queue length, machine status) driving agent decisions. This bridges the "black-box" gap and supports practitioner trust—essential for industrial adoption.

In summary, this thesis advances dynamic scheduling for smart manufacturing through three interlinked innovations: a decentralized reinforcement learning architecture that ensures robustness under uncertainty, a continuous prioritization mechanism that overcomes the rigidity of rule-based dispatching, and a multi-objective formulation that enables adaptive trade-offs between competing production goals. By integrating scalability, interpretability, and generalization into a unified framework, this work provides a foundation for intelligent, responsive, and deployable scheduling systems in real-world industrial environments.

Chapter 3. Formulation of the Dynamic Scheduling Problem

This chapter presents a formal formulation of the Dynamic Flexible Job Shop Scheduling Problem (DFJSP) under three progressively complex scenarios that reflect core challenges in modern, responsive manufacturing systems. These scenarios are designed to capture increasing levels of operational uncertainty: (1) stochastic job arrivals in a decentralized production environment, representing demand-side volatility; (2) unreliable machine resources subject to random breakdowns, modeling supply-side disruptions; and (3) multi-objective trade-offs between tardiness and sequence-dependent setup times, reflecting real-world conflicts between service-level performance and operational efficiency.

The baseline scenario (**Section 3.1**) assumes jobs arrive dynamically according to a Poisson process, each requiring a sequence of operations that can be processed on any eligible machine within predefined work centers. Processing times are stochastic but known in distribution, and scheduling decisions are made locally at each work center upon job arrival or machine idle events. The objective is to minimize total weighted tardiness, with decisions governed by a decentralized policy framework.

Section 3.2 extends this setting by incorporating machine unreliability: machines are subject to random failures modeled via exponential time-to-failure distributions, characterized by mean time between failures (MTBF), and require random-duration repairs when broken down. This introduces disruption states that interrupt processing and trigger rescheduling, thereby testing the robustness of decision policies under resource disruptions.

Finally, **Section 3.3** formulates a multi-objective DFJSP in which each scheduling decision affects both job tardiness and sequence-dependent setup times incurred when switching between job families on a machine. Rather than optimizing a single scalarized objective, the

problem explicitly requires balancing these competing goals, laying the foundation for preference-conditioned reinforcement learning in later chapters.

For each scenario, the problem is rigorously defined in terms of state representation, action space, reward structure, and system constraints. This formalization provides the necessary grounding for the design and evaluation of reinforcement learning–based scheduling agents in **Chapters 4** through **6**.

The remainder of this chapter is organized as follows: **Section 3.1** presents the baseline dynamic scheduling problem with stochastic arrivals; **Section 3.2** introduces machine breakdowns and models their impact on system dynamics; and **Section 3.3** extends the formulation to a dual-objective setting involving tardiness and setup costs.

3.1. Problem 1: Baseline Model with Random Job Arrivals

3.1.1. Core Flexible Job Shop Scheduling Model

In modern manufacturing systems, the DFJSP presents significant optimization challenges due to the inherent uncertainty in job arrivals and machine availability. Unlike static scheduling formulations, the DFJSP requires decisions to be made sequentially in response to dynamic events, such as machine status changes and newly released jobs. The objective is to minimize mean tardiness while ensuring efficient utilization of resources across distributed work centers. To model this sequential decision process, the problem is formulated as a Markov Decision Process (MDP), which provides a principled framework for representing state transitions, actions, and performance feedback. Within this framework, a reinforcement learning agent selects scheduling actions based on the current system state, guided by a defined action space and reward signal. The mathematical formulation of the DFJSP and its underlying modeling assumptions are presented in the following subsections.

Named Entities	
J_i	The i -th job instance released into the system ($i = 1, \dots, N$), ordered by release time r_i
τ_α	The α -th job type (category), defined by structural properties such as number of operations, routing, and processing time characteristics ($\alpha = 1, \dots, A$)
$O_{i,j}$	The j -th operation of job instance J_i ($j = 1, \dots, n_{\alpha(i)}$)
W_l	The l-th work center in the system. ($l = 1,2, \dots, L$)
$M_{l,k}$	The k-th machine of the l-th work center. ($k = 1,2, \dots, m_l$)

♦ Noted: Each job instance J_i belongs to exactly one job type: $J_i \sim \tau_{\alpha(i)}$

Indices	
$i = 1, \dots, N$	Index of job instance J_i
$\alpha = 1, \dots, A$	Index of job type τ_α
$\alpha(i) \in \{1, \dots, A\}$	Job type index assigned to instance J_i
$j = 1, \dots, n_{\alpha(i)}$	Index of operation $O_{i,j}$ within job J_i
$l = 1, \dots, L$	Index of work center W_l
$k = 1, \dots, m_l$	Index of machine $M_{l,k}$ within work center W_l

Parameters	
A	Number of job types ($\tau_1, \dots, \tau_A$)
N	Total number of job instances $(J_1, \dots, J_N)$ released during the simulation horizon
L	Number of the work centers $(W_1, \dots, W_L)$
m_l	Number of parallel machines in work center W_l
n_α	Number of operations required by any job of type τ_α
$w_{\alpha,j}$	Required work center for operation j of type-α jobs (i.e., $O_{i,j}$ of $J_i \sim \tau_\alpha$ must be processed in $W_{w_{\alpha,j}}$)
$P_{\alpha,j}^{l,k}$	Processing time of operation j of a type-α job on machine $M_{l,k}$ (Only defined when $l = w_{\alpha,j}$; otherwise, undefined or infeasible)
λ_α	Arrival rate (exponential distribution parameter) for jobs of type τ_α
r_i	Release time of job instance J_i
d_i	Due date of job instance J_i

Decision Variables	
$X_{i,j}^{l,k} \in \{0,1\}$	$\begin{cases} = 1, \text{if operation } O_{i,j} \text{ is assigned to machine } M_{l,k} \\ = 0, \quad \text{otherwise} \end{cases}$
$S_{i,j}^{l,k} \geq 0$	The starting time of operation $O_{i,j}$ on the machine $M_{l,k}$
$C_{i,j} \geq 0$	The completion time of the operation $O_{i,j}$
$C_i \geq 0$	The completion time of job J_i; $C_i = \max_j C_{i,j}$

A Flexible Job-Shop Scheduling Problem (FJSSP) is a hybrid of JSP and Parallel Machine Problem and is strongly NP-hard due to [122]:

(1) Sequence decisions of operations to a subset of the work center and

(2) Assignment decisions on operations on each machine in each work center.

An $A \times L$ Flexible Job Shop can be formulated as L independent work centers

$$W = \{W_1, W_2, \dots, W_L\}. \tag{3.1}$$

In each work center, there are m_l identical parallel machines

$$M_l = \{M_{l,1}, M_{l,2}, \dots, M_{l,m_l}\} \ (l = 1,2, \dots, L). \tag{3.2}$$

The number of machines in each work center is predefined, while each work center can have a different number of identical parallel machines. There are A job types, denoted by $\tau_1, \tau_2, \dots, \tau_A$. Each type of job τ_α, is characterized by a fixed sequence of n_α operations, denoted $O_{\alpha,1}, O_{\alpha,2}, \dots, O_{\alpha,n_\alpha}$, where $O_{\alpha,j}$ represents the j-th operation of any job belonging to type τ_α. Specifically, the processing time of operation j of a type-α job on machine $M_{l,k}$ is denoted by $P_{\alpha,j}^{l,k}$, which is only defined for machines in the required work center $W_{w_{\alpha,j}}$.

Table 3.1 is an example of the 3×3 Flexible Job Shop Manufacturing system, where the number presents the processing times $P_{\alpha,j}^{l,k}$ and the symbol '-' indicates that the operation cannot be processed on the corresponding machine. To complete a job of τ_α, all operations $O_{\alpha,j}$ must be completed on machines with machining capabilities.

In the static FJSSP, all scheduling information is known beforehand. In the dynamic FJSSP, job details such as release times are revealed during execution. The initial settings, including job types, operation sequences, and work center configurations, are fixed in advance. This research addresses real-time adaptability to dynamic job releases.

Table 3.1. Instance of a $\mathbf{3 \times 3}$ flexible job shop manufacturing system. Each row corresponds to an operation of a job type $(\boldsymbol{\tau_1, \tau_2, \tau_3})$. The numeric entries denote the processing time of the operation on the respective machine; a dash '-' indicates that the operation cannot be processed on the corresponding machine.

Job types	Operations	$\boldsymbol{W_1}$		$\boldsymbol{W_2}$		$\boldsymbol{W_3}$	
		$\boldsymbol{M_{1,1}}$	$\boldsymbol{M_{1,2}}$	$\boldsymbol{M_{2,1}}$	$\boldsymbol{M_{2,2}}$	$\boldsymbol{M_{3,1}}$	$\boldsymbol{M_{3,2}}$
τ_1	$O_{1,1}$	-	-	3	2	-	-
	$O_{1,2}$	1	4	-	-	-	-
	$O_{1,3}$	-	-	-	-	2	4
τ_2	$O_{2,1}$	1	3	-	-	-	-
	$O_{2,2}$	-	-	3	2	-	-
	$O_{2,3}$	-	-	-	-	5	4
τ_3	$O_{3,1}$	-	-	2	3	-	-
	$O_{3,2}$	-	-	-	-	6	7
	$O_{3,3}$	4	5	-	-	-	-

J_i is the i-th job released to the system, whose job type is $\tau_{\alpha(i)}$. r_i is the release time of the job J_i. Although the real manufacturing process is more complicated, some assumptions must be made to simplify the simulated environment. These assumptions are shown as follows:

System setup:

- *Initial Conditions:* All machines $M_{l,k}$ are available at the beginning (time zero).
- *Installation and Transportation:* The installation and transportation times are negligible.

Machine Characteristics:

- *Availability and Disruption:* Each machine is continuously available for production, and machine disruptions are ignored.
- *Exclusive Processing:* Each machine processes only one operation at a time, and operations cannot be interrupted once started. $\sum_{i=1}^{n}\sum_{j=1}^{c} X_{i,j}^{l,k} = 1$.
- *Identical parallel machines:* Machines have the same speed for the same job types.

Job and Operation Specification:

- *Operation Precedence:* Each operation $O_{i,j}$ has at least one capable machine and must follow a set order of precedence, e.g., $O_{i,1}$ must be processed before $O_{i,2}$.
- *Job Independence and Processing Time:* Each job J_i is independent and has a fixed processing time $P_{\alpha(i),j}^{l,k}$. The processing times $P_{\alpha,j}^{l,k}$ may differ on parallel machines within the same work center l for the same job type τ_α.

3.1.2. Dynamic Event Modeling: Stochastic Job Arrivals

Job Release Policies:

- *Dynamic Job Information:* Information about each job J_i becomes available only at its release time r_i. The release of different job types τ_α is independent.
- *Sequential Job Release:* No jobs are released at the beginning, and no batching is allowed, meaning only one job of each type can be released at any time.
- *Randomized Time Intervals for Job Releases:* The time intervals between the release times of consecutive jobs of the same job type follow the same time-distributed patterns,

incorporating a random function that leads to variable and unpredictable time intervals.

Job Arrival Process: Jobs are dynamically released according to a Poisson process with arrival rate λ_α. This means the inter-arrival times (T) between consecutive job releases are exponentially distributed:

$$P(T \leq t) = 1 - e^{-\lambda_\alpha t}, t \geq 0. \tag{3.3}$$

The probability of job $J_{i'}$ released at time t is described by **Equation (3.3)**. The time interval between two released jobs of the same job type follows the exponential distribution, while the mean intervals of different job types are the same, which is $1/\lambda_\alpha$. This release pattern simulates the random characteristic of the dynamic scheduling problem.

3.1.3. Total Work Content Method for Due Date Generation

In dynamic flexible job shop scheduling, generating realistic due dates for dynamically arriving jobs is critical for accurately simulating manufacturing environments and ensuring the practical relevance of scheduling strategies. The Total Work Content (TWC) method provides a systematic approach for establishing these due dates [103], [136], [137]. Under this method, the due date d_i for each dynamically released job J_i is calculated using its release date r_i and cumulative processing times of all operations associated with the job, adjusted by a scaling factor $k \in [1.0, 2.0] \subset R$. This interval reflects common practice in scheduling research, where $k = 1.0$ yields tight due dates (minimal slack) and larger values introduce more buffer time to accommodate shop-floor uncertainty. Mathematically, the due date calculation is expressed as:

$$d_i = r_i + k \sum_{j=1}^{n_i} P_{i,j}, \tag{3.4}$$

where r_i represents the release time of job J_i, n_i is the number of operations for job J_i, and $P_{i,j}$ is the processing time for the j operation of job J_i. The scaling factor k adjusts

the due date to reflect various urgency levels and manufacturing contexts.

This method mirrors real-world due date settings by accounting for the inherent processing time required for completing each job. It reflects realistic manufacturing environments where due dates are typically set based on estimated total work content to ensure feasibility and customer satisfaction. Moreover, it captures operational uncertainties and provides flexibility through the adjustment of the scaling factor k. By selecting an appropriate factor, schedulers can simulate different production scenarios, such as tighter deadlines during peak demand periods or more relaxed due dates in standard operations. Consequently, the TWC method not only simplifies the due date assignment process in simulations but also enhances the relevance and accuracy of dynamic scheduling models.

3.1.4. Objectives and Constraints

For the DFJSP, time intervals between the release times of consecutive jobs of the same job type follow the same time-distributed pattern. Each job is composed of a sequence of precedence-constrained operations. The mathematical model of the DFJSP follows the MILP process [138]. The release time r_i of job J_i of job type τ_α, and the processing time $P_{\alpha,j}^{l,k}$ are all integer values. The objective is to obtain a schedule with the lowest mean tardiness time of all jobs.

Objective:

$$\min \bar{T}_{mean} = \frac{1}{N}\sum_{i=1}^{N} T_i \text{, where } T_i = \begin{cases} C_i - d_i & C_i > d_i \\ 0 & C_i \le d_i \end{cases} \tag{3.5}$$

Subject to:

$$r_i \ge 0; \forall i = 1,2,\dots,N \tag{3.6}$$

$$r_1 \le r_2 \le \cdots \le r_N \tag{3.7}$$

$$P_{\alpha(i),j}^{l,k} > 0;\ \forall i = 1,2,\dots,N, j = 1,2,\dots,n_{\alpha(i)}, l = w_{\alpha(i),j}, k = 1,2,\dots,m_l \tag{3.8}$$

$$S_{i,1}^{l,k} \geq r_i,; \forall i = 1,2,\dots,N, l = w_{\alpha(i),j}, k = 1,2,\dots,m_l \tag{3.9}$$

$$S_{i,j}^{l,k} + P_{\alpha(i),j}^{l,k} \leq S_{i,j+1}^{l',k'}; \forall i = 1,2,\dots,N, j = 1,2,\dots,n_{\alpha(i)-1}, l = w_{\alpha(i),j}, \quad l' = w_{\alpha(i),j+1}, k = 1,2,\dots,m_l, k' = 1,2,\dots,m_{l'} \tag{3.10}$$

$$\text{For all } (i,j) \neq (i',j') \text{ with } X_{i,j}^{l,k} = X_{i',j'}^{l,k} = 1, \quad \text{either } C_{i,j} \leq S_{i',j'}^{l,k} \text{ or } C_{i',j'} \leq S_{i,j}^{l,k} \tag{3.11}$$

Objective Equation (3.5) minimizes the mean tardiness across all released job instances, where tardiness is defined as the positive difference between a job's completion time C_i and its due date d_i. **Constraint (3.6)** ensures that all jobs are released at non-negative times, reflecting that the system starts empty at time zero. **Constraint (3.7)** enforces that job indices are ordered chronologically by release time ($r_1 \leq r_2 \leq \cdots \leq r_N$), which provides a well-defined sequence for dynamic decision-making. **Constraint (3.8)** specifies that the processing time of operation j of job J_i on machine $M_{l,k}$ is determined solely by the job's type $\tau_{\alpha(i)}$, and is strictly positive when the machine belongs to the required work center. The **Constraint (3.9)** guarantees that no operation of a job can start before the job's release time r_i. **Constraint (3.10)** enforces the technological precedence among operations: each operation must be completed before its successor can begin, regardless of the work centers involved. **Constraint (3.11)** ensures machine capacity is respected—no machine processes more than one operation at any time—by requiring non-overlapping schedules for all operations assigned to the same machine. Finally, in the dynamic scheduling environment, decisions concerning operations that have already started are irreversible. At any rescheduling epoch t, only unscheduled or pending operations may be assigned; consequently, the start time of any such operation must satisfy $S_{i,j}^{l,k} \geq t$. This condition preserves the integrity of past decisions while enabling adaptive planning for future work.

3.1.5. Sub-Scenarios

To better reflect increasing levels of complexity and industrial relevance within Problem 1, three sub scenarios are introduced: **Sub-scenario 1A**, **Sub-scenario 1B**, and **Sub-scenario 1C**. These sub scenarios are designed to represent progressively more challenging conditions in dynamic flexible job shop scheduling, focusing on four key factors that significantly affect scheduling performance under uncertainty: processing time variability, routing flexibility, work center load imbalance, and machine heterogeneity. The choice of these factors is grounded in established literature identifying them as major sources of difficulty in real time scheduling decisions. **Sub-scenario 1A** provides a baseline with minimal uncertainty and structural simplicity, **Sub-scenario 1B** introduces moderate levels of stochasticity and resource diversity, and **Sub-scenario 1C** models a high complexity environment with pronounced variability and flexibility. The complete formal definition of each sub scenario, including the specific parameter values governing processing times, machine eligibility, work center configurations, and speed profiles, is provided later in this section.

Sub-Scenario 1A: Lawrence Benchmark Instances

This sub-scenario adapts the classical Lawrence benchmark instances [139], initially designed for static deterministic job shop scheduling, to align with dynamic flexible job shop scheduling requirements—a domain currently lacking standardized benchmark instances. The example of the DFJSP instances derived from the Lawrence instance la01 is shown in **Table 3.2**.

Table 3.2. Examples of DFJSP instances.

Job ID	Original Processing Times (unit times)	Modified Work Center Sequence	Parallel Machines per Work Center (WC)	Job Type	Arrival Distribution
1	[21, 53, 95, 55, 34]	WC1→WC0→WC4→WC3→WC2	WC0:2, WC1:3, WC2:2, WC3:3, WC4:4	Type A	Poisson (λ=0.5 jobs/hour)
2	[21, 52, 16, 26, 71]	WC0→WC3→WC4→WC2→WC1	WC0:2, WC1:3, WC2:2, WC3:3, WC4:4	Type B	Periodic (interval=2 hours)
3	[39, 98, 42, 31, 12]	WC3→WC4→WC1→WC2→WC0	WC0:2, WC1:3, WC2:2, WC3:3, WC4:4	Type C	Exponential (β=1.2 jobs/hour)
4	[77, 55, 79, 66, 77]	WC1→WC0→WC4→WC2→WC3	WC0:2, WC1:3, WC2:2, WC3:3, WC4:4	Type D	Uniform (min=1, max=4 hours)
5	[83, 34, 64, 19, 37]	WC0→WC3→WC2→WC1→WC4	WC0:2, WC1:3, WC2:2, WC3:3, WC4:4	Type E	Normal (μ=3 hours, σ=0.5)
6	[54, 43, 79, 92, 62]	WC1→WC2→WC4→WC0→WC3	WC0:2, WC1:3, WC2:2, WC3:3, WC4:4	Type F	Periodic (interval=4 hours)
7	[69, 77, 87, 87, 93]	WC3→WC4→WC1→WC2→WC0	WC0:2, WC1:3, WC2:2, WC3:3, WC4:4	Type G	Poisson (λ=0.3 jobs/hour)
8	[38, 60, 41, 24, 83]	WC2→WC0→WC1→WC3→WC4	WC0:2, WC1:3, WC2:2, WC3:3, WC4:4	Type H	Periodic (interval=3 hours)
9	[17, 49, 25, 44, 98]	WC3→WC1→WC4→WC0→WC2	WC0:2, WC1:3, WC2:2, WC3:3, WC4:4	Type I	Uniform (min=2, max=5 hours)
10	[77, 79, 43, 75, 96]	WC4→WC3→WC2→WC1→WC0	WC0:2, WC1:3, WC2:2, WC3:3, WC4:4	Type J	Exponential (β=1.5 jobs/hour)

Key Adaptations:

To simulate realistic production environments and accommodate dynamic decision-making within reinforcement learning frameworks, several key adaptations have been introduced to the baseline scheduling model. These adaptations are necessary to bridge the gap between simplified benchmark settings and more complex, real-world shop floor configurations. Specifically, they enhance model flexibility, improve resource representation, and facilitate decentralized decision-making. The main modifications are described as follows.

1) Work Center Abstraction with Parallel Machines:

- Original single machines (0–4) are grouped into work centers (WC0–WC4). Each WC

contains parallel machines (e.g., WC1 has 3 machines) to enable flexible resource allocation.

2) Dynamic Job Type-Specific Arrivals:

- Jobs are categorized into 10 distinct types (Type A–J), each assigned a stochastic arrival process (e.g., Poisson, periodic, uniform) to simulate real-world demand variability.
- Arrival parameters are calibrated to ensure the total job count matches the original instance (10 jobs) over a 100-job simulation horizon.

3) Online Scheduling Requirements:

- Jobs dynamically enter the system based on their type-specific distributions, requiring algorithms to handle incomplete information

These modifications retain the original instances' processing times and precedence logic while introducing dynamism (time-varying job arrivals) and flexibility (parallel machine assignment). The adapted benchmarks enable direct comparisons between static and dynamic scheduling policies under controlled parametric variations, bridging a critical gap in the literature.

Sub-Scenario 1B: Stochastic Full-Path Job Generation

Building on the dynamic framework of **Sub-Scenario 1A**, this sub-scenario generalizes the Lawrence benchmark structure by introducing stochastic job types, randomized processing times, and variable machine configurations, creating a more flexible and generally applicable testbed. Unlike the fixed job types and deterministic sequences in the classical Lawrence instances (e.g., as illustrated in **Table 3.2**), each simulation run in this sub-scenario generates a new, unique problem instance. This ensures that learned policies are not overfitted to specific job structures or processing time patterns.

To clarify the relationship with **Table 3.2**:

In **Sub-Scenario 1B**, **Table 3.2** would not be fixed. Instead, it is regenerated for every simulation run:

- The number of rows (i.e., job types) varies stochastically—e.g., between 5 and 20—so **Table 3.2** may have 5, 12, or 18 job-type blocks across different runs.
- The numeric entries (processing times) are sampled independently from a uniform distribution over $[1,99]$ for each eligible machine-operation pair, replacing the deterministic values shown in the static example.
- The machine configuration (number of machines per work center) may also vary, altering the number of columns in the table.

Key Enhancements for Generalization

To ensure that the proposed scheduling framework can generalize across diverse production scenarios, several key enhancements have been introduced to the simulation environment. These enhancements increase the variability and representativeness of the training data, allowing reinforcement learning agents to learn robust policies that can adapt to different job types, machine configurations, and operational uncertainties. The following modifications are implemented to support broader applicability and generalization capability.

1) Randomized Job Types:

- The number of job types A is generated stochastically (e.g., $A \sim \text{Uniform}\{5,6,\dots,20\}$), sampled from a discrete uniform distribution.
- Each job type τ_α is assigned a full-path routing, meaning its operations must visit all L work centers in a fixed sequence (e.g., $W_1 \to W_2 \to \cdots \to W_L$), preserving structural comparability with classical benchmarks while ensuring diversity across runs.

2) Stochastic Processing Times:

- For each operation $O_{i,j}$ of type τ_α, and for each machine $M_{l,k}$ in the required work center W_l, the processing time $P_{\alpha,j}^{l,k}$ is sampled independently from Uniform$\{1,2,\dots,99\}$.
- Once sampled, these values remain fixed throughout the simulation (i.e., no online uncertainty), but differ across instances, mimicking real-world variability in part families or process capabilities.

Sub-Scenario 1C: Flexible Routing with Partial Paths

This sub-scenario extends traditional job shop scheduling by replacing fixed, full-path routings with job-type-dependent partial paths, where each job type visits only a subset of the available work centers. Unlike classical benchmarks (e.g., Lawrence instances) or **Sub-Scenario 1B**—which enforce that all jobs traverse every work center in sequence—this framework allows operations to skip certain work centers entirely, better reflecting product-driven process variability in modern manufacturing.

1) Implementation of Partial Routing

For each job type τ_α, the routing is defined by a customized sequence of mandatory operations, each assigned to a specific work center W_l. The number of operations n_α satisfies $1 \le n_\alpha \le L$, and the mapping $w_{\alpha,j} \in \{1,\dots,L\}$ specifies which work center processes the j-th operation. Crucially, not all work centers need to appear in each job's path; for example, a job may route through $W_2 \to W_5 \to W_3$ while skipping W_1 and W_4. This is directly reflected in data structures such as the processing time table (as in **Table 3.2**): rows corresponding to a job type contain non-zero entries only for machines in its designated work centers, with other columns left empty (or marked infeasible).

2) Heterogeneous Work Center Utilization

Because of partial routing, work centers exhibit non-uniform utilization across job types. High-demand centers (e.g., CNC machining) may appear in 80% of job types, while specialized stations (e.g., laser marking) are used in only 15%. This induces dynamic bottlenecks and load imbalances, requiring schedulers to adaptively allocate capacity rather than rely on uniform workload assumptions.

Industrial Motivation

These routing flexibilities reflect actual practices in modern manufacturing, where process flows are increasingly tailored to product specifications and real-time quality feedback. For instance, in electronics assembly, non-premium smartphone models often skip burn-in testing to reduce cycle time without compromising reliability; similarly, generic pharmaceutical tablets may omit film coating when intended for bulk distribution, streamlining production. In aerospace manufacturing, composite components can bypass redundant non-destructive testing if in-line sensors already validate structural integrity. Such product- and condition-dependent pathways are emblematic of mass customization paradigms, where rigid, one-size-fits-all routings give way to adaptive, value-driven process chains—making partial routing not just a modeling convenience, but a necessity for realistic scheduling evaluation.

This sub-scenario extends the classical job shop model by introducing dynamic routing flexibility and heterogeneous work center utilization, addressing limitations in prior frameworks. The table below contrasts its innovations against Lawrence instances and Sub-**Scenario 1B**, emphasizing its alignment with modern industrial demands:

Table 3.3. Comparison of three sub-scenarios under Problem 1.

Aspect	Sub-Scenario 1A	Sub- Scenario 1B	Sub- Scenario 1C
Processing time	Fixed	Stochastic predefined	Stochastic predefined
Routing Flexibility	Fixed full-path	Predefined stochastic full-path	Predefined stochastic partial-path
Work Center Load	Uniform	Uniform	Variable by job type (heterogeneous)
Job Arrival Dynamics	Stochastic (Poisson)	Stochastic (Poisson)	Stochastic (Poisson)
Parallel Machines	Fixed identical	Variable identical	Variable identical
Industrial Relevance	Special assembly line	Generalized assembly scenarios	High (mass customization)
Complexity Level	Low	Medium	High

This sub-scenario significantly expands upon conventional job shop scheduling frameworks by introducing adaptive partial routing and variable work center utilization. **Sub-Scenario 1C** provides a robust testbed for evaluating advanced scheduling methods. It simulates realistic manufacturing contexts such as mass customization and product-specific process variability, enabling dynamic adaptation to complex operational requirements. Consequently, it is a critical bridge for applying scheduling theory within practical, flexible industrial settings.

3.2. Problem 2: Extended Model with Machine Breakdowns

The extension to incorporate machine breakdowns in **Scenario 2** addresses a specific limitation of **Scenario 1**'s sub-scenarios, namely, their assumption of uninterrupted machine availability. In real production systems, particularly in mass customization environments such as those modeled in **Sub-scenario 1C**, machines experience unplanned downtime at stochastic intervals, which directly disrupts processing and propagates delays across the shop floor. To reflect this, **Scenario 2** introduces random machine failures, where each machine

is subject to breakdowns occurring according to an exponential time-to-failure distribution (with mean time between failures calibrated to typical industrial values), followed by repair periods of fixed or distributed duration. When a machine in a mandatory work center fails, the affected operation must either be reassigned to another eligible machine within the same work center, if one is available, or wait until repairs are completed. This not only delays the job in question but also causes downstream operations to stall, leading to cumulative tardiness and potential congestion in subsequent work centers. By embedding such disruptions into the scheduling environment, **Scenario 2** explicitly captures the interplay between machine reliability, dynamic routing decisions, and performance degradation under resource unavailability. This formulation aligns with key characteristics of modern flexible manufacturing: high routing variability, sensitivity to downtime frequency, and tight delivery requirements inherent in mass customization contexts.

The following sections show the extension of the mathematical models with machine breakdown events.

3.2.1. Dynamic Event Extension

In **Scenario 2**, the FJSSP incorporates machine breakdown events, significantly enhancing the realism of the dynamic scheduling environment. Machine availability is no longer continuously guaranteed; instead, it is dynamically represented by binary state variables indicating operational or breakdown status:

$$\text{MachineStatus}_{l,k}(t)\begin{cases}1, \text{if machine } M_{l,k} \text{ is operational at time } t, \\ 0, \text{if machine } M_{l,k} \text{ is under breakdown at time } t.\end{cases} \quad (3.12)$$

Machine breakdown events are modeled through stochastic events characterized by Mean Time Between Failures (MTBF) and Mean Time to Repair (MTTR). The machine failure events are specifically triggered when the cumulative working time of a machine reaches the generated Failure Interval, which follows an exponential distribution:

$$\text{Failure Interval}_{l,k} \sim \text{Exponential}\left(\frac{1}{MTBF_{l,k}}\right). \tag{3.13}$$

Similarly, repair durations after each breakdown event are modeled as exponentially distributed random variables:

$$\text{Repair Duration}_{l,k} \sim \text{Exponential}\left(\frac{1}{MTTR_{l,k}}\right). \tag{3.14}$$

This approach effectively captures the unpredictable nature of machine breakdowns and repair durations typical in manufacturing scenarios.

3.2.2. Updated Constraints

Integrating machine breakdowns into the scheduling model introduces critical modifications to constraints:

- Operation Scheduling during Machine Breakdowns: Operations cannot be scheduled on machines experiencing downtime. This restriction is formally represented by:

$$X_{i,j}^{l,k}(t) \leq \text{MachineStatus}_{l,k}(t), \forall i, j, l, k, t. \tag{3.15}$$

where $X_{i,j}^{l,k}(t)$ is a binary decision variable determining if operation j of job i is assigned to machine k in work center l at time t.

- Dynamic Rescheduling: When an operation is interrupted by a machine breakdown, it is returned to the waiting queue of its designated work center. Upon any change in machine availability—such as the completion of a repair—the scheduling agent recomputes a priority score for each job in the queue using a learned linear policy over job-specific features (e.g., time-to-due, accumulated waiting time, remaining workload, and processing duration). The queue is then sorted in descending order of this score, and

idle machines are assigned jobs sequentially from the front of the reordered queue. If multiple repair events share the same completion time, they are processed in the order in which the corresponding breakdowns originally occurred; for breakdowns that occurred simultaneously, machine identifiers are used as a secondary tie-breaking criterion to ensure deterministic and reproducible execution.

$$S_{i,j}^{l,k} \geq t + \text{Repair Duration}_{l,k}, \qquad \text{if MachineStatus}_{l,k}(t) = 0. \tag{3.16}$$

This ensures that operations begin only when machines are fully operational.

3.2.3. Objective Consistency

Although **Scenario 2** introduces stochastic machine breakdowns, thereby transforming the problem into a partially observable stochastic scheduling environment, the primary objective remains the minimization of mean tardiness, as defined in **Equation (3.5)**. This consistency ensures cross-scenario comparability and reflects the industrial priority of on-time delivery in make-to-order and mass-customization settings.

Nevertheless, machine unreliability introduces an implicit trade-off between nominal scheduling efficiency and operational robustness. A policy optimized solely for ideal conditions may suffer severe performance degradation under disruptions, whereas one that strategically exploits routing flexibility or preserves temporal slack can mitigate failure-induced delays. While the objective function does not explicitly penalize reliability-related factors such as downtime cost or rescheduling frequency, their effects are endogenously captured through realized job completion times. Thus, mean tardiness serves as an emergent proxy that aggregates both scheduling performance and resilience, enabling the reinforcement learning agent to implicitly balance efficiency and robustness even under a scalar reward signal.

3.3. Problem 3: Multi-Objective Optimization with Sequence-Dependent Setups

The extension builds upon the stochastic scheduling model of **Scenario 2** by introducing multi-objective optimization that explicitly accounts for sequence-dependent setup times. This addresses a key limitation of **Scenarios 1** and **2**, which assume zero or job-independent setups and therefore overlook the operational trade-offs arising from the order in which jobs are processed on a machine. Real-world manufacturing systems often encounter sequence-dependent setup times (e.g., cleaning, tool changing between jobs) that significantly impact efficiency, energy consumption, and operational costs—factors omitted in prior models.

While several secondary objectives could be considered, such as makespan, total flow time, energy consumption, or workload balance, we selected total setup time as the most representative and actionable complement to tardiness minimization. Setup time directly correlates with idle machine time, reconfiguration costs, and energy use, yet it remains measurable using only job-type transition matrices without requiring detailed power profiles or complex environmental data. Moreover, in dynamic environments where rescheduling occurs frequently due to machine breakdowns, uncontrolled setup accumulation can severely degrade productivity even if tardiness appears low. Thus, minimizing total setup time serves as an effective proxy for internal operational efficiency and schedule stability.

By integrating these setups, **Scenario 3** captures the fundamental trade-off between two critical dimensions: external responsiveness (measured by average tardiness) and internal efficiency (captured by total setup time). For instance, scheduling jobs with similar setups consecutively on a machine may reduce energy costs and machine wear but could delay high-priority incoming jobs, worsening tardiness. This tension is especially pronounced in **Scenario 2**'s breakdown-prone environments, where repair-induced rescheduling often fragments job sequences and triggers cascading setup penalties.

The multi-objective framework enables decision-makers to evaluate scheduling policies that effectively balance productivity (e.g., low setup-induced idle time), sustainability (e.g., reduced energy consumption), and reliability (e.g., consistent on-time delivery under disruptions). This is especially critical in industries such as automotive and electronics, where both setup times and due-date performance directly influence profitability and customer satisfaction. By bridging the gap between dynamic scheduling, machine reliability, and sequence-dependent logistics, **Scenario 3** transforms the model into a general-purpose tool for optimizing modern manufacturing systems under multifactor real-world pressures.

3.3.1. Sequence-Dependent Setup Time Modeling

In dynamic flexible job shop scheduling, sequence-dependent setup times significantly influence system efficiency, particularly in manufacturing scenarios where the transition from one job type to another requires reconfiguration of machinery, tool adjustments, or cleaning processes. In **Scenarios 1** and **2**, setup times are assumed to be either negligible or constant, independent of the order in which jobs are processed. **Scenario 3** removes this simplification by explicitly modeling sequence-dependent setup times, where the setup duration between two jobs on a machine is determined by the specific combination of the preceding and succeeding operations.

The setup time between consecutive operations on the same machine is defined as $ST_{ij,i'j'}^{l,k}$, which represents the setup time required on machine k in work center l when transitioning from operation j of job i to operation j' of job i'.

Additional Scheduling Constraints:

- A machine cannot start processing a new job until the setup for the transition is complete.
- Setup activities are non-preemptive and must be completed once started.

To enforce the temporal precedence between consecutive operations on the same machine, the following constraint is applied:

$$C_{i'j'} \geq C_{ij} + ST^{l,k}_{ij,i'j'} + P^{lk}_{i'j'}, \tag{3.17}$$

for consecutive operations (ij) to $(i'j')$ on machine k in work center l.

3.3.2. Multi-Objective Formulation

This scenario introduces a multi-objective optimization framework to simultaneously consider conflicting objectives:

- Primary Objective: Minimize the average tardiness of all jobs.

$$\text{Minimize } \frac{1}{N}\sum_{i=1}^{N} T_i, \tag{3.18}$$

- Secondary Objective: Minimize the total setup time across all machines and work centers.

$$\text{Minimize } \sum_{l=1}^{L}\sum_{k=1}^{M_l}\sum_{(ij,i'j')} ST^{l,k}_{ij,i'j'}. \tag{3.19}$$

3.3.3. Dynamic Event Integration

Scenario 3 incorporates dynamic events, including random job arrivals and machine breakdowns, which further complicate the optimization problem. The presence of dynamic events introduces additional uncertainty and necessitates real-time adjustments to maintain system performance.

- **Random Job Arrivals:** Modeled through a Poisson process like **Scenario 1**, with jobs dynamically entering the system based on stochastic arrival rates specific to each job type.

- **Machine Breakdowns:** Following **Scenario 2**, machines experience random breakdowns and repairs, modeled with MTBF and MTTR both following exponential distributions.

These dynamics highlight conflicts between the two objectives in real time:

- Minimizing total setup time encourages sequencing jobs of similar types consecutively but may increase tardiness for urgent incoming jobs.
- Prioritizing the reduction of average tardiness may lead to frequent setup changes, thus increasing the total setup time and operational costs.

This trade-off becomes particularly important under dynamic disruptions, emphasizing the requirements for adaptive scheduling solutions that effectively balance efficiency and responsiveness in real-time operations.

Chapter 4. Problem 1: Baseline Dynamic Scheduling under Random Job Arrivals with RL

This chapter builds upon the formulation of the DFJSP presented in the previous chapter. In this chapter, the problem was mathematically modeled using an MDP, incorporating key aspects of modern manufacturing environments such as random job arrivals, machine availability, and routing flexibility. Transitioning from theoretical modeling to practical implementation, this chapter introduces a decentralized, event-driven scheduling framework in which DDQN agents are embedded within each work center. These agents make independent scheduling decisions based on local information, enabling real-time responsiveness and system-wide adaptability. The proposed framework aims to minimize mean tardiness while supporting scalable decision-making aligned with the principles of Industry 4.0. The remainder of this chapter is structured as follows:

Section 4.1 details the system architecture and outlines the DDQN algorithm, including its neural network structure, experience replay mechanism, and target network stabilization, along with the full decision loop—state observation, action selection, reward computation, and network updates. The framework is then rigorously evaluated in **Section 4.2** through numerical experiments across three problem instances using both benchmark and randomized datasets. Performance is assessed against conventional dispatching rules using metrics such as average tardiness and win rates, while SHAP (SHapley Additive exPlanations) values are employed to interpret agent decisions and quantify the influence of key state features. **Section 4.3** discusses the practical implications of the study, emphasizing the framework's interpretability, scalability, and suitability for smart manufacturing environments. Finally, **Section 4.4** summarizes the key findings, reaffirming the DDQN-based approach's superior performance over traditional heuristics and its robustness in handling uncertainty and dynamism in flexible job shop scheduling.

4.1. Methodology

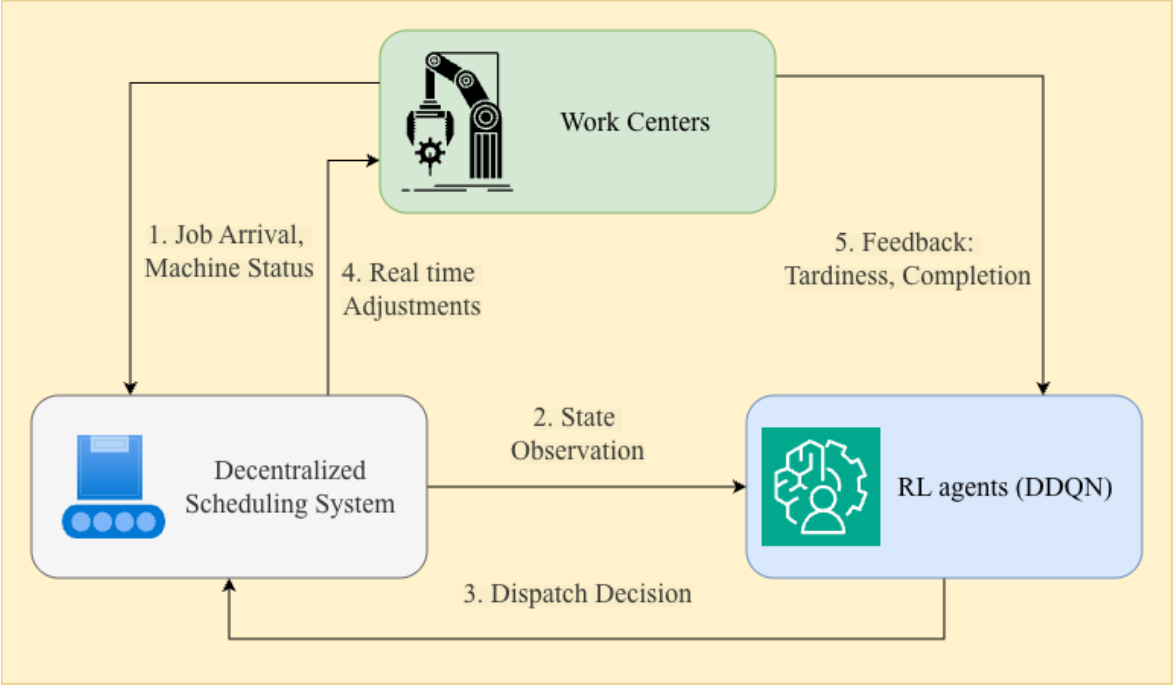


Figure 4.1. Suggested multi-agent, event-driven reinforcement learning framework for DFJSP.

The research utilized the DDQN method within a decentralized, event-driven framework to address the DFJSP, as shown in **Figure 4.1**. The decentralized decision-making model allows each work center to dispatch jobs independently. Decentralized decisions are made in near real-time when jobs start arriving or machines become ready, thus ensuring adaptive schedules. This event-driven structure maximizes machine utilization: jobs are assigned immediately and directly to idle machines through this approach. Every work center employs a DDQN-based agent that schedules jobs and allocates machines based solely on local observations, using a policy optimized to minimize mean tardiness while adapting to dynamic disruptions such as machine breakdowns and sequence-dependent setups. Through continued interaction with the environment, the agents progressively refine their policies to enhance scheduling performance. This approach reduces computational burdens compared to centralized systems, thus improving scalability and making it suitable for large-scale,

dynamic manufacturing environments typical of Industry 4.0.

4.1.1. Architecture of a Distributed Event-Driven Decision-Making System

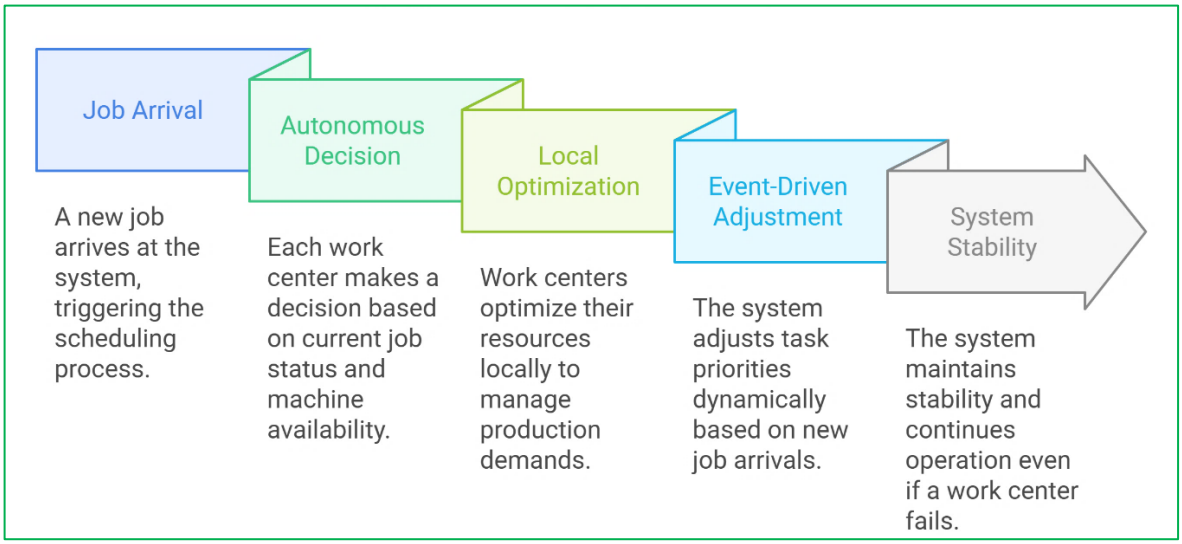


Figure 4.2. Distributed scheduling process with event-driven decision-making.

Figure 4.2 depicts RL-based DFJSP, and several work centers run autonomously using parallel processors while considering the distributed scheduling system. To avoid the bottlenecks associated with centralized systems, independent scheduling of each work center is carried out based on local job status and machine availability. Event-driven methodology provides real-time responsiveness, initiating decisions dynamically with changes in machine availability or work arrivals. The structure aims to reduce job tardiness, maximize resource usage, and minimize redundant calculations. It also offers stability and scalability, with the flexibility to accommodate dynamic production environments by allowing the introduction of new machines or work centers without extensive architectural redesign. The study employs an event-driven approach to address the uncertainty of job arrivals in a DFJSP setting.

4.1.2. Reinforcement Learning Outline

In DFJSP, every work center operates autonomously to allocate tasks efficiently in real-time within the framework's distributed, event-driven architecture. Through interaction with the dynamic environment and focusing on minimizing mean delay, the DDQN agents learn the optimal scheduling protocols. The system adapts to unpredictable job arrivals and varying machine availability by maximizing the expected cumulative reward. In dynamic industrial settings, this real-time decision-making approach ensures responsiveness, reduces delays, and enhances overall scheduling effectiveness.

DDQN Outline

In reinforcement learning (RL), an agent learns to make sequential decisions by interacting with an environment. A fundamental challenge is evaluating how good it is to take a particular action in a given state. This evaluation should not focus solely on immediate reward but also consider the total future return over time. The state-action value function, commonly denoted as $Q(\boldsymbol{s}, a)$, provides this capability: it estimates the expected total discounted reward the agent will receive if it starts in state $\boldsymbol{s}$, takes action a, and then follows a specific policy thereafter. This function is central to RL because it allows the agent to compare actions directly, enabling it to select the one with the highest future payoff. Without such a predictive mechanism, the agent would have no basis for choosing between actions beyond their immediate outcomes, making it impossible to learn optimal behavior in settings like scheduling, where the consequences of a decision may only become apparent many steps later.

In reinforcement learning, the goal of an agent is to learn a policy that maximizes the expected cumulative reward over time. Central to this objective is the state action value function, also known as the Q-function, defined as:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{t} \mid \boldsymbol{s_0} = \boldsymbol{s}, a_0 = a\right], \tag{4.1}$$

where π is the policy, $\gamma \in [0,1)$ is the discount factor, and r_t is the reward at time t. The optimal Q-function, $Q^{*}(\boldsymbol{s}, a)$, satisfies the Bellman optimality equation and yields the best possible action in any state.

In traditional tabular Q-learning, $Q(\boldsymbol{s}, a)$ is stored in a lookup table. However, this becomes infeasible in large or continuous state spaces, such as those encountered in dynamic job shop scheduling. To overcome this, Deep Q-Networks (DQN) [81] approximate the Q-function using a neural network $Q_{\theta}(\boldsymbol{s}, a)$, parameterized by weights θ, which maps raw state inputs to estimated Q-values for all actions.

Despite its success, standard DQN suffers from overestimation bias: because the same network is used both to select and evaluate actions in the target computation ($y_t = r_t + \gamma \max_{a'} Q_{\theta}(\boldsymbol{s_{t+1}}, a')$), it tends to overestimate Q-values, leading to unstable or suboptimal policies.

The Double DQN (DDQN) algorithm [140] addresses this by decoupling action selection from value estimation. Specifically, the target value is computed as:

$$y_t = r_t + \gamma \cdot Q_{\theta^-}(\boldsymbol{s_{t+1}}, \arg\max_{a'} Q_{\theta}(\boldsymbol{s_{t+1}}, a')), \tag{4.2}$$

where:

Q_{θ} (online network) is used to select the best action in the next state,

Q_{θ^-} (target network) is used to evaluate the value of that action.

The target network has the same architecture as the online network, but its parameters θ^- are updated less frequently. Specifically, they are copied directly from the online network every C steps using hard updating ($\theta^- \leftarrow \theta$). This approach stabilizes training by reducing temporal correlation in the target values. Both networks are initialized

randomly before training begins, for example using Xavier initialization [141].

To further improve stability and sample efficiency, DDQN employs an experience replay buffer D. All transitions $(\boldsymbol{s_t}, a_t, r_t, \boldsymbol{s_{t+1}})$ are stored in this buffer, and mini-batches are sampled uniformly at random during training. This breaks the temporal correlation between consecutive samples, which is a common source of divergence in online learning, and allows experiences to be reused multiple times.

The network parameters θ are updated by minimizing the mean squared temporal difference (TD) error:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}[(y - Q_\theta(\boldsymbol{s}, a))^2], \tag{4.3}$$

where y is the DDQN target defined above. In practice, the expectation is approximated using a mini-batch of size B, and gradients are computed via backpropagation with an optimizer such as Adam.

While theoretical convergence guarantees for deep RL remain limited due to the use of nonlinear function approximators, the combination of experience replay, target networks, and the double estimator in DDQN has been shown empirically to yield stable and robust learning across a wide range of control and decision-making tasks—including complex scheduling environments like the DFJSP considered here.

Algorithm 1 summarizes the full DDQN-based scheduling procedure used in this work, integrating the components described above with our domain-specific state, action, and reward design.

Algorithm 1: DDQN-Based Scheduling

Initialize: State Space S, action space A, learning rate α, discount factor γ, exploration rate ε, replay buffer size N, batch size B, target update interval C.

Initialize online Q-network $Q_\theta(\boldsymbol{s}, a)$ and target network $Q_{\theta^-}(s, a) \leftarrow Q_\theta$

Initialize empty replay buffer $D \leftarrow \emptyset$

For each episode = 1 to max_episodes do

Set $\varepsilon \leftarrow \max(\varepsilon_{\min}, \varepsilon \cdot \delta_\varepsilon)$ *(exponential decay)*

Observe initial state $\boldsymbol{s_0}$

For each step t = 0, 1, … until episode terminates do

With probability ε, select random action $a_t \in A$;

otherwise: $a_t \leftarrow \arg\max_a Q_\theta(\boldsymbol{s_t}, a)$

Execute a_t , observe reward r_t and next state $\boldsymbol{s_{t+1}}$

Store transition $(\boldsymbol{s_t}, a_t, r_t, \boldsymbol{s_{t+1}})$ in D

If $|D| \geq B$

Sample mini-batch $\{(\boldsymbol{s_i}, a_i, r_i, \boldsymbol{s_{i+1}})\}_{i=1}^{B}$ from D

Compute target: $y_i = r_i + \gamma \cdot Q_{\theta^-}(\boldsymbol{s_{i+1}}, \arg\max_{a'} Q_\theta(\boldsymbol{s_{i+1}}, a'))$

Update θ by minimizing: $L(\theta) = \frac{1}{B}\sum_{i=1}^{B}[Q_\theta(\boldsymbol{s_i}, a_i) - y_i]^2$

If $t \bmod C = 0$ then

Update target network: $\theta^- \leftarrow \theta$

State Representation

A layered structure captures system-level, job-level, and machine-level information:

- **System-Level:** Number of parallel machines f_1, number of waiting jobs f_2.
- **Job-Level:** Due date statistics (mean f_3, min f_4, max f_5, std f_6), waiting time (mean f_7, min f_8, max f_9, std f_{10}), flow time (mean f_{11}, min f_{12}, max f_{13}, std f_{14}), remaining operations (mean f_{15}, min f_{16}, max f_{17}, std f_{18}), processing time (mean f_{19}, min f_{20}, max f_{21}, std f_{22}), remaining processing time (mean f_{23}, min f_{24}, max f_{25}, std f_{26}).
- **Machine-Level:** Earliest machine available time (mean f_{27}, min f_{28}, max f_{29}, std f_{30}).

Action Space

The DDQN selects job sequences using dispatching rules: FIFO, LIFO, EDD, LDD, SPT, LOR, and ERD.

Reward Function

The reward measures changes in average modified time to due date before and after the action: $r = \overline{\Delta d_{after}} - \overline{\Delta d_{before}}$, $\Delta d_i = \begin{cases} d_i - ECT_i, & d_i < ECT_i \\ k(d_i - ECT_i), & d_i \geq ECT_i \end{cases}$ where d_i is the due date, ECT_i is the expected completion time, and k is a penalty factor.

Figure 4.3. State, action, and reward representations.

The reinforcement learning framework is built upon three core components: state representation, action space, and reward function, all of which are designed to reflect the operational realities of dynamic flexible job shop scheduling (DFJSP). These elements are illustrated in **Figure 4.3** and described in detail below.

State Representation

The agent observes a high-dimensional but structured state vector that encodes real-time information from three hierarchical levels of the production system:

- **System-level features** capture global workload and urgency, including the number of waiting jobs, statistical summaries (mean, minimum, maximum) of due dates, flow times, remaining operations, and processing times across all jobs, as well as machine load metrics.
- **Job-level features** describe the characteristics of the candidate's job under consideration at the current decision point, such as its due date, total processing time, and expected completion time (ECT).

- **Machine-level features** reflect resource availability, including the earliest available time of each eligible machine and its recent utilization pattern.

  All continuous features are normalized to the range $[0,1]$ using min-max scaling based on historical bounds observed during preliminary simulations. This normalization stabilizes neural network training and ensures consistent gradient behavior across episodes.

Action Space

Critically, the action space is defined not as raw machine-job assignments (which would be combinatorially large and environment-specific), but as a set of eight interpretable, rule-based dispatching policies commonly used in industrial practice:

- FIFO (First In, First Out): prioritizes jobs by arrival order;
- LIFO (Last In, First Out): favors recently arrived jobs;
- EDD (Earliest Due Date): selects the job with the nearest deadline;
- LDD (Latest Due Date): prioritizes jobs with distant deadlines;
- SPT (Shortest Processing Time): chooses the quickest-to-complete job;
- LPT (Longest Processing Time): selects the most time-consuming job;
- LOR (Least Operations Remaining): favors jobs close to completion;
- ERD (Earliest Release Date): processes jobs that became available earliest.

At each decision epoch, the agent selects one of these rules, which is then executed by the local work center to determine the next job-machine assignment. This formulation transforms the complex sequencing problem into a discrete, finite-action decision task, enabling efficient learning while preserving interpretability: the learned policy can be understood as a dynamic switching mechanism among established heuristics, rather than an opaque end-to-end mapping. Moreover, this design ensures scalability; adding new machines or job types does not expand the action space, as the same rule set remains applicable.

Reward Function

The reward signal is designed to directly incentivize the minimization of job tardiness, a key performance indicator in manufacturing. Specifically, the agent receives a scalar reward based on the change in the average Modified Due Date before and after its action. The Modified Due Date for a job is defined as $\max\{d,\text{ECT}\}$, where d is the due date and ECT is the expected completion time. The instantaneous reward is computed as:

$$\Delta k_t = \begin{cases} \dfrac{d - \text{ECT}}{k_t}, & \text{if } d < \text{ECT(tardy case)} \\ k_t - \text{ECT}, & \text{if } d \geq \text{ECT(on-time case)} \end{cases} \tag{4.4}$$

where $k_t > 0$ is a time-dependent penalty factor that increases with system congestion to emphasize urgency during peak loads. A higher (less negative) reward indicates better scheduling performance. This formulation encourages the agent to not only avoid delays but also complete jobs as early as possible when deadlines are not binding, thereby improving overall shop floor responsiveness.

Together, this state-action-reward design enables the DDQN agent to learn a robust, adaptive scheduling policy that responds effectively to stochastic job arrivals, machine breakdowns, and varying due date tightness—all while maintaining transparency through its reliance on well-known dispatching logic.

4.2. Numerical Experiments

The experiments consist of two parts: First, training DDQN agents used in the distributed decision-making system. Second, to verify the effectiveness of the trained DDQN, its performance is compared against different dispatching rules across various random problems, job release densities, and configurations.

Figure 4.4 illustrates the complete process of training and testing a DDQN agent within a

distributed decision-making framework designed for the DFJSP. This process encompasses three main phases: training, production simulation, and testing. Throughout the training and testing phases, the work centers in the FJSSP maintain similar structures and scopes, enabling the agents to function under identical parameter settings.

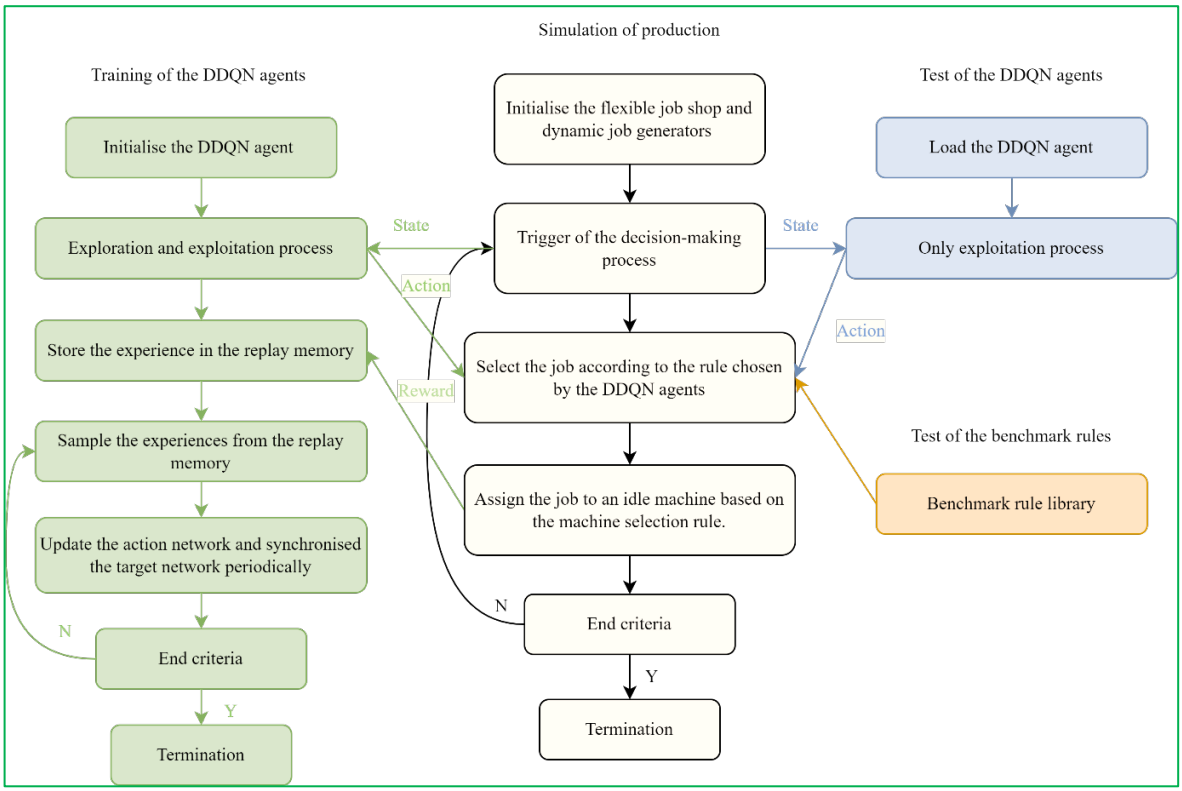


Figure 4.4. Workflow of DDQN-based scheduling and benchmark comparison.

Several key parameters define the DDQN agent and critically influence its learning dynamics and final performance. The agent selects from an action space of size 7 at each decision step, corresponding to seven dispatching rules, and processes a 32-dimensional state vector that encodes machine statuses, job priorities, and temporal features of the shop floor. For function approximation, we evaluated neural network architectures ranging from shallow (e.g., two hidden layers with widths [64, 128]) to deeper configurations (up to five layers with widths [32, 64, 128, 256, 128]). The final choice—a four-layer network with widths [64, 128, 256, 128]—was selected because it provided sufficient capacity to model complex scheduling heuristics without introducing excessive variance or overfitting. All

layers employ the Leaky ReLU activation function [142] with a negative slope of 0.01, which maintains small gradient flow for negative inputs and effectively mitigates the vanishing gradient problem during backpropagation.

Optimization is performed using the Adam optimizer [143], which adapts per-parameter learning rates based on gradient moments to improve stability and convergence speed. We tested learning rates in the set $\{3 \times 10^{-3}, 10^{-4}, 3 \times 10^{-4}, 10^{-5}, 3 \times 10^{-5}\}$; the final value of 3×10^{-5} was chosen because higher rates led to unstable Q-value estimates and occasional divergence—particularly during early training when rewards are sparse—while lower rates resulted in unacceptably slow learning over the 20,000-episode horizon. The loss function is mean squared error (MSE) between predicted and target Q-values, a standard and well-behaved choice for regression-based value learning.

The discount factor γ was fixed at 0.99 after comparing values in $\{0.9, 0.95, 0.99\}$; this high value emphasizes long-term performance (e.g., minimizing total tardiness) over immediate rewards, which is essential in scheduling contexts with delayed feedback.

Exploration follows an ε-greedy policy, where ε is decayed exponentially after every decision step as

$$\varepsilon_t = \max(\varepsilon_{\min}, \varepsilon_0 \cdot \rho^t), \tag{4.5}$$

with $\varepsilon_0 = 1.0$ and $\varepsilon_{\min} = 0.01$. Rather than tuning the decay factor ρ through blind search, we calibrated it based on the total training budget of 20,000 episodes (each containing 100 decision steps, i.e., 2 million steps in total). Specifically, we selected $\rho = 0.999997$ so that ε reaches its minimum of 0.01 only after approximately 1.54 million steps—about 77% through training. This ensures sustained exploration across diverse job arrival scenarios, which is crucial in stochastic environments. In preliminary experiments, faster decays (e.g., reaching $\varepsilon = 0.01$ within the first few thousand steps) caused premature

convergence to suboptimal policies.

For experience replay, we evaluated buffer sizes in $\{5000, 10000,20000,50000\}$ and batch sizes in $\{62,128,256,512\}$. Given our hardware constraints (8 GB RAM) and transition dimensionality, a replay buffer of 20,000 and mini-batch size of 256 were selected: smaller buffers led to high sample correlation and policy instability under non-stationary job arrivals, while larger batches increased computational overhead without significant gains in final performance.

Finally, the target network update interval was tuned over $\{50,100,200\}$ episodes. Updating every 100 episodes provided the best trade-off: more frequent updates introduced oscillations due to rapidly shifting Q-estimates, while less frequent updates delayed the propagation of learned knowledge.

The simulation is based on Python and utilizes a DDQN agent developed with the PyTorch machine-learning library [144]. Training and testing occur on a home computer with an Intel Core i5-8250U CPU running at 1.8 GHz and featuring 8 GB of RAM. Firstly, the parameters of the training procedure are presented, followed by a demonstration of the resulting outputs. Next, the performance of the trained agents across various random instances of the problem is compared, modifying the job release density and configuration. The results are evaluated against the benchmark dispatching rules applied to the same problem instances.

4.2.1. Training

The DDQN agent for the DFJSP was trained using a decentralized, event-driven simulation framework. It was designed to enhance job allocation and machine utilization according to specific parameters. This section details the configuration and essential settings employed in the training process.

The datasets used in this study consist of three distinct cases, each evaluating the proposed

scheduling framework's performance under varying levels of complexity and constraints. The parameter settings for each case are summarized in **Table 4.1**.

- **Case 1A** uses the Lawrence instances la01–la05 as job templates to generate dynamic scheduling problems. Training employs stochastically varied instances, while testing is conducted on fixed scenarios (with identical random seeds) derived from the same templates to ensure reproducible benchmarking.
- **Case 1B** introduces a more general-purpose environment where job types and processing routes are randomly generated within defined constraints. It also includes an extended set of scheduling actions.
- **Case 1C** further increases the complexity by allowing flexible paths, where job types do not need to visit every work center, and additional state features are considered.

Table 4.1. Parameter setting of the training environment.

Case	Case 1A	Case 1B	Case 1C
Job Instances	la01, la02, la03, la04, la05	Randomly generated with full paths	Randomly generated with flexible paths
Job Types Number	10	[1, 15]	[1, 15]
Work Centers Number	5	5	5
Parallel Machine Number	[1, 2, 3, 4, 5]	[1, 5]	[1, 5]
Job Release Pattern	Exponential distribution	Exponential distribution	Exponential distribution
Mean Interval	[5, 15]	[5, 15]	[5, 15]
Job Number	100	100	100
Episode Number	20,000	20,000	20,000
Processing Time	[1, 99]	[1, 99]	[1, 99]

Note: All intervals denoted as $[a, b]$ indicate that integer values are sampled uniformly at random from the inclusive discrete set $\{a, a+1, \ldots, b\}$. For Parallel Machine Number in **Case 1A**, the list $[1,2,3,4,5]$ means the number of parallel machines per work center is independently drawn with equal probability from this finite set.

The training time for the DDQN model across all three cases was approximately 69,499.72 seconds.

Figures 4.5, **4.6**, and **4.7** depict the average training loss per episode for the DDQN agents across all work centers in **Cases 1A**, **1B**, and **1C**, respectively. The loss is computed as the mean squared error between predicted and target Q-values during each training step, serving as a measure of policy learning progress. A consistent decline in loss over time indicates successful convergence of the agents' policies, reflecting their ability to minimize prediction errors and improve scheduling decisions through experience.

In the early stages of training, the agents prioritize exploration, leading to higher initial loss values and greater variability due to frequent state-action sampling. As training progresses, the focus shifts toward exploitation, where learned policies are increasingly utilized, resulting in smoother and more stable loss curves. Across all three cases, the overall trend shows a steady reduction in loss, demonstrating effective learning and robustness of the decentralized reinforcement learning framework.

Comparing the three cases, the loss trajectories in **Figure 4.5** (**Case 1A**) and **Figure 4.6** (**Case 1B**) exhibit similar patterns: rapid initial decrease followed by gradual stabilization, with minor fluctuations attributable to environmental dynamics and stochastic exploration. These results confirm that the DDQN agents effectively learn optimal scheduling strategies under standard flexible job shop conditions.

In contrast, **Figure 4.7** (**Case 1C**) reveals a distinct behavior, where the loss curve intermittently drops to zero or exhibits flat segments. This phenomenon arises from the structural design of **Case 1C**, which introduces job-type-dependent partial routings: unlike classical benchmarks (e.g., Lawrence instances) or **Case 1B**, where jobs visit all work centers sequentially, each job type in **Case 1C** only traverses a subset of available centers. Consequently, during certain episodes, no jobs are routed through specific work centers,

resulting in no state-action transitions and thus no Q-value updates. In such episodes, the loss is either undefined or recorded as zero, manifesting as oscillations or plateaus at zero in the loss curve.

Importantly, these apparent "gaps" do not indicate training instability but rather reflect the inherent sparsity of the environment due to flexible routing logic. This behavior aligns with real-world manufacturing scenarios, where not all resources are engaged in every production cycle. The fact that agents still achieve stable convergence despite such sparsity underscores the adaptability and effectiveness of the proposed DDQN-based approach in handling complex, dynamic, and heterogeneous scheduling environments.

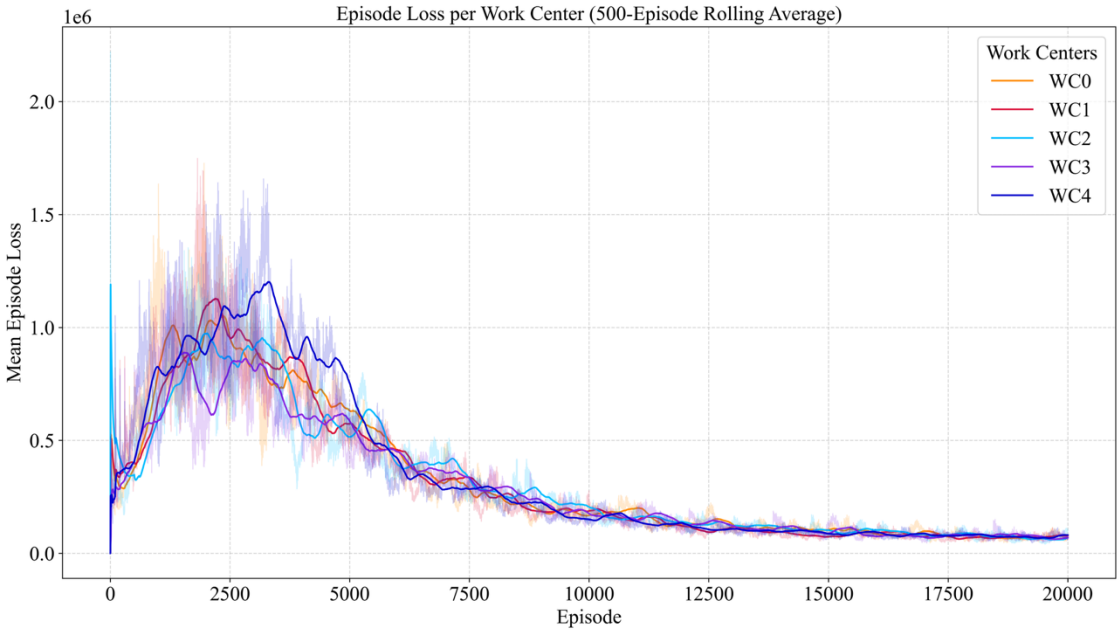


Figure 4.5. Mean episode loss per work center in **Case 1A**, computed as a 500-episode rolling average. The plot shows the training convergence of DDQN agents across five work centers (WC0–WC4), with each line representing the average loss over time.

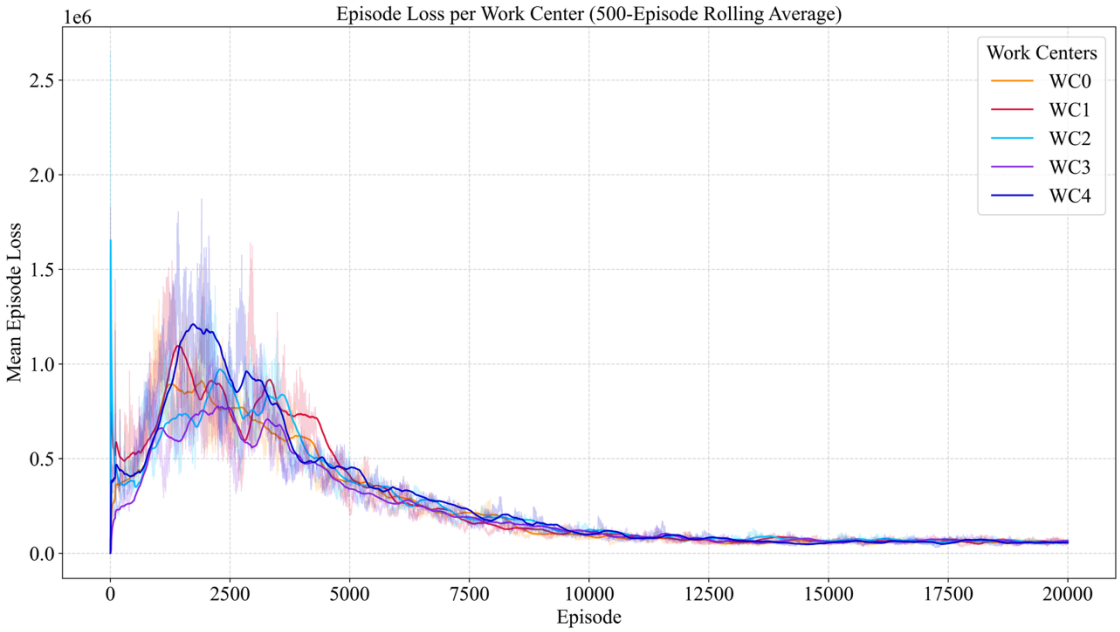


Figure 4.6. Mean episode loss per work center in **Case 1B**, computed as a 500-episode rolling average. The plot shows the training convergence of DDQN agents across five work centers (WC0–WC4), with each line representing the average loss over time.

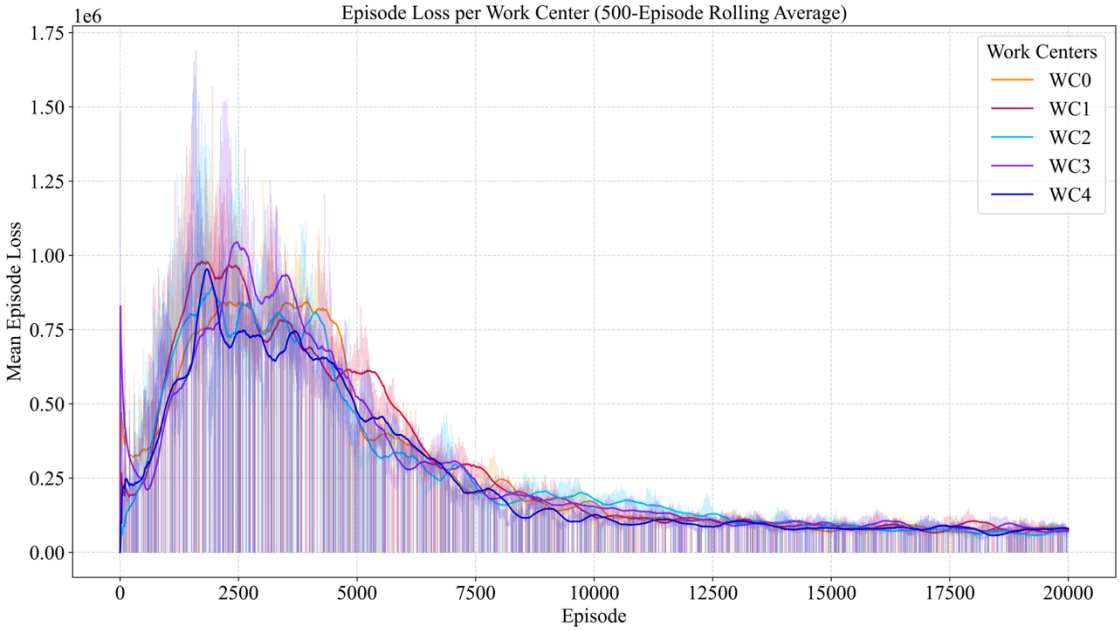


Figure 4.7. Mean episode loss per work center in **Case 1C**, computed as a 500-episode rolling average. The plot shows the training convergence of DDQN agents across five work centers (WC0–WC4), with each line representing the average loss over time.

4.2.2. Testing Results and Statistical Analysis

Following the training of the DDQN agents, their performance was tested in various test scenarios to determine robustness and adaptability. Benchmark studies compare the DDQN-based scheduling agent with conventional dispatching rules. The evaluation focuses on minimizing mean tardiness and assessing consistency under varying job release patterns and system configurations.

The rules used for testing are First In, First Out (FIFO), which executes tasks in their arrival order, and Last In, First Out (LIFO), which performs the newest arriving tasks first. Earliest Due Date (EDD) first treats tasks with the earliest deadlines, and Largest Due date (LDD) first deals with functions with the latest ones. The Shortest Processing Time (SPT) rule chooses tasks with the shortest processing times, while the Longest Processing Time (LPT) rule chooses tasks with the longest processing times. Least Operation Remaining (LOR) handles tasks with the least number of operations remaining, and Earliest Release Date (ERD) selects available jobs for processing as early as possible based on their release times.

Table 4.2 summarizes the configuration of the testing environments. All three cases are evaluated on 100 independently generated test instances, produced using random seeds 1 to 100 to ensure reproducibility and statistical reliability.

Each test instance respects the respective problem structure defined in **Section 3.1** and **Table 4.1**: **Case 1A** follows the Lawrence benchmark topology, **Case 1B** employs full-path routings with variable job-type counts, and **Case 1C** allows partial routings where jobs may skip work centers. Across all cases, processing times are assigned as independent uniform random integers from $[1,99]$. Job inter-arrival times follow an exponential distribution, where the mean parameter is itself sampled uniformly at random from the integer interval $[5,15]$ for each test instance. This design introduces controlled stochasticity in both timing and system structure while preserving the core routing logic of each case. All other

parameters align with the training configuration (**Table 4.1**).

Table 4.2. Testing environment parameters for three cases.

Parameter	Case 1A	Case 1B	Case 1C
Number of Episodes	20000	20000	20000
Total Number of Jobs	100	100	100
Mean Interval Range	[5, 15]	[5, 15]	[5, 15]
Number of Machines per Work Center	[1, 2, 3, 4, 5]	[1, 5]	[1, 5]
Number of Work Centers	5	5	5
Number of Job Types Range	10	[1, 15]	[1, 15]
Number of Operations	Fixed sequence through each work center	Each job type goes through every work center once	Each job type may skip some work centers
Processing Time Range	[1, 99]	[1, 99]	[1, 99]
Actions	['FIFO', 'LIFO', 'EDD', 'LDD', 'SPT', 'LPT', 'LOR', 'ERD']	['FIFO', 'LIFO', 'EDD', 'LDD', 'SPT', 'LPT', 'LOR', 'ERD']	['FIFO', 'LIFO', 'EDD', 'LDD', 'SPT', 'LPT', 'LOR', 'ERD']
Testing Environment	la01, la02, la03, la04, la05	Random instances	Flexible process with random instances
Testing Random Seed	Seed 1 to 100	Seed 1 to 100	Seed 1 to 100

The performance of the proposed DDQN framework is evaluated across three experimental cases (**1A**, **1B**, and **1C**), with results summarized in **Figure 4.8a–c**.

In **Case 1A** (**Figure 4.8a**), the DDQN method achieves the lowest average tardiness of 1136.42, significantly outperforming all traditional heuristics. It also attains the highest win rate at 56%, as shown in the pie chart, indicating that it outperforms other methods in more than half of the test instances. Traditional rules such as FIFO and LIFO perform poorly

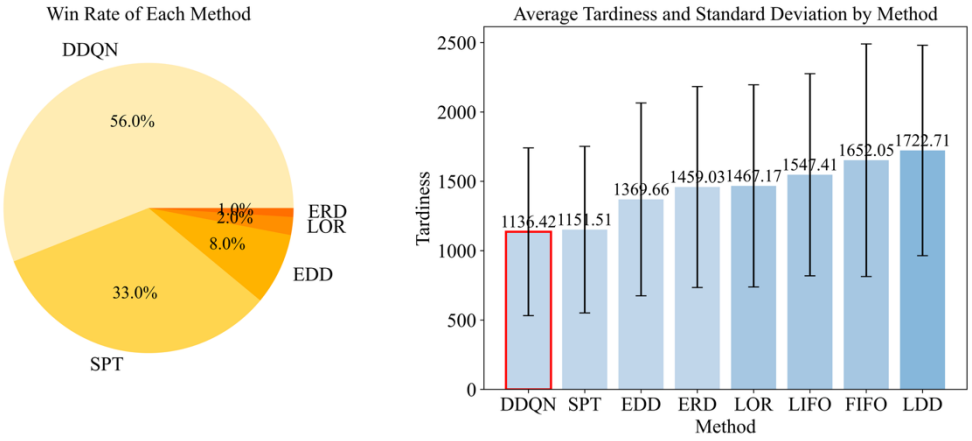
due to their lack of priority awareness, while even the best-performing heuristic, SPT, lags because it cannot adapt to dynamic job arrivals or varying operation sequences. The low standard deviation of DDQN's tardiness (≈108) reflects its consistent performance across diverse scenarios.

In **Case 1B** (**Figure 4.8b**), DDQN continues to lead with the lowest average tardiness (1178.19) and a win rate of 55%, demonstrating its robustness under routing flexibility where each job type follows a distinct full-path sequence. The improved performance of SPT compared to FIFO/LIFO/EDD is evident, but its static nature limits its ability to respond to real-time changes in machine availability and job release patterns.

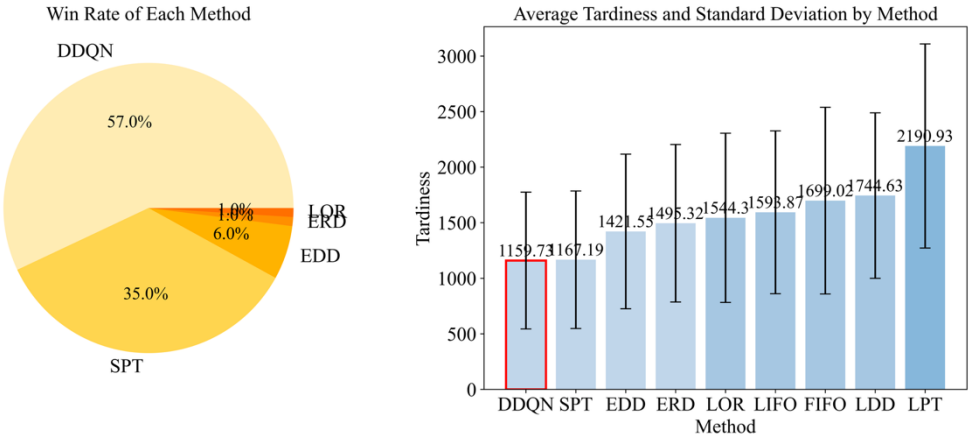
In **Case 1C** (**Figure 4.8c**), which introduces partial routings and allows jobs to skip work centers, DDQN maintains its superiority with the lowest average tardiness (1173.78) and a win rate of 58%—the highest among all methods. This further validates the framework's adaptability to complex, flexible manufacturing environments. While SPT remains competitive for short-duration tasks, it fails to dynamically adjust priorities based on evolving system states, resulting in higher variability and lower overall performance.

Notably, DDQN achieves its peak win rate precisely in this most complex scenario—a result that may initially appear counter-intuitive. However, the structural sparsity introduced by partial routings reduces contention at individual work centers, leading to more decoupled and locally observable decision problems. Combined with its richer state representation (which includes job-type distribution and waiting-time statistics in **Cases 1B** and **1C**), the DDQN agent can proactively anticipate workload patterns and allocate resources more effectively, even during episodes with no immediate job arrivals. In contrast, the denser, fully connected job flows in **Cases 1A** and **1B** create higher system-wide coupling, where static heuristics like SPT occasionally benefit from simplicity. Thus, rather than being hindered by complexity, the DDQN framework leverages the flexibility of **Case 1C** to achieve its most consistent advantage.

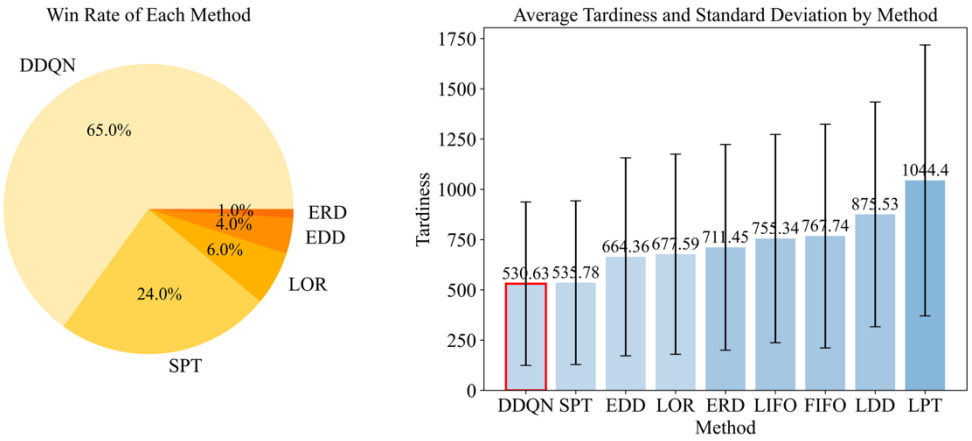
Across all cases, the DDQN agent consistently delivers superior scheduling decisions by learning from environmental feedback and adapting in real time. Its decentralized decision-making process enables scalability and reliability in large-scale, dynamic systems. The low standard deviations observed in all three figures confirm the stability and robustness of the approach, making it a promising solution for real-world dynamic flexible job shop scheduling problems.



(a)



(b)



(c)

Figure 4.8. Win rate and average tardiness (a) Case 1A, (b) Case 1B, (c) Case 1C.

4.2.3. SHAP Analysis of State Representation

Table 4.3. SHAP feature importance across work centers in **Case 1A**. Each cell reports the mean absolute SHAP value and its rank within that work center (value (rank)). Features are ordered by descending importance in Work Center 0. Higher values indicate greater influence on the agent's scheduling decisions.

Feature	WC0	WC1	WC2	WC3	WC4
Num_Machines_in_WC	2123.103 (1)	1772.334 (1)	2141.605 (1)	1546.981 (1)	2244.993 (1)
Num_Waiting_Jobs_in_WC	276.267 (2)	254.380 (2)	266.195 (3)	162.516 (4)	375.477 (4)
Min_Flow_Time	186.016 (3)	247.029 (3)	319.110 (2)	310.150 (2)	486.949 (2)
Mean_Flow_Time	182.199 (4)	137.623 (5)	148.784 (6)	143.201 (5)	237.909 (5)
Mean_Processing_Time_in_WC	166.267 (5)	139.634 (4)	194.504 (4)	199.474 (3)	380.296 (3)
Mean_Time_to_Due_Date	143.398 (6)	122.413 (6)	165.894 (5)	100.501 (6)	217.626 (6)
Max_Flow_Time	100.800 (7)	54.809 (10)	117.848 (7)	74.207 (8)	135.073 (8)
Std_Processing_Time_in_WC	84.266 (8)	59.583 (8)	60.903 (12)	57.890 (10)	115.392 (11)
Min_Time_to_Due_Date	71.508 (9)	46.088 (11)	46.058 (15)	47.264 (14)	57.207 (15)

Min_Processing_Time_in_WC	65.014 (10)	34.136 (15)	72.236 (9)	97.034 (7)	167.317 (7)
Std_Waiting_Time	55.881 (11)	75.155 (7)	49.453 (13)	54.822 (11)	127.691 (9)
Max_Processing_Time_in_WC	41.224 (12)	33.135 (16)	98.750 (8)	33.227 (17)	125.527 (10)
Max_Time_to_Due_Date	34.241 (13)	37.439 (13)	48.535 (14)	62.661 (9)	54.911 (16)
Mean_Remaining_Operations	34.033 (14)	19.673 (21)	17.654 (22)	9.815 (24)	54.540 (17)
Max_Remaining_Operations	32.229 (15)	7.718 (28)	14.909 (24)	12.477 (23)	13.490 (27)
Mean_Waiting_Time	31.908 (16)	31.382 (17)	67.685 (10)	15.011 (21)	29.023 (21)
Std_Remaining_Operations	31.174 (17)	24.821 (19)	36.500 (18)	44.923 (15)	53.897 (18)
Std_Flow_Time	30.409 (18)	45.966 (12)	61.826 (11)	48.987 (12)	70.579 (13)
Std_Remaining_Processing_Time	27.843 (19)	35.839 (14)	40.272 (17)	47.491 (13)	62.796 (14)
Std_Time_to_Due_Date	21.657 (20)	28.303 (18)	25.359 (19)	30.368 (19)	39.046 (20)
Min_Remaining_Operations	21.117 (21)	9.895 (26)	19.416 (21)	33.000 (18)	25.279 (22)
Max_Waiting_Time	19.541 (22)	55.647 (9)	45.679 (16)	42.453 (16)	79.863 (12)
Max_Remaining_Processing_Time	16.926 (23)	21.823 (20)	15.187 (23)	14.913 (22)	24.436 (23)
Mean_Remaining_Processing_Time	12.665 (24)	18.958 (22)	23.436 (20)	22.671 (20)	43.480 (19)
Min_Remaining_Processing_Time	9.662 (25)	10.450 (25)	9.304 (28)	8.166 (27)	17.480 (26)
Max_Earliest_Machine_Available_Time	4.157 (26)	12.769 (23)	11.015 (27)	8.661 (26)	20.783 (24)
Min_Earliest_Machine_Available_Time	3.939 (27)	9.479 (27)	11.655 (26)	7.326 (28)	9.728 (28)
Mean_Earliest_Machine_Available_Time	3.792 (28)	11.141 (24)	13.522 (25)	9.454 (25)	18.292 (25)
Std_Earliest_Machine_Available_Time	3.646 (29)	1.909 (29)	0.962 (29)	3.578 (29)	1.778 (29)
Min_Waiting_Time	0.000 (30)	0.000 (30)	0.000 (30)	0.000 (30)	0.000 (30)

Num_Job_Types_in_System	0.000 (30)	0.000 (30)	0.000 (30)	0.000 (30)	0.000 (30)
Num_Work_Centers	0.000 (30)	0.000 (30)	0.000 (30)	0.000 (30)	0.000 (30)

Table 4.3 presents the SHAP feature importance for each work center (WC0–WC4) in **Case 1A**, with features ordered by their influence on WC0's policy. The analysis reveals a consistent and pronounced pattern across all work centers: "*Num_Machines_in_WC*" is unequivocally the most important feature, ranking first in every case with SHAP values far exceeding those of other inputs. This is followed by "*Num_Waiting_Jobs_in_WC*," which ranks second in WC0 and WC1, but third or fourth in WC2–WC4. Together, these two local indicators form the core of the agent's decision logic, though machine count consistently holds greater weight.

This hierarchy offers an important insight into the learned scheduling behavior. In real-world job shops, the number of machines defines the fundamental processing architecture of a work center, whether it is a single-machine bottleneck or a multi-machine parallel station. This structural property directly shapes which dispatching rules are effective. For instance, the shortest processing time (SPT) may be optimal for a single machine, while load balancing rules might be preferred in multi-machine settings. The agent's strong sensitivity to machine count suggests it has internalized this operational principle. Queue length ("*Num_Waiting_Jobs_in_WC*") remains relevant as a signal of immediate congestion, but its secondary role implies that capacity structure is a more decisive factor than transient workload levels.

Another notable observation is the relatively high importance of flow-time-related features. "*Min_Flow_Time*" ranks second in three out of five work centers (WC2, WC3, WC4), and "*Mean_Flow_Time*" also appears in the top six universally. This indicates that the agent pays close attention to how long jobs have already spent in the system, likely to prevent excessive delays and maintain throughput. This strategy indirectly supports the primary objective of minimizing tardiness.

In contrast, due-date-related features such as "*Mean_Time_to_Due_Date*" and "*Min_Time_to_Due_Date*" rank only moderately (typically 6th to 15th). This may seem counterintuitive given that total weighted tardiness is the optimization target. However, it reflects a sophisticated trade-off learned through reinforcement learning: in a dynamic and stochastic environment, aggressively prioritizing imminent due dates can lead to unstable queue dynamics and poor overall performance. Instead, the agent appears to prioritize system fluidity by managing capacity utilization and preventing congestion, which ultimately leads to better global due-date adherence. This challenges the dominance of classical due-date-driven heuristics like EDD and demonstrates the adaptability of RL policies.

Finally, system-level attributes such as "*Num_Work_Centers*" and "*Num_Job_Types_in_System*" show zero SHAP values and rank last across all agents. This is not a limitation but a strength of the decentralized design. It confirms that effective scheduling decisions can be made using only local, readily observable information. The agent does not require knowledge of the global factory state, which enhances robustness, reduces communication overhead, and improves scalability. These are key advantages for real-world deployment, where full observability is often impractical.

Table 4.4. SHAP feature importance across work centers in **Case 1B**. Each cell reports the mean absolute SHAP value and its rank within that work center (value (rank)). Features are ordered by descending importance in Work Center 0. Higher values indicate greater influence on the agent's scheduling decisions.

Feature	**WC0**	**WC1**	**WC2**	**WC3**	**WC4**
Num_Machines_in_WC	1944.762 (1)	1838.226 (1)	1927.075 (1)	2404.701 (1)	1756.968 (1)
Min_Time_Since_Release	363.492 (2)	258.142 (2)	195.948 (4)	291.903 (4)	144.587 (4)
Mean_Processing_Time_in_WC	217.786 (3)	222.977 (3)	283.779 (2)	367.691 (3)	232.264 (3)
Num_Waiting_Jobs_in_WC	162.553 (4)	194.165 (4)	246.141 (3)	397.722 (2)	234.057 (2)

Mean_Time_Since_Release	144.554 (5)	122.041 (5)	176.530 (5)	241.662 (5)	118.098 (6)
Mean_Time_to_Due_Date	114.026 (6)	106.642 (6)	127.882 (7)	146.103 (7)	49.911 (13)
Std_Waiting_Time	113.484 (7)	57.580 (10)	80.140 (9)	137.291 (8)	63.477 (9)
Max_Time_Since_Release	108.935 (8)	78.521 (9)	131.528 (6)	83.228 (11)	135.687 (5)
Min_Processing_Time_in_WC	98.410 (9)	47.108 (13)	65.289 (11)	57.211 (15)	65.300 (8)
Mean_Waiting_Time	89.501 (10)	41.068 (15)	47.317 (14)	100.589 (10)	58.275 (10)
Num_Job_Types_in_System	81.449 (11)	34.830 (18)	56.063 (12)	62.500 (12)	49.402 (14)
Max_Processing_Time_in_WC	66.154 (12)	56.688 (11)	71.217 (10)	41.605 (16)	57.056 (11)
Std_Time_Since_Release	60.256 (13)	30.930 (19)	48.829 (13)	57.999 (14)	56.339 (12)
Std_Processing_Time_in_WC	59.996 (14)	95.292 (7)	103.669 (8)	152.143 (6)	76.141 (7)
Max_Waiting_Time	38.741 (15)	15.624 (22)	35.768 (16)	27.290 (20)	21.921 (18)
Std_Time_to_Due_Date	37.016 (16)	21.861 (20)	30.947 (19)	27.722 (19)	25.588 (17)
Std_Remaining_Processing_Time	29.187 (17)	38.257 (16)	35.600 (17)	23.104 (21)	31.777 (16)
Mean_Remaining_Operations	25.463 (18)	15.476 (23)	11.835 (28)	29.339 (18)	19.203 (21)
Min_Time_to_Due_Date	20.704 (19)	56.129 (12)	26.345 (20)	103.040 (9)	17.297 (22)
Std_Remaining_Operations	17.424 (20)	37.586 (17)	25.217 (21)	31.060 (17)	13.680 (23)
Max_Remaining_Operations	17.331 (21)	19.843 (21)	32.496 (18)	10.325 (27)	13.033 (24)
Max_Time_to_Due_Date	16.141 (22)	79.801 (8)	37.092 (15)	58.075 (13)	21.457 (19)
Min_Remaining_Processing_Time	15.423 (23)	6.596 (27)	19.691 (23)	18.567 (23)	10.857 (25)
Max_Remaining_Processing_Time	12.131 (24)	7.092 (25)	22.720 (22)	10.836 (25)	32.067 (15)
Mean_Remaining_Processing_Time	11.838 (25)	45.012 (14)	18.300 (24)	22.354 (22)	21.420 (20)

Min_Remaining_Operations	9.254 (26)	11.384 (24)	12.504 (27)	17.282 (24)	10.408 (26)
Max_Earliest_Machine_Available_Time	9.209 (27)	6.705 (26)	10.777 (29)	7.246 (29)	6.452 (28)
Mean_Earliest_Machine_Available_Time	7.494 (28)	6.104 (28)	14.047 (25)	10.600 (26)	5.579 (29)
Min_Earliest_Machine_Available_Time	5.514 (29)	5.836 (29)	13.839 (26)	10.275 (28)	6.467 (27)
Std_Earliest_Machine_Available_Time	0.625 (30)	1.018 (30)	1.703 (30)	1.569 (30)	1.518 (30)
Min_Waiting_Time	0.000 (31)	0.000 (31)	0.000 (31)	0.000 (31)	0.000 (31)
Num_Work_Centers	0.000 (31)	0.000 (31)	0.000 (31)	0.000 (31)	0.000 (31)

Table 4.4 presents the SHAP feature importance analysis for each work center in **Case 1B**, quantifying the average impact of each input feature on the agent's decision-making. The results reveal a striking and consistent pattern: "*Num_Machines_in_WC*" is the most important feature across all five work centers, consistently holding rank 1 with substantially higher SHAP values than other inputs. This indicates that the agent's dispatching rule selection is primarily sensitive to the local processing capacity of the work center.

In contrast, "*Num_Waiting_Jobs_in_WC*" ranks second only in Work Centers 3 and 4, and third or fourth in Work Centers 0–2. While still among the more relevant features, its influence is clearly secondary to machine count in most cases. Other notable secondary factors include "*Mean_Processing_Time_in_WC*" (ranked 2nd or 3rd in four out of five centers) and "*Min_Time_Since_Release*" (ranked 2nd in WC0 and WC1), suggesting that job characteristics (average processing duration and recency of arrival) also inform the agent's decisions.

This emphasis on machine count over queue length offers a nuanced insight into the learned policy. In real-world scheduling, the number of machines fundamentally determines whether a work center can process jobs in parallel. An agent facing a single-machine bottleneck may prioritize rules that minimize flow time (e.g., SPT), whereas one with multiple machines

might adopt different strategies to balance utilization or avoid idleness. The data suggest that capacity structure outweighs instantaneous congestion level as a decision cue in this environment.

Furthermore, system-level features such as "*Num_Work_Centers*" and "*Num_Job_Types_in_System*" exhibit zero SHAP values, confirming that the agent operates purely on local information, which is a hallmark of the decentralized design. The negligible role of the global state implies that effective scheduling can be achieved without centralized coordination, enhancing scalability and robustness.

Notably, the relatively modest importance of queue length ("*Num_Waiting_Jobs_in_WC*") may seem counterintuitive at first glance, as traditional heuristics often treat it as a primary signal. However, this result highlights how reinforcement learning can uncover context-dependent priorities: when machine capacity and job processing characteristics are known, they may provide more predictive power for minimizing tardiness than raw queue size alone.

Table 4.5. SHAP feature importance across work centers in **Case 1C**. Each cell reports the mean absolute SHAP value and its rank within that work center (value (rank)). Features are ordered by descending importance in Work Center 0. Higher values indicate greater influence on the agent's scheduling decisions.

Feature	**WC0**	**WC1**	**WC2**	**WC3**	**WC4**
Num_Machines_in_WC	1565.879 (1)	1406.514 (1)	1733.186 (1)	1407.124 (1)	1340.597 (1)
Min_Flow_Time	152.199 (2)	171.447 (2)	169.305 (2)	167.985 (2)	164.070 (2)
Mean_Processing_Time_in_WC	137.130 (3)	131.637 (3)	158.721 (3)	62.760 (10)	140.262 (3)
Num_Waiting_Jobs_in_WC	103.616 (4)	126.555 (4)	142.286 (4)	125.483 (3)	137.690 (4)
Max_Processing_Time_in_WC	93.202 (5)	73.314 (6)	136.725 (6)	120.789 (4)	72.879 (5)
Max_Flow_Time	89.974 (6)	61.524 (9)	139.349 (5)	92.750 (6)	63.694 (6)

Potential_Job_Type_Number_in_WC	88.535 (7)	55.172 (10)	58.995 (11)	48.540 (12)	20.538 (17)
Num_Job_Types_in_System	68.516 (8)	61.601 (8)	54.496 (12)	48.669 (11)	50.299 (9)
Mean_Flow_Time	67.030 (9)	68.924 (7)	98.043 (7)	64.385 (9)	54.196 (7)
Max_Waiting_Time	64.906 (10)	38.474 (11)	73.263 (10)	96.412 (5)	22.440 (15)
Mean_Time_to_Due_Date	64.199 (11)	79.722 (5)	86.006 (8)	68.628 (7)	53.276 (8)
Min_Processing_Time_in_WC	53.336 (12)	37.766 (12)	84.159 (9)	41.170 (13)	47.499 (10)
Std_Waiting_Time	49.271 (13)	32.001 (13)	33.615 (13)	65.584 (8)	18.660 (20)
Mean_Remaining_Operations	38.773 (14)	13.077 (25)	20.476 (21)	9.970 (33)	8.610 (31)
Std_Processing_Time_in_WC	36.589 (15)	31.430 (14)	30.873 (15)	27.952 (16)	43.370 (11)
Std_Remaining_Processing_Time	26.862 (16)	11.055 (27)	32.585 (14)	13.593 (27)	22.178 (16)
Std_Flow_Time	24.628 (17)	18.381 (21)	24.985 (19)	21.874 (21)	18.952 (19)
Min_Remaining_Operations	23.012 (18)	15.563 (22)	25.833 (18)	7.275 (34)	8.920 (29)
Min_Time_to_Due_Date	22.595 (19)	22.922 (19)	27.236 (16)	11.036 (32)	14.848 (21)
Min_Remaining_Processing_Time	22.102 (20)	7.743 (33)	8.241 (34)	32.566 (15)	12.522 (23)
Mean_Remaining_Processing_Time	17.912 (21)	27.636 (15)	10.324 (32)	25.565 (17)	8.629 (30)
Max_Remaining_Processing_Time	17.276 (22)	10.429 (28)	20.108 (22)	14.378 (26)	5.991 (34)
Std_Remaining_Operations	16.765 (23)	23.073 (18)	19.470 (24)	20.025 (22)	28.799 (13)
Max_Time_to_Due_Date	14.918 (24)	13.728 (23)	14.594 (26)	22.625 (20)	10.611 (25)
Mean_Waiting_Time	13.995 (25)	25.203 (16)	10.872 (30)	18.457 (24)	24.763 (14)
Std_Time_to_Due_Date	13.123 (26)	13.622 (24)	10.300 (33)	12.132 (28)	7.016 (33)
Min_Operations	12.877 (27)	7.347 (34)	11.162 (29)	23.837 (18)	20.147 (18)

Max_Earliest_Machine_Available_Time	12.797 (28)	10.185 (29)	12.349 (27)	11.325 (31)	11.857 (24)
Min_Earliest_Machine_Available_Time	12.783 (29)	7.864 (32)	11.578 (28)	11.521 (30)	9.071 (28)
Mean_Earliest_Machine_Available_Time	12.191 (30)	8.595 (31)	10.480 (31)	12.052 (29)	9.942 (26)
Max_Remaining_Operations	12.091 (31)	9.773 (30)	25.837 (17)	23.125 (19)	8.103 (32)
Mean_Operations	10.834 (32)	11.135 (26)	16.936 (25)	19.481 (23)	9.703 (27)
Max_Operations	9.187 (33)	18.856 (20)	21.028 (20)	36.821 (14)	12.654 (22)
Std_Operations	8.094 (34)	23.873 (17)	19.966 (23)	16.831 (25)	29.814 (12)
Std_Earliest_Machine_Available_Time	2.644 (35)	3.733 (35)	6.169 (35)	3.392 (35)	2.198 (35)
Min_Waiting_Time	0.000 (36)	0.000 (36)	0.000 (36)	0.000 (36)	0.000 (36)
Num_Work_Centers	0.000 (36)	0.000 (36)	0.000 (36)	0.000 (36)	0.000 (36)

Table 4.5 details the SHAP analysis findings for each work center in Case 1C. A clear and consistent pattern emerges across all five work centers: "*Num_Machines_in_WC*" is the single most important feature, holding the top rank (1) in every case. This is followed by flow-time-related metrics, with "*Min_Flow_Time*" consistently ranking second. In contrast, "*Num_Waiting_Jobs_in_WC*" ranks only third or fourth, indicating it is influential but not the primary driver of decisions.

This result offers a meaningful insight into the agent's learned policy. The paramount importance of machine count suggests that the agent's rule selection is most sensitive to the local processing capacity of the work center. In real-world terms, knowing whether a work center has one machine or five fundamentally changes the scheduling dynamics—more machines imply greater parallelism and potentially lower congestion pressure, which may favor different dispatching rules (e.g., less urgency for SPT). The high importance of "*Min_Flow_Time*" (i.e., the shortest time a job has spent in the system) further indicates that

the agent prioritizes jobs that are either newly arrived or have progressed quickly, possibly to maintain smooth flow and avoid bottlenecks.

Notably, system-level features such as "*Num_Work_Centers*" and "*Num_Job_Types_in_System*" show zero SHAP values and rank last, confirming that the agent relies exclusively on local state information. This is a finding consistent with the decentralized design. Perhaps more surprisingly, "*Num_Waiting_Jobs_in_WC*," while relevant, is not the dominant signal one might intuitively expect in a queueing context. This suggests that in this specific environment (with its due date distributions, arrival patterns, and objective of minimizing tardiness), resource capacity (machines) and job progress (flow time) provide more decisive cues than raw queue length alone.

In summary, the SHAP analysis reveals that the agent's decisions in **Case 1C** are primarily governed by two local factors: available machine capacity and the minimum flow time of waiting jobs. The relatively lower weight on queue length challenges a common heuristic assumption and highlights how reinforcement learning can uncover non-intuitive, yet effective decision criteria grounded in the specific dynamics of the simulated environment.

4.3. Implications of the Study

The study has important implications for both theoretical research and practical applications in the field of dynamic scheduling and smart manufacturing systems:

- *Improved Real-Time Decision-Making:* The study proved that real-time scheduling decisions in highly dynamic environments can be achieved through a decentralized event-driven framework based on reinforcement learning, namely DDQN. The framework's capacity for adapting to unpredictable job arrivals and machine availability in response to continuous disruptions would lead to scalable solutions for the contemporary manufacturing system.

- *Enhanced Operational Performance:* Compared to conventional dispatching rules like FIFO and SPT, the proposed method reduces work tardiness and improves machine usage. This demonstrates how reinforcement learning can boost resource usage, decrease delays, and increase production flow in intricate real-world industrial applications.
- *Scalability for Industry 4.0 Applications:* The decentralized framework allows individual work centers to make independent decisions based on local information, reducing computational bottlenecks associated with centralized systems. This makes the method highly scalable for large-scale manufacturing systems, aligning with Industry 4.0's goals of smart, distributed, and automated production systems.
- *Interpretability and Transparency in AI Systems:* Through SHAP analysis, the study provides the most insightful information on what state features influence the DDQN agents' decisions. This will improve the transparency of reinforcement learning models, gaining trust in AI-driven systems and enabling practitioners to better understand and fine-tune decision-making processes.

The work provides practical, scalable, and interpretable methods that augment manufacturing systems while advancing the theoretical knowledge of reinforcement learning in dynamic scheduling.

4.4. Conclusions

This chapter investigates the problem of DFJSP by developing a decentralized, event-driven framework using DDQN reinforcement learning. The proposed method demonstrated superior performance in real-time job sequencing and machine allocation optimization. Specifically:

- The DDQN-based framework effectively minimized mean job tardiness, outperforming conventional dispatching rules such as FIFO, SPT, and EDD.

- Numerical experiments showed that the DDQN method achieved the highest win rate and lowest average tardiness across all three cases:
  - **Case 1A** (Benchmark Dataset): Highest win rate (56%) and lowest average tardiness (1136.42), outperforming SPT (1151.5), FIFO (1652.1), and EDD (1369.7).
  - **Case 1B** (Extended Random Dataset): Maintained the highest win rate and lowest average tardiness (1159.7), surpassing LPT (2190.9) and other conventional methods.
  - **Case 1C** (Flexible Process Type): Achieved the lowest average tardiness (530.6), significantly outperforming LPT (1044.4) and demonstrating adaptability in flexible process environments.
- SHAP analysis revealed that the most influential features for scheduling decisions were the "Number of Machines in the Work Center" and the "Number of Waiting Jobs in the Work Center," emphasizing the importance of local workload and resource availability.
- The decentralized framework enhances scalability and responsiveness, making it suitable for large-scale manufacturing systems aligned with Industry 4.0 requirements.
- The event-driven architecture reduces computational overhead by triggering scheduling decisions only when dynamic events occur, such as job arrivals or machine availability.

The study bridges the gap between theoretical RL and practical manufacturing applications. It shows that AI-driven methods can enhance efficiency and adaptability in DFJSP.

However, the proposed DDQN-based framework has notable limitations. Most significantly, its action space is discrete, restricted to selecting among a fixed set of eight heuristic rules (e.g., SPT, FIFO, LPT). While this design simplifies learning and ensures interpretability, it inherently limits the scheduler's flexibility: the agent cannot generate novel or interpolated dispatching behaviors tailored to unique system states. This becomes particularly constraining under severe disruptions, such as sudden machine breakdowns, urgent high-

priority job insertions, or rapid shifts in workload distribution. In such cases, fine-grained and continuous prioritization may be necessary to achieve optimal recovery.

Furthermore, rule selection is inherently reactive; even with real-time state observation, the agent can only choose from pre-defined strategies rather than dynamically constructing a response. This discrete nature, while effective for the tested scenarios, highlights a fundamental trade-off between simplicity and adaptability in dynamic scheduling.

These limitations motivate the extension presented in **Chapter 5**, where a feature-weighted continuous scoring mechanism based on Proximal Policy Optimization (PPO) is introduced to overcome the rigidity of discrete rule selection and enable more nuanced, disruption-resilient decision-making.

Chapter 5. Problem 2: Scheduling under Machine Reliability Constraints with RL

5.1. Introduction

In **Problem 1**, a decentralized, event-driven scheduling framework using the DDQN was implemented to address the DFJSP. This approach demonstrated substantial improvements over traditional scheduling heuristics, specifically in minimizing mean tardiness and enhancing real-time adaptability to dynamic conditions, such as random job arrivals and variations in machine availability. Despite these achievements, several limitations emerged from this rule-selection-based approach, particularly the discrete nature of its actions. This discreteness restricts flexibility and responsiveness, especially under conditions of significant disruptions, such as sudden machine breakdowns or abrupt changes in production priorities.

To overcome these limitations, this chapter introduces a feature-weighted continuous scoring mechanism based on the PPO algorithm. This advancement transitions the scheduling strategy from discrete rule selection to a continuous prioritization scheme, providing greater adaptability and nuanced responsiveness to real-time manufacturing disruptions.

The main contributions in this chapter are:

- Continuous Prioritization Transition: Replacing the discrete decision-making process with a continuous, feature-weighted prioritization system, allowing for more flexible and finely tuned scheduling responses.
- Stable Learning through PPO Integration: Employing PPO to manage continuous action spaces in highly dynamic and uncertain manufacturing environments.
- Incorporation of Machine Breakdown Dynamics: Explicitly integrating machine

breakdown events into both the decision-making process and the state representation, enhancing the system's robustness and its ability to swiftly adapt schedules in response to real-time disruptions.

Through these enhancements, the proposed scheduling design in **Problem 2** aims to improve scheduling performance, achieving greater flexibility, robustness, and scalability, thereby addressing the demands of modern smart manufacturing systems under the Industry 4.0/5.0 paradigm.

The remainder of this chapter is organized as follows. **Section 5.2** presents **Problem 2**, which extends the baseline scheduling model with stochastic machine breakdowns and repairs, thereby introducing event-driven disruptions that require real-time rescheduling and robust decision-making under uncertainty. **Section 5.3** describes the enhanced reinforcement learning approach, PPO-WF, which uses a continuous feature-weighted prioritization strategy to improve scheduling flexibility and robustness. **Section 5.4** provides an overview of the experimental settings, including the breakdown model and simulation configuration. **Section 5.5** presents the comparative performance evaluation across multiple methods. **Section 5.6** discusses the SHAP-based feature attribution analysis of the PPO-WF agent in the context of stochastic machine failures and repairs, revealing how the policy leverages local resource capacity, machine health indicators, and event-driven signals to make robust dispatching decisions under uncertainty. **Section 5.7** outlines the broader implications of the study's findings for real-world manufacturing systems. Finally, **Section 5.8** summarizes the key conclusions and prepares the foundation for the multi-objective extension explored in **Case 3**. To facilitate reproducibility, the complete source code and environment setup scripts have been uploaded to GitHub: https://github.com/daylunch/RL_DFJSP/tree/UVFA.

5.2. Problem Overview

Problem 2, as described in **Chapter 3**, extends the baseline dynamic flexible job shop

scheduling model by explicitly incorporating stochastic machine breakdowns and repairs into the system dynamics, thereby changing both the event structure and the feasibility constraints of the scheduling problem. While the original formulation accounts for stochastic job arrivals and decentralized decision-making across multi-machine work centers, this case adds complexity by modeling machine reliability through stochastic failures and repairs. Each machine can randomly transition between operational, broken, and repairing states, with failure and repair times following exponential distributions. Jobs in progress on a failed machine are aborted and rescheduled on other qualified machines, requiring full reprocessing of the affected operation. These breakdowns interrupt ongoing operations and directly impact the scheduling process, requiring the system to adapt dynamically in real time.

To handle this increased complexity, the event-driven scheduling architecture is expanded with two additional event types: machine breakdown and machine repair. These join the existing triggers of job arrival and operation completion, enabling the decentralized reinforcement learning agents to respond immediately to reliability-related disruptions. As a result, more realistic and challenging test instances are provided in **Problem 2** for evaluating the adaptability and robustness of the scheduling framework under uncertain manufacturing conditions.

5.3. Methodology

5.3.1. Distributed Event-Driven Framework

The methodology adopted in **Problem 2** extends the decentralized scheduling architecture initially established in **Problem 1** by integrating additional real-time event triggers and employing a continuous prioritization mechanism facilitated by PPO. This approach aims to improve adaptability and robustness under dynamically changing manufacturing conditions.

Overview of Decentralized Architecture

The decentralized architecture is structured around multiple autonomous work centers, each equipped with a dedicated PPO-based reinforcement learning agent. These agents independently process local scheduling decisions based on real-time data streams relevant to their operational contexts. This decentralized strategy supports scalability and flexibility, avoiding bottlenecks common in centralized systems and providing rapid local responsiveness to dynamic events.

Each work center functions autonomously, utilizing parallel processing capabilities to manage local job queues and machine resources effectively. Real-time communication protocols enable agents to remain synchronized and informed of critical system-wide conditions, thereby supporting cohesive decision-making across the production environment.

Event Triggers Considered

To accurately reflect dynamic operational realities, the decentralized framework recognizes and responds to the following event triggers:

- Job Arrivals: The scheduling decisions are initiated in response to new job arrivals, aiming for rapid assignment and reducing initial waiting times, thus improving resource utilization and minimizing delays.
- Operation Completions: Completion of operations signals machine availability, prompting agents to reallocate resources swiftly and maintain efficient production flows.
- Machine Breakdowns (New): Incorporating machine breakdown events allows the agents to immediately reassess scheduling priorities, thereby reducing disruption through dynamic reallocation of tasks to alternate resources or adjustment of production sequences.
- Machine Repairs (New): Upon repair completion, agents immediately reincorporate the

restored machine into the operational workflow, updating task assignments to maintain efficient resource utilization.

By explicitly considering these event triggers, the decentralized event-driven framework can effectively respond to real-time changes and disturbances, thereby enhancing system agility, resilience, and performance to meet modern manufacturing demands.

5.3.2. Feature-Weighted Prioritization Strategy

Traditional rule selection approaches (**Problem 1**) discretize the decision-making process into a limited set of predefined dispatching rules, such as FIFO, LIFO, EDD, or SPT, in dynamic flexible job shop scheduling. While these methods are computationally efficient and provide straightforward heuristics, they inherently lack the flexibility and granularity required to capture subtle variations in real-time manufacturing conditions. Specifically, such rule-based systems struggle to model nuanced combinations of scheduling priorities, limiting the system's capability to respond dynamically to unpredictable events like machine failures, random job arrivals, or urgent priority changes.

To overcome these limitations, this chapter proposes a feature-weighted scheduling method based on PPO. Feature-weighted approaches allow continuous adjustments of scheduling priorities based on real-time, localized state information. This enables finer control, greater interpretability, and enhanced adaptability, which are essential for handling complex scenarios frequently encountered in Industry 4.0 manufacturing environments, such as those featuring random job arrivals, dynamic routing flexibility, and unexpected machine disruptions.

The main concept of the proposed methodology is reordering waiting jobs by leveraging a learnable feature-weighting mechanism driven by PPO agents. The PPO agent outputs a continuous weight vector $\boldsymbol{w}$, each dimension of which corresponds to a specific scheduling

feature $f_a(i)$, such as processing time, due date proximity, machine availability, or remaining job operations.

Formally, each candidate's waiting job J_i is assigned a priority score $F(i)$ calculated as follows:

$$F(i) = \sum_a w_a \cdot f_a(i), \tag{5.1}$$

This priority score integrates multiple potentially conflicting attributes into a unified, interpretable metric used to reorder the waiting jobs dynamically. Jobs are sequenced according to their scores, and the highest-ranked jobs are subsequently assigned to idle machines, thereby improving overall operational performance adaptively and in real time.

The feature-weighted PPO-based scheduling mechanism significantly advances the state-of-the-art in dynamic scheduling through multiple key advantages:

- Expressive Power of Continuous Weights: Unlike traditional discrete dispatching rules, continuous weight vectors afford a richer representational capacity, enabling the precise tuning of job priority based on subtle differences in system conditions. This flexibility allows the system to accommodate unique operational scenarios without relying on predefined heuristics.
- Smooth Learning Landscape for PPO: The continuous nature of weight adjustments creates a smooth optimization landscape, which supports more stable and sample-efficient learning with PPO. This stability is crucial in ensuring convergence to optimal or near-optimal scheduling policies in highly dynamic environments.
- Selective Feature Emphasis through Positive and Negative Weights: By allowing both positive and negative values, the PPO agents can dynamically penalize, or reward specific job features according to real-time system states and operational objectives. For instance, negative weights can reduce the priority of jobs requiring extensive setup or

those with distant due dates, while positive weights can prioritize jobs approaching their deadlines or critical production bottlenecks.

- Enhanced Interpretability: Continuous, learnable weights directly represent the influence of each scheduling feature. This transparency facilitates easier refinement and justification of decisions within complex operational environments.

Overall, the proposed feature-weighted PPO-based scheduling framework provides a scalable, robust, and highly interpretable approach suitable for advanced smart manufacturing systems, significantly outperforming traditional rule-based methods by dynamically adapting to complex, real-time scheduling scenarios.

5.3.3. The Reinforcement Learning Approach

In dynamic scheduling environments, where decision-making involves high-dimensional state representations and continuous adaptation to real-time events, reinforcement learning offers a promising alternative to traditional rule-based heuristics. However, achieving stable and efficient policy learning remains a challenge, particularly when system dynamics are non-stationary or when reward signals are sparse or delayed. To address these issues, this study adopts the PPO algorithm.

PPO is a policy-gradient-based reinforcement learning algorithm designed to achieve stable, sample-efficient training. As an actor-critic method, PPO employs two neural networks: the actor, which defines the policy to select actions given states, and the critic, which estimates the expected cumulative reward from a given state. PPO simplifies earlier trust-region-based methods, such as Trust Region Policy Optimization (TRPO), by using a clipped surrogate objective function to limit policy updates within a controlled "trust region," thus avoiding drastic policy shifts and promoting robust learning [145].

In this study, the PPO algorithm is adopted to solve continuous action space problems,

offering a balance between performance and training stability, both critical in dynamic and high-dimensional environments such as real-time scheduling systems. The policy network outputs a multivariate Gaussian distribution characterized by a mean vector $\mu_\theta(\boldsymbol{s_t})$ and a standard deviation vector $\sigma_\theta(\boldsymbol{s_t})$, where $\boldsymbol{s_t}$ denotes the current state. The continuous action a_t is then sampled from the distribution $N(\mu_\theta(\boldsymbol{s_t}), {\sigma_\theta}^2(\boldsymbol{s_t}))$, after which it is transformed through a hyperbolic tangent function to ensure boundedness, yielding $\tilde{a}_t = \tanh(a_t)$. This transformation is particularly important in real-world applications where the control variables, such as priority weights or machine assignments, must lie within fixed operational limits.

To optimize the policy, PPO utilizes Generalized Advantage Estimation (GAE), which offers a principled method for computing a smoothed estimate of the advantage function by reducing the variance of policy gradient updates without significantly increasing bias. GAE defines the advantage A_t at time step t as an exponentially weighted sum of temporal difference (TD) residuals [146]:

$$A_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \tag{5.2}$$

where:

$$\delta_t = r_t + \gamma V(\boldsymbol{s_{t+1}}) - V(\boldsymbol{s_t}), \tag{5.3}$$

here, γ is the discount factor that captures the importance of future rewards, and λ is a smoothing parameter controlling the trade-off between bias and variance. The policy is updated via a clipped surrogate objective function, which introduces a constraint on the change of the policy between successive updates. The objective is given by:

$$L^{\mathrm{CLIP}}(\theta) = E_t[\min(r_t(\theta)A_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)], \tag{5.4}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|\boldsymbol{s_t})}{\pi_{\theta_{old}}(a_t|\boldsymbol{s_t})}$ represents the ratio between the new and old policy probabilities. The clipping mechanism, controlled by the hyperparameter ϵ, prevents excessively large

policy updates that could degrade performance, thereby maintaining the policy within a "trust region" of the previous iteration. This conservative update strategy enables PPO to strike a favorable balance between fast convergence and robustness, making it highly applicable to complex, real-time optimization tasks. In the context of dynamic scheduling, where system states evolve unpredictably, and decisions must be made frequently and efficiently, PPO's stability and generalization capability offer significant advantages over more volatile policy gradient methods.

5.3.4. State Representation and Action Space

State Components

The state representation in **Problem 2** extends the initial setup from **Problem 1**, incorporating new metrics explicitly designed to handle the added complexity of machine breakdowns. The enhanced state components are shown in **Figure 5.1**. Specifically, a new category of machine reliability metrics has been added, including the number of broken machines, time since last repair, cumulative working time, and repair durations. These features provide the agent with critical information to assess machine availability and degradation over time. Furthermore, event indicators are introduced to capture discrete dynamic events such as job arrivals, machine failures, and repairs, enabling the agent to respond contextually to real-time system changes. These additions enhance the agent's situational awareness and decision-making granularity, allowing for more adaptive and resilient scheduling behavior under the uncertainty typical of modern smart manufacturing environments.

Definition of Action Space

As discussed in **Section 5.3.1**, features $f_a(i)$, together with a set of learnable weights $\boldsymbol{w}$, are employed to prioritize waiting jobs. The learnable weights $\boldsymbol{w}$ constitute the continuous

action space learned by the PPO agent and are constrained to the range $[-1,1]$ via a tanh activation function in the policy network's output layer.

Features for Prioritization

In contrast to the state $\boldsymbol{s_t}$, which characterizes the global environment at a decision point, features $f_a(i)$ are specifically designed to capture the distinguishing attributes of each waiting job. This distinction enables the model to effectively differentiate among waiting jobs and determine their execution order based on learned priorities.

State Representation (53)

- **Work Center Metrics (3):**
  Number of machines in work center; Number of job types in system; Number of waiting jobs in work center.
- **Job Queue Metrics (29):**
  Time to due date (mean, min, max, std); Waiting time (mean, min, max, std); Time since release (mean, min, max, std); Remaining operations (mean, min, max, std); Processing time (mean, min, max, std); Remaining total processing time (mean, min, max, std); Number of operations (mean, min, max, std); Potential job type number in work center.
- **Machine Reliability Metrics (13):**
  Number of broken machines; Time since last repair (mean, min, max, std); Accumulative working time (mean, min, max, std); Time been repaired (mean, min, max, std).
- **Event Indicators (5):**
  Binary indicators for: job arrival, job released to next work center, machine breakdown, machine repaired, operation completion.
- **Global Environment Indicators (3):**
  Jobs in other work centers; Available machines in other work centers; Machine breakdown counts.

Figure 5.2. The state representative of PPO-WF.

The candidate job features $f_a(i)$ include:

- **Time to Due Date:** $d_i - t$, where d_i is the due date of the job J_i and t is the current

time. This distinguishes the urgency of different jobs and helps prioritize jobs nearing their deadline.

- **Waiting time:** $t - r_{i,j}$, where r_i is the release time of the job J_i in this operation.
- **Remaining operation:** $o_i - j$, which represents the unfinished steps of job J_i. These features reflect the job complexity and remaining workload of the waiting jobs.
- **Processing time:** P_{ij}^{l}, the processing time of the operation O_{ij} in work center U_l. This feature reflects the immediate resource requirements.

Together, these features allow the model to distinguish between jobs based on temporal urgency, accumulated delay, structural complexity, and immediate processing demands, enabling dynamic prioritization aligned with scheduling objectives such as minimizing mean tardiness.

5.3.5. Reward

The primary objective of the DFJSP is to minimize job tardiness, defined as the time by which a job's completion exceeds its due date. By structuring the reward function specifically around tardiness, the PPO agent is guided towards scheduling decisions that systematically reduce job delays.

In PPO, the critic estimates the value function $V(\boldsymbol{s})$, representing the expected cumulative reward from a given state under the current policy, rather than explicitly evaluating individual actions with $Q(\boldsymbol{s}, a)$. At each decision point, the immediate reward r_t provides feedback directly related to the scheduling decision's effectiveness in reducing tardiness. The PPO agent utilises these rewards to update its policy through policy gradients, leveraging advantage estimation to guide policy improvement.

In the distributed decision-making process, using global tardiness directly as a reward is infeasible, as it provides no immediate or localized feedback on the impact of an individual

work center's dispatching action. To address this, we propose a novel local reward function that evaluates the change in scheduling quality within the work center before and after applying the agent's decision.

Specifically, the reward r is defined as the difference between the average modified slack of waiting jobs after and before the dispatching action:

$$r = \overline{\Delta d_{\text{after}}} - \overline{\Delta d_{\text{before}}}, \text{ where } \overline{\Delta d} = \frac{1}{|\mathcal{J}|}\sum_{i\in\mathcal{J}} \Delta d_i \,, \tag{5.5}$$

where,

$\mathcal{J}$: The set of jobs currently waiting in the work center's queue

Δd_i: Modified time to due date of job J_i

$\overline{\Delta d}$: Average modified time to due date of waiting jobs. The subscript "before" denotes the job sequence and expected completion times prior to the agent's action (i.e., under the existing queue order); The subscript "after" denotes the sequence and expected completion times after reordering the queue according to the agent's continuous scoring output (or selected dispatching logic).

$$\Delta d_i = \begin{cases} d_i - ECT_i & d_i < ECT_i \\ k \times (d_i - ECT_i) & d_i \geq ECT_i \end{cases}, \tag{5.6}$$

where,

d_i: Due date of job J_i in the waiting jobs

ECT_i: Expected completion time of job J_i based on the sequence before or after applying the selected dispatching rule.

k: Penalty factor is the expected completion time of job J_i is larger than the due date

This reward function quantifies the impact of the agent's dispatching decision on the expected completion times of the waiting jobs relative to their due dates. where a positive reward indicates an improvement in adherence to job due dates, effectively guiding the agent to favor decisions that reduce overall tardiness. To further emphasize adherence to due dates,

a penalty factor k is applied to jobs expected to exceed their due dates, amplifying the negative impact on the reward. The proposed reward function addresses a key challenge in decentralized scheduling, namely the lack of immediate feedback, by introducing a new local evaluation metric based on the change in average modified slack. This formulation has not appeared in prior work on RL-based dispatching rules and enables effective credit assignment at the work center level.

5.4. Numerical Experiments

This section presents numerical experiments for **Problem 2**, where a PPO-based agent is deployed to address scheduling under machine breakdown scenarios and then tested on a separate set of test instances to assess generalization performance. The goal is to examine the agent's adaptability and robustness by comparing it with conventional dispatching rules. Evaluation metrics include mean tardiness, win rate, and performance consistency across multiple random seeds. In addition, SHAP analysis is conducted to interpret the learned policy by identifying the most influential features that drive the decision-making process.

5.4.1. Experiment Setup

Simulation Environment Setup

The numerical experiments are conducted in a simulated environment programmed in Python, leveraging the PyTorch machine learning library for implementing the PPO agents. The simulation framework models a decentralized, event-driven decision-making system designed for DFJSP, incorporating stochastic job arrivals and machine breakdowns. The computational setup includes an Intel Core i5-8250U CPU at 1.8 GHz with 8 GB RAM, providing a practical configuration for agent training and evaluation.

Dataset and Problem Instance Configuration

The experimental evaluation is based on **Case 1C** (Flexible Routing Instances), emphasizing job routing flexibility and adaptability. In this scenario, jobs are dynamically generated with flexible routing options, meaning that job types can selectively skip certain work centers based on specific requirements. The processing times for operations vary uniformly between 1 and 99 units of time. Additional state features, such as statistical summaries (mean, minimum, maximum, standard deviation) of operations waiting and potential job types per work center, are incorporated to enhance agent decision-making. This case includes:

- A total of 20,000 episodes for robust training and evaluation.
- Each episode comprises 100 dynamically arriving jobs following an exponential inter-arrival distribution with mean intervals ranging from 5 to 15 units of time.
- Five work centers, each with a variable number of parallel machines ranging from 1 to 5.

This structured and flexible experimental setup provides a comprehensive basis for assessing the PPO scheduling framework's ability to manage dynamic and complex manufacturing scenarios.

PPO Training Hyperparameters

Several key hyperparameters define the PPO agent and critically influence its learning dynamics, policy stability, and final scheduling performance. Optimization is performed using the Adam optimizer, which adapts per-parameter learning rates based on estimated first and second moments of gradients. We systematically tuned the learning rate over the set $\{1 \times 10^{-4}, 3\times10^{-5}, 1\times10^{-5}\}$. A rate of 1×10^{-5} was selected because higher values caused oscillations in policy loss and occasional divergence—particularly during early training when reward signals are sparse and noisy—while lower rates led to negligible improvement over the 50,000-episode training horizon.

The clipping range (ε) controls the conservatism of policy updates and was evaluated in $\{0.1,0.2,0.3\}$. A value of 0.2 yielded the best trade-off: smaller clips (e.g., 0.1) overly constrained policy improvement and slowed convergence, whereas larger clips (e.g., 0.3) occasionally permitted destructive updates that degraded due-date adherence performance.

The discount factor (γ) governs the agent's emphasis on future rewards and was tested over $\{0.95,0.97,0.99\}$. We fixed $\gamma = 0.99$ because scheduling outcomes (e.g., tardiness) depend heavily on long-term sequencing decisions; a high discount factor ensures the agent accounts for delayed consequences of early dispatching choices, which is essential in environments with sparse immediate feedback.

The number of PPO epochs (K_{epochs})—i.e., the number of times the collected trajectory data are reused for policy updates—was tuned over $\{3,5,8,10\}$. Setting $K_{\text{epochs}} = 5$ provided optimal sample efficiency: fewer epochs underutilized the collected experience, while more epochs (e.g., 8 or 10) led to overfitting on recent trajectories and increased policy variance across episodes.

Entropy regularization is handled implicitly through a learnable log standard deviation parameter in the Gaussian action distribution, eliminating the need for a manually tuned entropy coefficient. This design automatically balances exploration and exploitation by adjusting uncertainty based on observed returns.

Finally, the GAE lambda parameter was set to 0.95 following standard recommendations [146], as preliminary tests showed minimal sensitivity in the range $[0.95,0.97]$. The complete hyperparameter configuration is summarized in **Table 5.1**.

Table 5.1. PPO agent hyperparameters.

Hyperparameter	Value	Description
Learning Rate	1e-5	Controls the rate at which the PPO agent updates its neural network parameters, ensuring stable training.
Epochs (K_epochs)	5	Determines the number of iterations over each collected dataset before performing an update.
Clipping Epsilon (ε)	0.2	Limits the extent of policy updates during each training epoch, contributing to training stability.
Entropy Regularization	Learnable log_std	Implicitly integrated into the action distribution via a learnable standard deviation parameter, encouraging exploration by penalizing overly deterministic policy distributions.
Gamma (Discount Factor)	0.99	Emphasizes the importance of future rewards, thus supporting long-term decision-making strategies.
GAE Lambda	0.95	Controls the trade-off between bias and variance in advantage estimation.
Optimizer	Adam	Selected for its adaptive learning rate capabilities, promotes stable convergence and training efficiency
Loss Function	MSE	Measures the deviation between predicted and target value estimations, drives effective policy improvements throughout the training process.

Stochastic Breakdown Parameters

Machine breakdowns are modeled as stochastic events characterized by MTBF and MTTR. The MTBF for each machine is randomly set between 500 and 1500 units of operation time, following an exponential distribution. Repair times are similarly stochastic, following an exponential distribution with a mean repair duration of 50 units of simulation time. These parameters realistically represent the unpredictability inherent in manufacturing systems and provide a robust environment for evaluating dynamic scheduling performance.

5.4.2. Training Process

Training PPO Agents in Distributed Work Centers

Each PPO agent is independently trained within distributed work centers to enable scalable and decentralized decision-making capabilities.

- **Episode Length and Number of Episodes:** The training process involves a total of 20,000 episodes, each comprising 100 jobs dynamically entering the system, allowing agents to adapt progressively to diverse and stochastic job sequences.

Initial Conditions and Randomization Protocols

Initial conditions are uniformly randomized for each episode to prevent overfitting and encourage generalization. Randomization includes job release patterns, processing times, and machine breakdown intervals, ensuring variability and robustness in agent performance evaluation.

The PPO agent is configured with a state size of 53, capturing detailed environmental observations such as job counts, machine statuses, and operational statistics. The action size is 4, corresponding to composite priority scores used for job scheduling decisions. **Figure 5.3** shows the structure of the neural network used in PPO. The neural network architecture

includes a shared trunk beginning with an input layer of 53 units, followed by three fully connected hidden layers with 128, 256, and 128 units, respectively. Each hidden layer employs the Leaky ReLU activation function (negative slope of 0.01) to support stable gradient flow. After the shared trunk, the network branches into two separate heads: one for the actor, which outputs a probability distribution over feasible actions, and one for the critic, which estimates the state value.

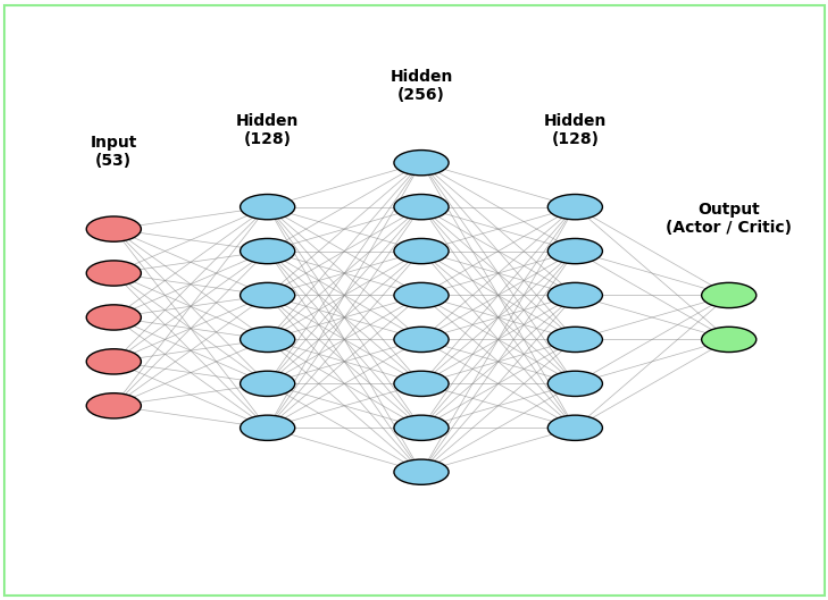
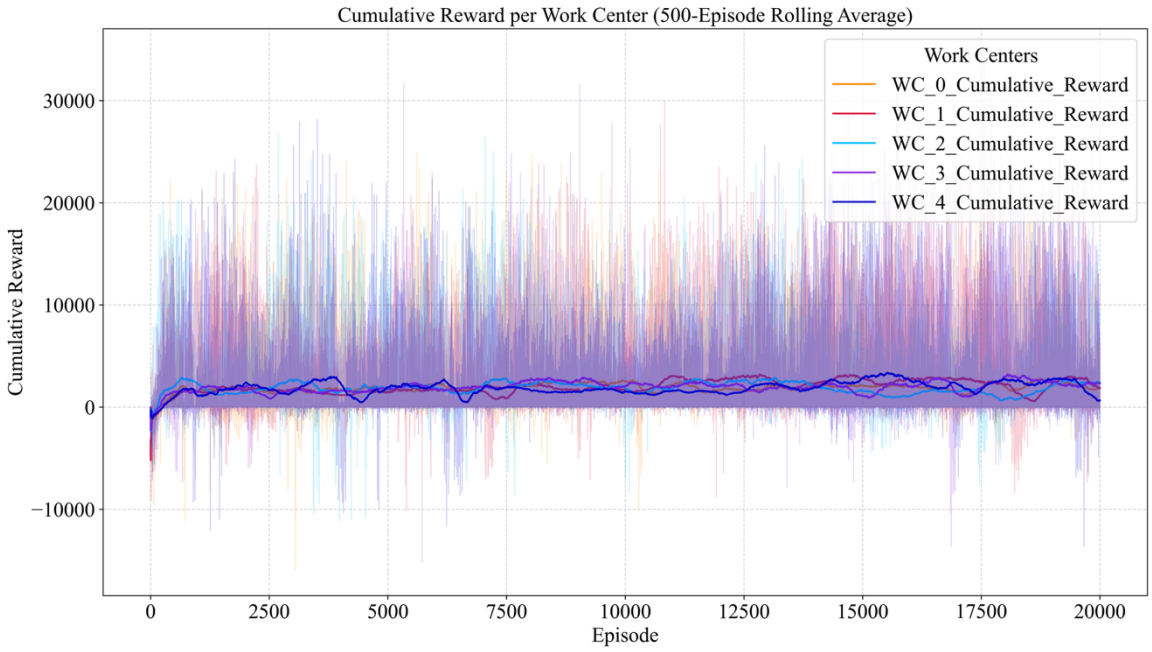


Figure 5.3. Multi-layer perceptron network architecture of PPO agent (shared trunk architecture).

Training Results



(a)

(b)

Figure 5.4. Problem 2: (a) Cumulative reward per work center over training episodes (500-episode rolling average). The plot shows the learning progress of individual work centers (WC_0 to WC_4), where each line represents the moving average of cumulative rewards.

(b) Evolution of the logarithm of standard deviation for each work center during training.

Analysis of the training process from the cumulative reward and log standard deviation (log_std) graphs reveals complex learning dynamics that reflect both progress and instability. The cumulative rewards across all work centers exhibit high variance and do not show a consistent upward trend. However, in some work centers such as WC0, WC1, and WC3, fluctuations appear to diminish after approximately 12,000 episodes, suggesting possible convergence toward a stable, albeit suboptimal, policy.

The noise in the reward curves is expected, given the stochastic nature of job arrivals, machine breakdowns, and dynamic dispatching decisions. Each episode presents a unique scheduling scenario, which leads to inherent variability in cumulative rewards. Consequently, the absence of a monotonic increase does not necessarily indicate failed learning; rather, it reflects the challenge of evaluating long-term performance through short-term reward signals in a decentralized setting.

At the same time, the log_std parameter shown in **Figure 5.4b** consistently decreases across all work centers over training. This indicates that agents gradually reduce their action uncertainty, shifting from exploration to exploitation. Such a trend confirms that learning is occurring, even if the policy stabilizes within a local optimum rather than discovering globally superior strategies.

These results were obtained using the hyperparameter configuration described in **Table 5.1**, which was selected based on preliminary tuning of the learning rate, clipping range, discount factor, and number of PPO epochs. Although techniques such as adaptive learning rates or curriculum-based training could potentially improve convergence stability, they were not explored in this study due to computational constraints.

In the context of flexible job shop scheduling, even modest improvements in average

performance can be meaningful. The problem's complexity, combined with limited coordination between work centers, makes consistent gains over simple heuristics (e.g., FIFO or SPT) non-trivial. Therefore, while the training process remains noisy and full convergence is not achieved, the learned policies still demonstrate practical utility.

5.5. Results and Analysis

In **Problem 2**, the performance of the PPO-WF (Weighted Sum of Features) agent was comprehensively evaluated against both traditional dispatching rules, namely FIFO (First-In-First-Out), SPT (Shortest Processing Time), and EDD (Earliest Due Date), and the baseline reinforcement learning variant PPO_RS (Rule Selection). To better contextualize the comparison, we clarify the two PPO-based agents evaluated in **Problem 2**.

PPO-RS (Rule Selection) uses a discrete action space in which each action corresponds to selecting one of eight predefined dispatching rules, such as SPT, FIFO, EDD, or COVERT. This approach mirrors the design of the earlier DDQN agent and prioritizes interpretability and implementation simplicity. However, as noted in **Section 4.4**, it restricts the scheduler to a fixed set of heuristics and cannot generate novel or hybrid dispatching behaviors.

In contrast, PPO-WF (Weighted Features) employs a continuous action space. The policy outputs a weight vector that linearly combines normalized job features, including remaining processing time, due date urgency, and queue length, to compute a priority score for each waiting job. This enables fine-grained, state-dependent prioritization and allows the agent to interpolate between traditional rules or construct new strategies tailored to the current system state.

The transition from PPO-RS to PPO-WF addresses a key limitation of discrete frameworks. Under dynamic disruptions, such as machine failures or urgent job insertions, rigid rule selection often fails to adapt optimally. By learning continuous weights over interpretable

features, PPO-WF retains transparency while gaining flexibility to respond effectively to complex and evolving conditions.

As illustrated in **Figure 5.5** and summarized in **Table 5.2**, PPO-WF achieved the highest win rate of 40.0%, outperforming all other methods. PPO-RS followed with 30.0%, and the best traditional heuristic rule, SPT, reached 27.0%. Other traditional rules, such as EDD and LOR, trailed significantly, achieving win rates of only 2.0% and 1.0%, respectively. This result alone highlights the consistent superiority of learning-based approaches, particularly PPO-WF, under dynamic and uncertain scheduling environments.

In terms of average tardiness, PPO-WF again showed dominant performance, achieving the lowest mean tardiness of 653.65, compared to 671.65 by PPO-RS and 672.51 by SPT. This gap, though numerically moderate, becomes statistically and operationally meaningful when viewed in the context of complex job arrival patterns and varying machine availability.

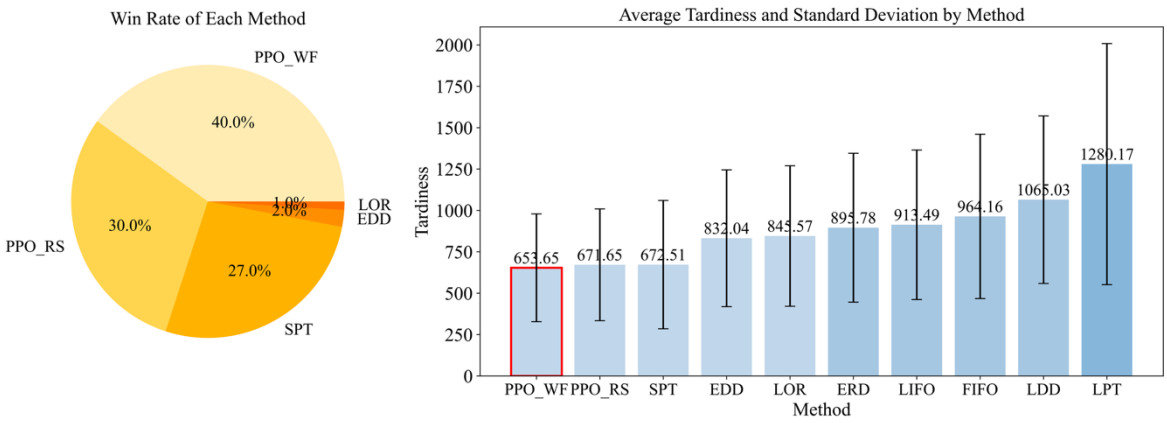


Figure 5.5. Win rate and average tardiness of Problem 2.

Table 5.2. Win rate and average tardiness of Problem 2.

Methods	Win Rate	Mean Tardiness
PPO-WF	40.0%	653.65
PPO-RS	30.0%	671.65
SPT	27.0%	672.51
EDD	2.0%	832.04
LOR	1.0%	845.57

A deeper examination of the scheduling mechanisms helps explain the performance differences observed in **Table 5.2**. PPO-WF achieves the lowest mean tardiness (653.65) among all methods and a win rate of 40.0%, outperforming PPO-RS, which attains 671.65 mean tardiness and a 30.0%-win rate. This 18-unit reduction in mean tardiness (≈2.7%) reflects PPO-WF's ability to make fine-grained, context-sensitive decisions. Unlike PPO-RS, which selects from a fixed set of dispatching rules (e.g., SPT or EDD) and thus applies a single priority logic at each decision point, PPO-WF dynamically computes a weighted composite score for each job-machine pair using engineered features such as slack, remaining workload, and machine availability. This enables it to balance competing objectives (e.g., avoiding bottlenecks while respecting due dates) in ways that static rules cannot. The advantage is further evident when comparing against traditional heuristics: EDD and LOR, despite being due-date-focused, suffer from high mean tardiness (832.04 and 845.57, respectively) and near-zero-win rates (2.0% and 1.0%), underscoring their rigidity in dynamic environments. In contrast, PPO-WF's adaptive policy not only improves average performance but also increases the likelihood of producing the best schedule across diverse instances, as reflected in its highest win rate.

The advantage of PPO-WF becomes especially apparent in high-variability scenarios. Unlike traditional rules that rely on fixed sorting logic, PPO-WF learns to balance multiple factors—such as due date tightness, remaining operations, and expected wait times—based

on empirical outcomes during training. This capability allows the agent to perform consistently across the diverse set of scheduling instances used in our evaluation, suggesting reduced sensitivity to instance-specific characteristics.

Although PPO-WF outperforms both rule-based heuristics and the discrete RL baseline, the practical value of this improvement depends on computational cost, deployment constraints, and the operational context. The primary cost of PPO-WF lies in offline training, which requires substantial interaction with the simulator but incurs no runtime overhead during deployment. At inference time, the policy computes job priorities via a lightweight neural network, adding only ~1–2 ms per decision compared to the near-instantaneous evaluation of SPT. Given that dispatching decisions in most manufacturing systems occur on timescales of seconds or minutes, this marginal latency is negligible.

Moreover, although the reduction in mean tardiness relative to SPT is modest (approximately 2.7%), its operational significance should not be underestimated. In high-volume or high-value production environments, even small reductions in tardiness can translate into substantial cost savings. This is especially true in industries such as electronics assembly or aerospace manufacturing, where savings come from avoided contractual penalties, reduced reliance on expedited shipping, and better on-time delivery performance. Critically, PPO-WF also achieves the best result in 40% of test instances (**Table 5.2**), compared to 27% for SPT, indicating greater robustness across diverse and stochastic conditions. This reliability is particularly valuable when system dynamics shift unexpectedly due to machine failures or urgent orders, where static rules like SPT may perform suboptimally.

Therefore, while SPT remains an excellent choice for simple, stable environments, PPO-WF offers a justifiable trade-off for complex, dynamic settings where adaptability and consistent performance outweigh minimal increases in computational load.

5.6. SHAP-Based Feature Attribution Analysis

Table 5.3. SHAP-based feature importance across work centers in Case 2 (rows = features, columns = work centers). Each cell reports the mean absolute SHAP value and its within-WC rank: "value (rank)". Features are ordered by domain relevance and consistent appearance across centers.

Feature	WC0	WC1	WC2	WC3	WC4
Num_Machines_in_WC	3.955 (1)	7.093 (1)	3.375 (1)	6.535 (1)	2.707 (1)
Mean_PT_WC	2.075 (2)	3.845 (2)	1.675 (2)	2.895 (2)	1.388 (2)
Breakdown_Counts_in_WC	1.237 (3)	1.836 (6)	0.659 (6)	1.664 (4)	0.662 (6)
Max_Time_Since_Last_Repair	1.147 (4)	2.129 (3)	0.840 (4)	1.936 (3)	0.865 (4)
Max_PT_WC	1.112 (5)	1.923 (5)	1.087 (3)	1.525 (5)	0.903 (3)
Mean_Time_Since_Last_Repair	0.892 (6)	1.968 (4)	0.820 (5)	1.198 (6)	0.748 (5)
Potential_Job_Type_Number_in_WC	0.600 (7)	0.589 (14)	0.344 (12)	0.714 (9)	0.355 (10)
Event_Operation_Complete_Next_WC	0.534 (8)	0.706 (12)	0.217 (18)	0.636 (13)	0.555 (7)
Event_Job_Arrival	0.454 (9)	0.370 (25)	0.177 (23)	0.702 (11)	0.333 (12)
Min_PT_WC	0.406 (10)	0.839 (8)	0.415 (9)	0.784 (7)	0.429 (8)
Min_Time_Since_Release	0.405 (11)	0.827 (9)	0.387 (11)	0.443 (18)	0.273 (14)
Std_Time_Since_Last_Repair	0.347 (12)	0.552 (16)	0.588 (7)	0.713 (10)	0.341 (11)
Min_Work_Since_Last_Repair	0.338 (13)	1.095 (7)	0.512 (8)	0.625 (14)	0.359 (9)
Other_WCs_Available_Machines	0.314 (14)	0.463 (22)	0.216 (19)	0.588 (15)	0.163 (21)
Other_WCs_Waiting_Jobs	0.303 (15)	0.708 (11)	0.226 (17)	0.742 (8)	0.281 (13)

Num_Job_Types_in_System	0.282 (16)	0.117 (43)	0.216 (20)	0.112 (40)	0.207 (16)
Min_Time_Since_Last_Repair	0.278 (17)	0.611 (13)	0.037 (43)	0.639 (12)	0.201 (17)
Max_Operations	0.276 (18)	0.202 (33)	0.045 (40)	0.243 (30)	0.170 (19)
Mean_Time_to_Due_Date	0.263 (19)	0.514 (18)	0.242 (14)	0.263 (27)	0.175 (18)
Max_Time_to_Due_Date	0.256 (20)	0.546 (17)	0.234 (15)	0.232 (31)	0.168 (20)
Mean_Time_Since_Release	0.247 (21)	0.557 (15)	0.280 (13)	0.344 (23)	0.157 (25)
Min_Remaining_PT	0.219 (22)	0.513 (19)	0.153 (25)	0.402 (19)	0.159 (23)
Max_Time_Since_Release	0.208 (23)	0.448 (23)	0.195 (21)	0.316 (24)	0.130 (26)
Mean_Work_Since_Last_Repair	0.207 (24)	0.190 (34)	0.141 (27)	0.064 (45)	0.050 (39)
Min_Time_to_Due_Date	0.185 (25)	0.472 (21)	0.191 (22)	0.266 (26)	0.093 (30)
Event_Machine_Breakdown	0.183 (26)	0.261 (31)	0.131 (30)	0.344 (22)	0.212 (15)
Mean_Remaining_PT	0.166 (27)	0.127 (40)	0.063 (36)	0.053 (47)	0.069 (36)
Min_Waiting_Time	0.157 (28)	0.473 (20)	0.397 (10)	0.350 (21)	0.086 (31)
Min_Remaining_Operations	0.143 (29)	0.075 (46)	0.077 (33)	0.263 (28)	0.157 (24)
Max_Remaining_Operations	0.137 (30)	0.306 (28)	0.047 (39)	0.279 (25)	0.077 (34)
Mean_Operations	0.136 (31)	0.097 (44)	0.021 (47)	0.040 (48)	0.076 (35)
Std_Repaired_Time	0.131 (32)	0.414 (24)	0.142 (26)	0.485 (16)	0.102 (29)
Std_Work_Since_Last_Repair	0.124 (33)	0.207 (32)	0.133 (29)	0.057 (46)	0.079 (33)
Max_Remaining_PT	0.120 (34)	0.125 (41)	0.041 (41)	0.072 (43)	0.081 (32)
Std_PT_WC	0.108 (35)	0.292 (29)	0.079 (32)	0.393 (20)	0.053 (37)
Max_Repaired_Time	0.100 (36)	0.826 (10)	0.174 (24)	0.471 (17)	0.161 (22)

Event_Machine_Repaired	0.097 (37)	0.156 (37)	0.027 (45)	0.183 (32)	0.110 (28)
Mean_Waiting_Time	0.083 (38)	0.265 (30)	0.133 (28)	0.154 (36)	0.041 (40)
Min_Operations	0.070 (39)	0.321 (26)	0.052 (38)	0.115 (39)	0.113 (27)
Num_Broken_Machines	0.070 (40)	0.140 (38)	0.095 (31)	0.168 (33)	0.052 (38)
Max_Waiting_Time	0.058 (41)	0.167 (36)	0.068 (34)	0.105 (41)	0.018 (46)
Max_Work_Since_Last_Repair	0.055 (42)	0.174 (35)	0.227 (16)	0.165 (34)	0.032 (42)
Mean_Remaining_Operations	0.054 (43)	0.076 (45)	0.056 (37)	0.157 (35)	0.020 (44)
Std_Operations	0.054 (44)	0.072 (47)	0.030 (44)	0.092 (42)	0.007 (49)
Mean_Repaired_Time	0.041 (45)	0.131 (39)	0.019 (49)	0.116 (38)	0.039 (41)
Std_Remaining_PT	0.036 (46)	0.051 (48)	0.019 (48)	0.065 (44)	0.024 (43)
Std_Remaining_Operations	0.019 (47)	0.119 (42)	0.066 (35)	0.147 (37)	0.018 (47)
Std_Waiting_Time	0.018 (48)	0.037 (50)	0.022 (46)	0.024 (49)	0.010 (48)
Num_Waiting_Jobs_in_WC	0.013 (49)	0.307 (27)	0.040 (42)	0.262 (29)	0.019 (45)
Min_Repaired_Time	0.009 (50)	0.038 (49)	0.006 (50)	0.018 (50)	0.006 (50)
Std_Time_to_Due_Date	0.001 (51)	0.018 (51)	0.004 (51)	0.011 (51)	0.002 (51)
Std_Time_Since_Release	0.001 (52)	0.004 (52)	0.000 (52)	0.005 (52)	0.001 (52)
Event_Operation_Complete	0.000 (53)	0.000 (53)	0.000 (53)	0.000 (53)	0.000 (53)

To provide a deeper understanding of the learned scheduling policy under dynamic and failure-prone environments, this section investigates the contribution of different input features through SHAP analysis. The PPO-WF agent's actions are interpreted by attributing its decisions to specific features, thereby uncovering the underlying rationale in distributed and partially observable settings.

SHAP values represent the marginal contribution of each feature to the model's output, computed based on coalitional game theory. In this study, the SHAP analysis was performed over the PPO-WF policy for five selected work centers (WC0–WC4), each equipped with multiple parallel machines. The agent's decision-making behavior is analyzed in terms of the mean absolute SHAP value, which reflects the average impact of each feature across a wide range of scheduling instances. **Table 5.3** illustrates the top contributing features for each work center. Across all five cases, three consistent observations can be drawn:

1. Dominance of Local Resource Attributes:

The most influential feature across all work centers is *Num_Machines_in_WC*, which directly reflects the available processing capacity. This is closely followed by *Mean_PT_WC* and *Max_PT_WC*, which indicate the processing workload of jobs assigned to that work center. The agent thus exhibits a strong preference toward allocating jobs based on local capacity and processing burden, aligning with the decentralized nature of the scheduling system.

2. Incorporation of Machine Health and Repair History:

Features such as *Max_Time_Since_Last_Repair*, *Breakdown_Counts_in_WC*, and *Std _Time_Since_Last_Repair* appear consistently among the top 10 features. This highlights that the agent is sensitive to machine breakdown risks and prioritizes more reliable resources when allocating jobs. This is an essential capability for dynamic environments with stochastic disruptions.

3. Moderate Consideration of Temporal and Systemic Context:

Although not dominant, several features related to job urgency (*Min_Time_to_Due_Date*, *Mean_Time_Since_Release*), system congestion (*Other_WCs_Waiting_Jobs*, *Other_WCs _Available_Machines*), and operational events (*Event_Operation_Complete_Next_WC*) still

contribute to the decision process. This reflects a learned policy that balances local optimization with limited global context under a distributed framework. These findings validate the effectiveness of the proposed state representation and confirm that the PPO_WF agent can capture relevant environmental signals that correlate with robust scheduling performance. Importantly, the high consistency of key features across different work centers suggests the potential benefit of policy sharing or parameter tying in future work.

These findings validate the effectiveness of the proposed state representation and confirm that the PPO-WF agent can capture relevant environmental signals that correlate with robust scheduling performance. Importantly, the high consistency of key features across different work centers suggests the potential benefit of policy sharing or parameter tying in future work.

Further interpretation of the SHAP rankings reveals that the agent's policy closely mirrors real-world operational priorities. The overwhelming emphasis on *Num_Machines_in_WC* and processing time extremes (*Max_PT_WC*) reflects a bottleneck-aware strategy: in failure-prone environments, ensuring sufficient capacity and avoiding overloaded resources is more critical than fine-grained load balancing. Notably, the agent treats machine health not through average metrics but via worst-case indicators (e.g., *Max_Time_Since_Last_Repair*), effectively modeling equipment as a reliability risk rather than a static asset—a behavior aligned with predictive maintenance practices. Perhaps most revealing is what the agent ignores: global congestion signals and variability measures (e.g., *Std_PT_WC*) rank consistently low, suggesting that under partial observability, the policy favors stable, locally grounded decisions over noisy or delayed system-wide information. This "pragmatic localism" not only enhances robustness but also offers a compelling explanation for the agent's strong empirical performance in dynamic settings.

5.7. Implications of the Study

The findings from **Problem 2** hold several critical implications for both the development of intelligent scheduling algorithms and their deployment in dynamic manufacturing systems:

- **Improved Adaptability through Continuous Prioritization**: By transitioning from discrete rule selection to a continuous feature-weighted prioritization approach, the proposed PPO-WF method enhances the system's responsiveness to real-time disturbances such as machine breakdowns. By computing job-machine scores based on real-time features, the approach is designed to support adaptive scheduling under fluctuating job demands and equipment availability—conditions reflected in our experimental setup.
- **Superior Scheduling Performance over Baselines**: Compared to both heuristic methods (SPT, EDD, FIFO) and previous reinforcement learning strategies (PPO-RS), the PPO-WF agent consistently achieved better results in terms of mean tardiness and win rate. The reinforcement learning model not only improved the average performance but also demonstrated lower variance across simulations, suggesting more reliable and robust scheduling under uncertainty.
- **Integration of Disruption Handling into Learning**: **Problem 2** incorporated machine breakdowns directly into the agent's state representation, enabling the PPO agent to proactively account for system reliability in its scheduling decisions.
- **Scalability and Generalization for Industry 4.0 Systems**: The decentralized PPO-based architecture supports scalability by allowing each work center to make autonomous decisions. The agent's learning process, grounded in continuous control and local observations, supports adaptability to large-scale, interconnected production systems that define the Industry 4.0/5.0 landscape.

5.8. Summary

This chapter presented **Problem 2**, which advanced the scheduling framework by integrating a continuous prioritization mechanism via the PPO algorithm. Building on the limitations identified in **Problem 1**, particularly regarding the rigidity of discrete rule selection, the new approach introduced the following key improvements and results:

- The PPO-WF agent outperformed both traditional heuristic rules and the previous PPO-RS agent. It achieved the lowest mean tardiness (653.65) and the highest win rate (40%), indicating both higher efficiency and greater robustness under dynamic conditions.
- The use of continuous feature weighting provided more flexibility in job selection and sequencing, allowing for nuanced prioritization strategies that adapt to real-time disruptions, including machine failures.
- The PPO framework exhibited stable convergence during training, with a decreasing log standard deviation indicating a reduction in policy stochasticity and a shift toward deterministic action selection.
- These results demonstrate that reinforcement learning with continuous action spaces can serve as a competitive alternative to rule-based heuristics in complex scheduling tasks. The method aligns well with the goals of smart manufacturing: flexibility, autonomy, and real-time responsiveness.

Together, these advancements confirm the value of feature-weighted, RL-based prioritization as a powerful and scalable strategy for modern production systems. **Problem 2** builds a strong foundation for further extending the framework to multi-objective and more intricate dynamic environments, as explored in **Problem 3**.

Chapter 6. Problem 3 Multi-Objective Scheduling with Sequence-Dependent Setup Times with RL

6.1. Introduction

In this chapter, the DFJSP is extended with sequence-dependent setup times and is further formulated as a multi-objective optimization problem. The presence of setup time dependencies adds significant complexity to the scheduling problem. Machines must consider not only the processing durations but also the transitional costs incurred when switching between different job types. This increases both the temporal and combinatorial difficulty of the scheduling decisions.

This added complexity introduces a dilemma between minimizing average job tardiness and reducing total setup time. These two objectives often conflict in practical scenarios: on the one hand, prioritizing jobs with imminent due dates can lead to frequent setup changes, which increase the non-productive time. Hence, it lowers the system efficiency, such as throughput. On the other hand, grouping similar job types to reduce setup overhead may delay high-priority jobs, resulting in tardiness, customer dissatisfaction, and potential contract penalties. Conventional approaches, such as dispatching rules, exact optimization (e.g., MILP), and metaheuristics, typically handle a single objective or rely on fixed weight combinations. These methods lack the flexibility to dynamically adapt to shifting priorities and often require extensive manual tuning. Traditional RL methods, such as policy iteration [147], value iteration, and tabular Q-learning, also struggle under these conditions. First, they rely on scalarized reward functions that collapse multiple objectives into a single signal. This limits their ability to explore the trade-off surface (i.e., the Pareto front), often producing biased solutions aligned with predefined weight preferences. Second, due to the large state and action spaces in DFJSP, these tabular methods are computationally infeasible.

While DRL offers the representational capacity to handle complex decision space, such as demonstrated in domains like Go, robotics, racing, and video games, it still suffers from the same scalarization bottleneck when applied to multi-objective problems [76], [77], [81]. Without a formulation that explicitly captures and generalizes across different objective trade-offs, even DRL tends to converge to a policy that overfits one side of the objective.

To address the challenge in multi-objective dynamic scheduling problems, the scheduling problem is formulated as a sequential decision-making process under the framework of an MDP. Based on this formulation, a DRL framework is proposed to optimize dynamic scheduling decisions in real time. More specifically, a DDQN is employed as the core learning algorithm due to its improved stability over standard Q-learning in value approximation. Furthermore, to enable effective learning across multiple objective trade-offs, the DDQN framework is further enhanced by integrating a UVFA. The UVFA conditions the value function not only on the environment state but also on a goal vector representing objective weights. By projecting a shared policy space onto varying objective preferences, the agent learns to approximate a family of value functions rather than a single scalar objective. This generalization ability enables the agent to explore a diverse set of policies along the Pareto front, without retraining for each trade-off configuration. As a result, the proposed framework can flexibly adapt to changing production priorities, such as shifting focus from delivery speed to energy efficiency or setup minimization.

To support generalization across objectives, it is important to emphasize that the reward function plays a central role in guiding policy learning in DRL. A baseline-referenced reward function is further designed, in which the performance of the RL-generated schedule is compared to heuristic baselines for both tardiness and setup time. This relative formulation provides more stable and directional reward signals, reduces variance across episodes, and accelerates convergence.

In summary, the contributions in this chapter are as follows:

- The dynamic, sequence-dependent flexible job shop problem is formulated as a Markov Decision Process (MDP), enabling stepwise optimization through reinforcement learning.
- A DDQN-based learning framework is proposed, designed to operate under decentralized, event-driven environments using a rule selection method.
- A Universal Value Function Approximator (UVFA) is integrated to enable multi-objective learning and generalization across varying objective preferences, allowing the agent to efficiently approximate the Pareto front.
- A baseline-referenced dual-objective reward function is introduced into the DRL framework, capturing relative improvements in tardiness and setup efficiency, which leads to improved sample efficiency and more stable training.

The chapter is structured as follows. **Section 6.2** describes the core methodology, beginning with the motivation behind incorporating sequence-dependent setup times and conflicting objectives. **Section 6.2.2** details the enhancements made to the scheduling architecture through rule selection mechanisms, followed by **Section 6.2.3**, which introduces a UVFA-based multi-objective reinforcement learning framework. **Section 6.3** presents the numerical experiments, including the environment configuration, neural network design, training and testing results, and evaluation using SHAP analysis. **Section 6.4** discusses the implications of the study's findings, and **Section 6.5** concludes the chapter with a summary of key insights.

6.2. Methodology

6.2.1. Motivation

In the DFJSP, the inclusion of sequence-dependent setup times introduces a substantial

increase in both temporal and combinatorial complexity. Unlike traditional models where setup durations are either constant or negligible, real-world manufacturing systems often incur significant time penalties when transitioning between jobs of differing types [148]. These sequence-dependent setup times are particularly prevalent in high-mix low-volume production systems, where frequent product switches, such as those driven by personalized manufacturing demands, necessitate additional preparatory tasks like tool changes, machine cleaning, or recalibration [21], [149].

This complexity manifests along three key dimensions. First, the scheduling process must account for temporal complexity, as setup durations are no longer constant but depend on the specific sequence of jobs. Consequently, the scheduler must evaluate not only job-to-machine assignments but also the ordering of jobs on each machine, since job sequences directly influence total processing time. Second, the system faces combinatorial complexity, as each additional job type introduces new permutations in the setup time matrix. This expansion leads to an exponential increase in the number of possible scheduling configurations, substantially enlarging the decision space and complicating the search for optimal solutions. Third, the inclusion of dynamic events, such as stochastic job arrivals and varying machine availability, further amplifies the complexity by introducing uncertainty into the system. This requires the scheduling policy to not only solve a high-dimensional optimization problem but also to adapt continuously in real time, which poses a significant challenge for conventional deterministic methods. In this enriched scheduling environment, the problem is further characterized by conflicting objectives. On one hand, minimizing mean job tardiness aligns with operational goals such as on-time delivery and customer satisfaction. On the other hand, minimizing total setup time contributes to operational efficiency, energy sustainability, and equipment reliability. However, these two objectives are inherently in conflict. For instance, prioritizing early due dates might necessitate frequent job type switches, increase setup time, while group similar jobs to reduce setup time can lead to delays for urgent jobs.

Existing approaches typically simplify this trade-off by scalarizing the multi-objective problem into a single-objective reward function, commonly using a weighted-sum method [115]. While this allows the application of conventional reinforcement learning techniques, such scalarization inherently reduces the solution space to a subset of the Pareto front. Consequently, it is unable to adaptively balance the tardiness and setup time, which is particularly limiting in dynamic environments where objective priorities may shift over time.

In response to the high complexity in multi-objective RL problems, some studies have explored alternative approaches based on multiple policy networks, where each policy is trained to optimize a specific objective weighting or trade-off configuration [150], [151]. This strategy can partially improve coverage of the Pareto front by learning a set of discrete solutions. However, it also introduces significant training overhead, as each policy must be trained separately, and it still lacks the ability to generalize across unseen or continuously varying objective preferences.

For instance, Luo et al. [116] proposed a hierarchical reinforcement learning framework that dynamically selects from a fixed set of predefined objectives to improve adaptability. While this method increases flexibility compared to static scalarization, it remains confined to a limited discrete set of trade-offs and does not support generalization across the continuous objective space. Leng et al. [152] employed a 2D-folded normal distribution (2D-FND) for preference sampling and trained a separate policy for each preference. While effective in approximating the Pareto front, the method requires a high computational cost, even with parallelism and shared structures, making it unsuitable for dynamic scheduling. Despite these advantages, it cannot achieve generalization by ignoring similarities across preferences, limiting adaptability to unseen or changing objectives.

To address these limitations in generalization for DRL, a reinforcement learning framework capable of generalizing across multiple objective trade-offs is proposed. Instead of relying

on scalarized rewards or multiple separately trained policies, the framework learns a value function conditioned on objective preferences, enabling a single agent to adapt its scheduling strategy dynamically. This is achieved by reformulating the value function within the DDQN framework to incorporate objective-aware conditioning, where the estimated value is jointly determined by the environment state and a vector representing the relative importance of multiple objectives. This allows the agent to approximate a range of Pareto-optimal policies without retraining for each new trade-off configuration. Our design not only reduces training cost but also improves flexibility and sample efficiency. As shown in feature attribution analyses in **Section 4.2.1**, the agent learns to rely on general scheduling features that remain effective across different objectives. This approach offers a practical and scalable solution for dynamic scheduling in Industry 4.0 environments, where objectives such as tardiness and setup efficiency often conflict and shift over time.

6.2.2. Enhancements Scheduling Architecture Based on Rule Selection

While the proposed reinforcement learning framework enables adaptive policy learning under shifting objective preferences, its effectiveness also depends on the underlying scheduling architecture and decision representation. In particular, the integration of reinforcement learning with rule-based heuristics provides a practical mechanism for making interpretable and efficient scheduling decisions in real time.

In **Problem 1**, scheduling decisions are made by selecting from a predefined set of dispatching rules, such as FIFO, EDD, or SPT. This rule selection framework serves as a lightweight yet effective decision model, suitable for environments with moderate complexity. However, in **Problem 3**, where sequence-dependent setup times and multi-objective trade-offs are explicitly modeled, this basic structure becomes insufficient. The scheduling system must now consider not only job urgency but also setup-aware behavior, requiring a more expressive state representation, additional dispatching rules, and a refined

reward structure.

The following section outlines the enhancements to the scheduling architecture necessary to address these challenges, detailing how the rule selection mechanism is expanded and restructured to support effective decision-making in more complex dynamic environments.

(1) New Priority Rules

To address the increased complexity arising from sequence-dependent setups, which extends beyond the rule selection framework of **Problem 1,** a set of new dispatching rules is implemented to reflect both traditional and setup-sensitive heuristics.

Unlike **Problem 1**, where the selected dispatching rule was applied only once at each decision point to generate a static job sequence based on the current system state, the application of that rule did not need to account for changing machine conditions after the decision was made. Once the job order was determined, it remained fixed until the next scheduling event (e.g., a new job arrival or a machine becoming idle). This was feasible because setup times were not sequence-dependent, and the choice of machine or job had no backward impact on previously determined sequences. In contrast, **Problem 3** introduces dynamic setup conditions that require additional adjustments. Specifically, because the setup time depends on the job type previously processed by the first available machine, the job sequence must be re-evaluated and re-sorted each time an action is applied. This ensures that the selected rule remains continuously valid as machine availability changes during runtime.

The newly added rules are as follows:

- **Shortest Setup Time (SST):** Chooses the job requiring the least setup time from the previously processed job type or the first-job setup time if the machine is idle.
- **Longest Setup Time:** Prioritizes jobs that incur the highest setup time relative to the

previous job type on the selected machine, emphasizing transitions with the greatest configuration overhead.

- **Shortest Setup Adjusted Processing Time (SSAPT):** Combines setup time and processing time into a unified metric.
- **Longest Setup Adjusted Processing Time:** Prioritizes the job with the highest combined setup and processing cost.

(2) MDP Modeling-State Representation:

In **Problem 3**, sequence-dependent setup times and conflicting objectives introduced greater complexity than in **Problem 1** and **Problem 2**, where setups were uniform or not explicitly modeled. To support effective decision-making in this setting, the state vector was expanded with additional features, as the basic representation used in **Problem 1** proved insufficient to capture the information required for high-quality scheduling decisions.

Therefore, the additional features, such as setup time statistics, job-type diversity, and workload-related indicators, are designed to precisely reflect the factors that directly influence scheduling outcomes in more complex environments. These extensions help the policy build a more accurate picture of the system status, allowing the agent to recognize important patterns, evaluate trade-offs more effectively, and adapt its policy in real time.

- **Potential Job Type Diversity:** Measures the number of unique job types currently waiting in the work center. A high value signals greater heterogeneity, which may lead to increased cumulative setup time.
- **Setup Time Statistics for the First Available Machine**: This state feature captures the distribution of setup times required to process all waiting jobs on the first available machine within a work center. Specifically, it includes four normalized statistical descriptors:
  - **Mean Setup Time:** Average setup duration across all waiting jobs relative to the

previous job type processed on the selected machine.

- **Minimum Setup Time:** The smallest setup time required, indicating the most setup-efficient option available.
- **Maximum Setup Time:** The highest potential setup cost, signaling worst-case transitions.
- **Standard Deviation of Setup Time:** A measure of setup time variability, reflecting the scheduling risk and diversity in job-type transitions.

(3) MDP Modelling-Reward Functions

In reinforcement learning, the reward function plays a central role in shaping the agent's behavior. It defines the learning objective by assigning feedback signals that guide the agent toward desirable actions. A well-designed reward function is especially critical in complex environments, where the agent must learn to balance competing objectives. To directly support the learning of scheduling policies that balance two competing objectives, namely, timely job completion and the minimization of frequent job-type transition costs, this study introduces a dual-component reward function. By separately evaluating improvements in both tardiness and setup time, this design provides the agent with more targeted feedback, enabling it to make trade-off decisions that reflect the real-world demands of dynamic, setup-sensitive manufacturing environments.

Tardiness-Oriented Component

This component evaluates how well the RL-generated schedule aligns job completion times with due dates. For each job J_i, let C_i be the estimated completion time and D_i be the due date. A tardiness-based signal is computed as:

$$\Delta_i = D_i - C_i \tag{6.1}$$

where $\Delta_i > 0$ indicates early completion, $\Delta_i = 0$ on-time delivery, and $\Delta_i < 0$ tardiness.

Two variants of this signal are employed across experiments:

Tardiness-focused formulation: Early or on-time jobs yield no reward incentive, while late jobs incur a linear penalty. Formally,

$$\mathrm{TTD}_i = \begin{cases} 0, & \text{if } \Delta_i \geq 0 \\ \Delta_i, & \text{if } \Delta_i < 0 \end{cases} \tag{6.2}$$

Shaped formulation: Both early and late deviations are actively shaped—early completions receive positive reward, and tardiness is amplified by a penalty factor $\alpha > 1$ (typically $\alpha = 10$):

$$\mathrm{TTD}_i = \begin{cases} \Delta_i, & \text{if } \Delta_i \geq 0 \\ \alpha \cdot \Delta_i, & \text{if } \Delta_i < 0 \end{cases} \tag{6.3}$$

This component evaluates, at each decision step, how the agent's proposed ordering of the currently waiting jobs compares to that of a due-date-aware heuristic baseline. Specifically, let $\mathcal{W}$ denote the set of unscheduled jobs at the current decision point. The PPO-WF agent assigns a priority score to each job in $\mathcal{W}$, yielding an ordered sequence π_{RL} . Independently, the Shortest Setup Adjusted Processing Time (SSAPT) rule is applied to the same set $\mathcal{W}$, producing a baseline sequence π_{SSAPT} .

Both sequences are then evaluated by simulating their immediate execution: for each, we estimate the resulting average time-to-due, denoted $\overline{\mathrm{TTD}}_{\mathrm{RL}}$ and $\overline{\mathrm{TTD}}_{\mathrm{SSAPT}}$, respectively. The tardiness-oriented reward is defined as the improvement of the RL policy over the baseline:

$$r_{\mathrm{tardiness}} = \overline{\mathrm{TTD}}_{\mathrm{RL}} - \overline{\mathrm{TTD}}_{\mathrm{baseline}} \tag{6.4}$$

A positive value means that the RL-generated schedule handles job urgency better than the baseline rule. This reward component provides the agent with a direct performance signal on how well it handles urgency and delivery reliability, which is especially important in

systems where delays can lead to customer dissatisfaction or missed production targets. By comparing the RL schedule to a rule-based reference rather than relying on absolute values, the learning process becomes more stable and easier to interpret. It also reduces the effect of random changes in job arrivals across different runs. In this way, the agent is encouraged to learn scheduling strategies that give priority to urgent jobs and improve on-time performance in a consistent and practical manner.

Setup-Time-Oriented Component

Similarly, the setup-time component compares the agent's job ordering against a setup-minimizing heuristic on the same waiting set. At each decision step, the PPO-WF agent generates a sequence π_{RL} over $\mathcal{W}$, while the Shortest Setup Time (SST) rule produces a baseline sequence π_{SST} by selecting the job with the smallest setup cost from the last processed job type (or initial setup if the machine is idle).

The total setup time required to execute all the jobs of each waiting sequence is simulated, yielding $\mathrm{SetupTime}_{\mathrm{RL}}$ and $\mathrm{SetupTime}_{\mathrm{SST}}$. The setup reward is then:

$$r_{\mathrm{setup}} = \mathrm{SetupTime}_{\mathrm{SST}} - \mathrm{SetupTime}_{\mathrm{RL}} \tag{6.5}$$

A positive value means the RL policy induces less setup overhead than SST for the same set of candidate jobs.

Composite Reward Formulation

The overall reward signal R for the agent is a weighted linear combination of the two components:

$$R = w_1 \cdot r_{\mathrm{tardiness}} + w_2 \cdot r_{\mathrm{setup}} \tag{6.6}$$

where w_1 and w_2 are dynamic weight parameters supplied as goal vectors within the

UVFA framework. These weights define the desired trade-off at each training episode and are sampled from a predefined range to support Pareto-efficient policy learning**.**

This stepwise, same-candidate-set comparison ensures a fair and stable learning signal. Because both the RL policy and the baseline operate on identical waiting job sets under identical system states, the reward reflects a direct, instance-specific performance difference between the learned policy and established heuristics. This design:

- Reduces variance caused by stochastic job arrivals across episodes.
- Provides immediate feedback after every action.
- Encourages the agent to consistently match or exceed the performance of practical, interpretable rules in real time.

6.2.3. UVFA-Based Multi-Objective Reinforcement Learning

Fixed-weight scalarization suffers from key limitations, particularly its inability to adapt to varying objective preferences without retraining—as discussed in **Sections 6.2.1** and **6.2.2**. To address this, this study introduces a UVFA to balance the conflicting objectives of mean tardiness and total setup time in a dynamic flexible job shop environment.

(1) Multi-Objective MDP

Traditional reinforcement learning formulates the scheduling problem as an MDP defined by the tuple $(\boldsymbol{S}, A, P, R, \gamma)$, where:

- $\boldsymbol{S}$ is the set of states,
- A is the set of actions,
- $P(\boldsymbol{s}'|\boldsymbol{s}, a)$ is the transition probability,
- $R(\boldsymbol{s}, a) \in \mathbb{R}$ is a scalar reward,

- $\gamma \in [0,1)$ is the discount factor.

The agent learns a policy $\pi: \boldsymbol{S} \rightarrow A$ that maximizes the expected return:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s_t}, a_t)\right] \tag{6.7}$$

However, in multi-objective scheduling problems, the agent must simultaneously consider multiple conflicting objectives, such as minimizing tardiness, setup time, or energy usage. This introduces a fundamental limitation: a single scalar reward function cannot fully capture the trade-offs among multiple objectives.

To handle this, the MDP is extended into a goal- or weight-conditioned MDP, denoted as:

$$M_w = (\boldsymbol{S}, A, P, \mathbf{R}, \gamma, \mathbf{w}), \tag{6.8}$$

where $\mathbf{R}(\boldsymbol{s}, a) = \left[R^{(1)}(\boldsymbol{s}, a), \ldots, R^{(k)}(\boldsymbol{s}, a)\right] \in \mathbb{R}^k$ is a vector of rewards for k objectives, $\mathbf{w} \in \mathbb{R}^k$ is a weight vector representing the agent's preference or priority over the objectives.

The overall scalar reward at time step t is defined by the inner product:

$$R_w(\boldsymbol{s_t}, a_t) = \mathbf{w}^{\mathrm{T}} \mathbf{R}(\boldsymbol{s_t}, a_t) \tag{6.9}$$

Accordingly, the objective-conditioned return becomes:

$$Q_{\mathbf{w}}(\boldsymbol{s}, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{w}^{\mathrm{T}} \mathbf{R}(\boldsymbol{s_t}, a_t)\right] \tag{6.10}$$

This formulation induces a family of MDPs, one for each possible weight vector $\mathbf{w}$. Each $\mathbf{w}$ leads to a different optimal policy $\pi^*_{\mathbf{w}}$, tailored to the specific trade-off among objectives.

As discussed in **Sections 6.1** and **6.2.1**, directly solving this family of MDPs is infeasible in practice due to the infinite possible combinations of $\mathbf{w}$. Standard approaches often scalarize the objective using fixed weights and train separate agents for different combinations, but this is computationally expensive and lacks generalization to unseen preferences.

(2) Universal Value Function Approximators

To address the challenge, UVFA was proposed by Schaul et al. as a means of generalizing value functions not only over states and actions but also over goals or weight vectors [153]. In the context of multi-objective reinforcement learning, UVFA aims to approximate the entire family of value functions $\{Q^*_{\mathbf{w}}(\boldsymbol{s}, a)\}$ induced by varying objective preferences $\mathbf{w} \in \mathbb{R}^k$, using a single parameterized function:

$$Q(\boldsymbol{s}, a; \mathbf{w}; \theta) \approx Q^*_{\mathbf{w}}(\boldsymbol{s}, a) \tag{6.11}$$

Here, the function is jointly conditioned on the environment state s, the action a, and the weight vector $\mathbf{w}$, with θ representing the shared deep neural network parameters. This formulation enables the agent to generalize across different objective priorities, offering adaptability to dynamically changing preferences, data efficiency through shared learning across objectives, and smooth interpolation between known trade-off solutions.

Schaul et al. [153] explored several architectural designs to implement UVFA, each with different ways of integrating the state and weight (or goal) information:

a) **Concatenated Architecture**

The most straightforward design simply concatenates the state vector $\boldsymbol{s}$, action (if needed), and weight vector $\mathbf{w}$ into a joint input, which is then passed through a multi-layer perceptron (MLP):

$$Q(\boldsymbol{s}, a; \mathbf{w}) = f_\theta([\boldsymbol{s}; \mathbf{w}; a]) \quad (6.12)$$

Figure 6.1a shows the concatenated architecture of UVFA. This design assumes the network can learn the interaction between $\boldsymbol{s}$ and $\mathbf{w}$ from raw concatenation and is easy to implement with minimal structural assumptions.

b) Two-Stream Architecture

A more structured approach splits the input processing into two separate embedding networks as shown in **Figure 6.1b**:

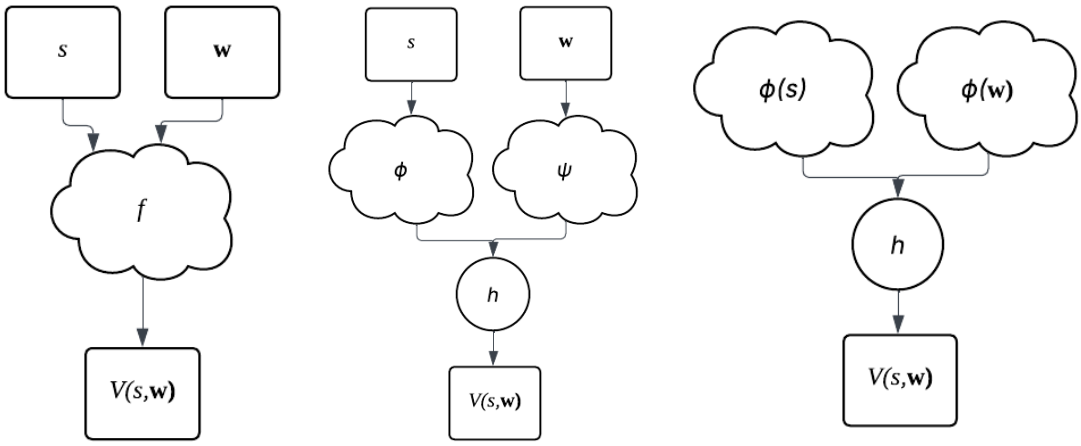
$$Q(\boldsymbol{s}, a; \mathbf{w}) = h(\phi(\boldsymbol{s}, a), \psi(\mathbf{w})) \quad (6.13)$$

where $\phi(\boldsymbol{s}, a)$ maps state-action pairs into a latent representation. $\psi(\mathbf{w})$ maps weight vectors into a latent representation. These two embeddings are combined by a dot product or other function h to produce the final Q-value.

c) Symmetric or Shared Embedding Architecture

In special settings (e.g., when goals and states are from the same domain), the two streams ϕ and ψ can share parameters or even be identical, which enforces additional symmetry and improves data efficiency. For instance,

$$\phi(\boldsymbol{s}) = \psi(\boldsymbol{s}), \qquad Q(\boldsymbol{s}, a; \mathbf{w}) = \phi(\boldsymbol{s})^{\mathrm{T}} \phi(\mathbf{w}) \quad (6.14)$$



(a) (b) (c)

Figure 6.1. Architecture of UVFA. ((a) Concatenated. (b) Two-Stream. (c) Symmetric or Shared Embedding)

In this work, a two-stream UVFA architecture is adopted to approximate the objective-conditioned action-value function $Q(\boldsymbol{s}, a; \mathbf{w})$. The model processes the environment state and the objective weight vector $\mathbf{w}$ through two separate encoding streams and fuses them via concatenation before generating the Q-value output for all candidate actions.

The UVFA network is defined as **Equation (6.11)**, where $\boldsymbol{s} \in \mathbb{R}^d$ represents the environment state vector. $\mathbf{w} \in \mathbb{R}^k$ represents the weight vector. $a \in A$ represents the action corresponding to a dispatching rule. The input tensor $[\boldsymbol{s}, \mathbf{w}] \in \mathbb{R}^{d+k}$ is split internally, ensuring modular encoding and enabling the agent to learn shared structure across both environment dynamics and objective priorities.

(3) DDQN Training with UVFA

The training of the UVFA network in this study follows the DDQN framework. It is extended to handle multi-objective returns via weight conditioning. The target value for Q-learning is computed based on the same objective weight $\mathbf{w}$ used in the current input, ensuring

consistency across training and evaluation.

Algorithm 3: UVFA-DDQN Training Process

1. Initialize: state dim d_s, weight dim d_w, action space A, learning rate α, discount factor γ, exploration rate ε_t, replay buffer size N, batch size B, target update interval T_{update}, objective weight distribution $\mathcal{W}$
2. Initialize online Q-network $Q_\theta(\boldsymbol{s}, a; \mathbf{w})$ and target network $Q_{\theta^-}(\boldsymbol{s}, a; \mathbf{w}) \leftarrow Q_\theta$
3. Initialize empty replay buffer $D \rightarrow \emptyset$
4. **For** each episode = 1 to max_episodes

   Sample initial state $\boldsymbol{s_0}$, sample weight vector $\mathbf{w} \sim \mathcal{W}$

   For each step t = 0,1,…until episode ends:

   With probability ε_t select a random action $a_t \in A$; otherwise choose

   $$a_t = \arg\max_a Q_\theta(\boldsymbol{s_t}, a; \mathbf{w})$$

   Execute a_t, observe next state $\boldsymbol{s_{t+1}}$ and reward factor $r_t \in \mathbb{R}^k$

   Compute scalar reward: $r_t = \mathbf{w}^{\boldsymbol{T}} r_t$

   Store transition $(\boldsymbol{s_t}, a_t, r_t, \boldsymbol{s_{t+1}}; \mathbf{w})$ in buffer D

   If $|D| \geq B$:

   Sample mini-batch $\{(\boldsymbol{s_i}, a_i, r_i, \boldsymbol{s_{i+1}}; \mathbf{w})\}_i^B$ from buffer D

   Compute target:

   $$y_i = r_i + \gamma \cdot Q(\boldsymbol{s_{i+1}}, \arg\max_{a'} Q\,(\boldsymbol{s_{i+1}}, a'; \mathrm{w}_{\boldsymbol{i}}; \theta); \mathrm{w}_{\boldsymbol{i}}; \theta^-)$$

   Update θ to minimize:

   $$\mathcal{L}(\theta) = \frac{1}{B}\sum_{i=1}^{B} [\,Q(\boldsymbol{s_i}, a_i;\ \mathrm{w}_{\boldsymbol{i}}; \theta) - y_i]^2$$

   If t mod $T_{\text{update}} = 0$: updatet target network $\theta^- \leftarrow \theta$

Given a transition tuple $(\boldsymbol{s}, a, r_{\mathbf{w}}, \boldsymbol{s}')$, the training target for the DDQN with UVFA is:

$$y = r_{\mathbf{w}} + \gamma \cdot Q\left(\boldsymbol{s}', \underset{a'}{\operatorname{argmax}}\, Q(\boldsymbol{s}', a'; \mathbf{w}; \theta), \mathbf{w}; \theta^{-}\right), \tag{6.15}$$

where $r_{\mathbf{w}} = \mathbf{w}^{\mathrm{T}}\boldsymbol{r}(\boldsymbol{s}, a)$ is the scalarized reward for the current step. θ is the parameter of the online Q-network**.** θ^{-} is the parameter of the target network**,** updated periodically from θ to improve training stability.

The overall loss function is the mean squared error (MSE) between the predicted Q-value and the target:

$$\mathcal{L}(\theta) = \mathbb{E}_{(\boldsymbol{s}, a, r_{\mathbf{w}}, \boldsymbol{s}') \sim D}[(Q(\boldsymbol{s}, a; \mathbf{w}; \theta) - y)^2], \tag{6.16}$$

where D is the predefined replayed buffer

6.3. Numerical Experiment

Having established the methodology of the proposed UVFA-based multi-objective RL framework in **Section 6.2**, including the agent architecture, objective conditioning, and reward formulation, this section outlines the experimental design used to validate its effectiveness. The dynamic scheduling scenarios are defined with sequence-dependent setup times. This section specifies the simulation environment and describes how varying objective preferences are incorporated through random weight sampling. This experimental setup is designed to evaluate the adaptability, robustness, and Pareto efficiency of the learned policy across different operational conditions.

6.3.1. Scenario Complexity Components

To evaluate the effectiveness and generalizability of the proposed DRL framework,

experimental scenarios are designed that reflect real-world manufacturing complexity. These scenarios incorporate dynamic job arrivals, stochastic machine breakdowns, and sequence-dependent setup times. All scheduling scenarios adhere to the formal definitions established in **Chapter 3**. Furthermore, a multi-objective learning setup is adopted to simulate competing scheduling goals under realistic system dynamics.

(4) Dynamic Job Arrivals

Jobs enter the system dynamically according to a Poisson process, capturing the stochastic nature of production orders. For each job type $\alpha \in A$, the inter-arrival time between two consecutive jobs follows an exponential distribution with mean μ,

$$P(T \leq t) = 1 - e^{-t/\mu} \tag{6.17}$$

where T is the time between two releases of job type α. The random job arrival mechanism reflects the variability in customer demand and order frequency, as previously defined in **Equation (3.1)**. Each arriving job J_i is associated with a predefined sequence of operations $\{O_{i,j}\}_{j=1}^{n_i}$, where n_i is the number of operations for job J_i.

(5) Machine Breakdowns

To simulate resource unreliability, each machine $M_{l,k}$ in work center W_l is subject to stochastic failure and repair events. A machine enters a breakdown state when its cumulative operating time exceeds a randomly sampled Failure Interval $\tau_f \sim \text{Exp}\left(\frac{1}{\text{MTBF}}\right)$, where MTBF is the mean time between failures. Upon failure, the machine becomes unavailable and enters a repair phase lasting for a random duration $\tau_r \sim \text{Exp}\left(\frac{1}{\text{MTTR}}\right)$, where MTTR denotes the mean time to repair.

During breakdown periods, operations cannot be scheduled on the affected machine, and any

ongoing operation is delayed until recovery. This disruption is explicitly modeled in the extended constraints of **Scenario 2** in **Section 3.2**.

(6) Sequence-Dependent Setup Times

In the third scenario, sequence-dependent setup times are introduced to reflect the operational costs of transitioning between jobs of different types on the same machine. The setup time required to switch from job J_i of type α to job $J_{i'}$ of type α' on machine $M_{l,k}$ is given by

$$S_{i,i'}^{l,k} = S_{\alpha,\alpha'}^{l,k} \tag{6.18}$$

where $S_{\alpha,\alpha'}^{l,k}$ denotes the setup time matrix, indexed by job types and machine identifiers. Setup times are non-preemptive and must be completed before the next operation starts processing. This factor is modeled as an additional scheduling constraint and objective component in Scenario 3 in **Section 3.3**.

(7) Objective Weights

As shown in **Section 6.2.2.3**, to represent varying production priorities, each training episode is conditioned on a randomly sampled weight vector $\mathbf{w} = [w_1, w_2] \in \mathbb{R}^2$, for which

$$w_1 + w_2 = 1, \;\; w_1, w_2 \in [0,1], \tag{6.19}$$

where w_1 represents the relative importance assigned to tardiness minimization, and w_2 represents the importance of minimizing setup times. The weight vectors are drawn randomly for each episode to encourage generalization across different objective preferences and enable UVFA.

6.3.2. Environment Settings

To evaluate the performance of the proposed UVFA-based multi-objective reinforcement learning framework, experiments are conducted in a simulated dynamic flexible job shop environment with controlled parameters. The system simulates a multi-work-center production line operating under high variability, including stochastic job arrivals, machine breakdowns, and setup-dependent scheduling costs. This environment replicates the complexity and uncertainty commonly encountered in real-world manufacturing systems. All experiments are conducted using custom-built Python simulation code integrated with PyTorch for DRL. The simulation platform allows precise control over event timing, agent interactions, and reward structures. Training and testing were conducted on a Mac mini equipped with an Apple M4 processor, 10-core CPU, and 10 GB of unified memory. This configuration provides sufficient computational capability for training reinforcement learning agents using PyTorch in a simulated scheduling environment. The environment configuration is summarized in **Table 6.1**.

Table 6.1. Environment settings for UVFA-based scheduling experiments.

Parameter	Value/Description
Number of maximum Work Centers L	5
Number of Parallel Machines per W_l	Uniformly sampled from [1,5]
Number of Job Types	Uniformly sampled from [1,15]
Mean Interval of Job Releasement	Uniformly sampled from [5,15]
Processing Time Range	Uniformly sampled from [1,99]
Routing Type	Scenario 1C
Sequence-Dependent Setup Time	Uniformly sampled from [1,50]
MTBF	Exponentially with a mean value of 1000
MTTR	Exponentially, with a mean value of 50
Job Due Factor	Uniformly sampled from $[1,2] \in \mathbb{R}$
Total Number of Jobs	100 jobs
Episode number	20000 episodes

6.3.3. Deep Neural Network Settings

Table 6.2. Two-stream UVFA network architecture.

Component	Configuration Details
Architecture Type	Two-stream UVFA (state stream + weight stream)
Environment Stream	FC(EnvState, 64) → LeakyReLU(0.01) → FC(64, 128) → LeakyReLU(0.01)
Weight Stream	FC(WeightDim, 16) → LeakyReLU(0.01)
Fusion Mechanism	Concatenation of env and weight embeddings
Post-Fusion Layers	FC(128+16, 128) → LeakyReLU(0.01) → FC(128, ActionSize)
Output Dimension	$Q_\theta(s, a; \mathbf{w})$ for each discrete action (e.g., dispatching rule)
Activation Functions	Leaky ReLU (negative slope = 0.01)
Optimizer	Adam
Learning Rate	3×10^{-5}
Loss Function	Mean Squared Error (MSE)
Target Network Update	Every 100 episodes
Action Selection Strategy	ε -greedy, initial value is 1.0, decay rate is 0.999997, min value is 0.1

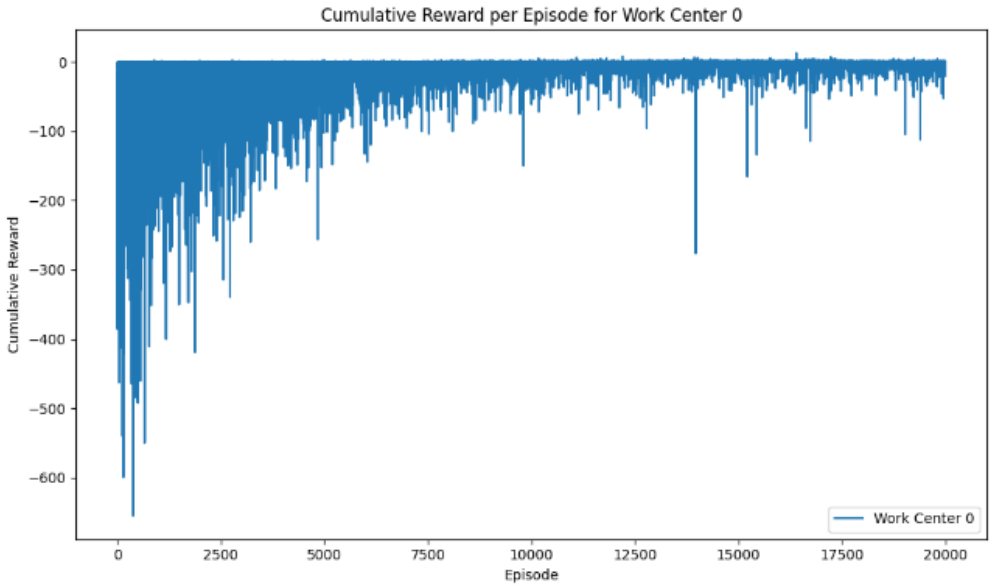
The neural network used in this study follows a two-stream architecture tailored for UVFA. One stream processes environmental state feature (e.g., system status), while the other handles the reward weight vector $\mathbf{w} = [w_1, w_2] \in \mathbb{R}^2$, allowing the agent to generalize value estimation across multiple objective combinations. The architecture is summarized in **Table 6.2**, where FC represents a fully connected layer, where every input neuron is connected to the output neurons.

6.3.4. Training Results

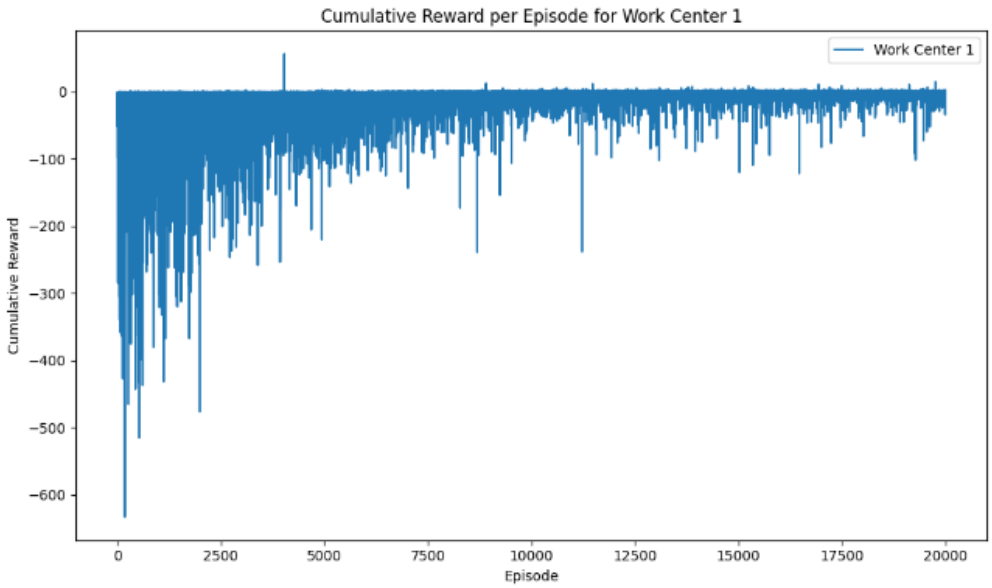
The UVFA-based reinforcement learning agents were trained for 20,000 episodes across all five work centers in the distributed scheduling framework. **Figure 6.2** presents the cumulative reward per episode for each work center (WC0–WC4), providing insights into the learning progression and convergence behavior of the agents. All five subplots exhibit a

common pattern: high variance and low cumulative rewards during the early training phase, gradually transitioning into more stable and higher reward regions as training progresses. This indicates that the agents initially perform random or suboptimal scheduling actions but gradually improve their policy through interaction with the environment.

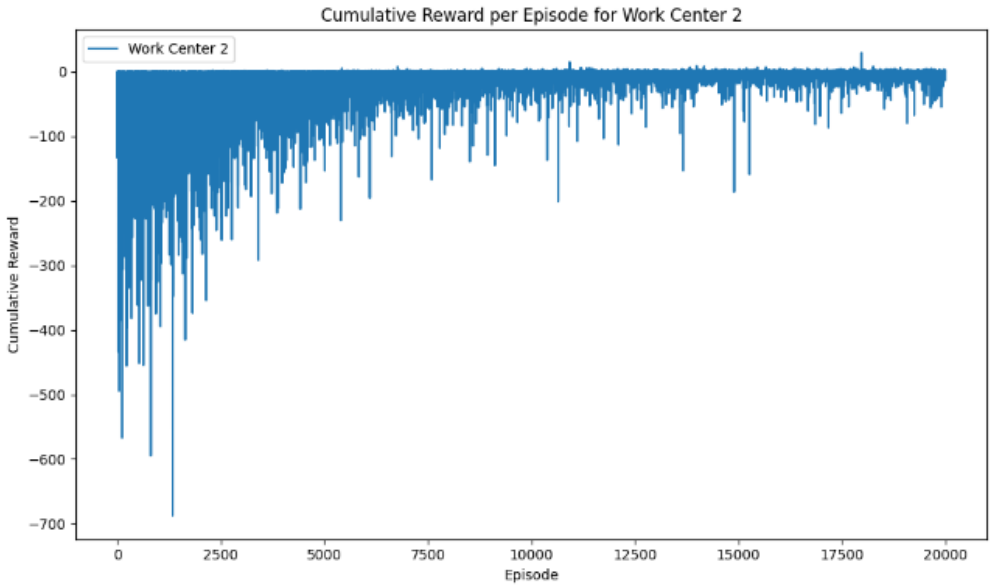
- Work Center 0 (**Figure 6.2a**): The cumulative reward shows a sharp improvement within the first 5,000 episodes, followed by a steady trend toward convergence. Spikes of negative rewards occasionally occur, likely due to extreme scenarios involving high setup penalties or severe job tardiness, but become increasingly rare in later stages.
- Work Center 1 (**Figure 6.2b**): The cumulative reward figure exhibits a similar learning pattern: high reward volatility during early episodes, followed by a steady increase in performance. The curve becomes notably smoother after episode 10,000, suggesting that a stable policy has been learned.
- Work Center 2 (**Figure 6.2c**): While the early learning curve is more volatile than WC0 and WC1, it nonetheless shows consistent improvement over time. The learning dynamics reflect an effective exploration-exploitation balance throughout the training process.
- Work Center 3 (**Figure 6.2d**): This agent demonstrates the fastest stabilization among all five centers. The reward trend plateaus earlier, around episode 8,000, indicating that WC3 may be facing comparatively less variability in job routing or setup conditions.
- Work Center 4 (**Figure 6.2e**): The training behavior closely follows that of WC2, with sharp fluctuations early on but eventual convergence. The amplitude of negative spikes is reduced over time, reinforcing the agent's capability to generalize across dynamic objectives.


Cumulative Reward per Episode for Work Center 0
Cumulative Reward
0
−100
−200
−300
−400
−500
−600
0
2500
5000
7500
10000
12500
15000
17500
20000
Episode
Work Center 0

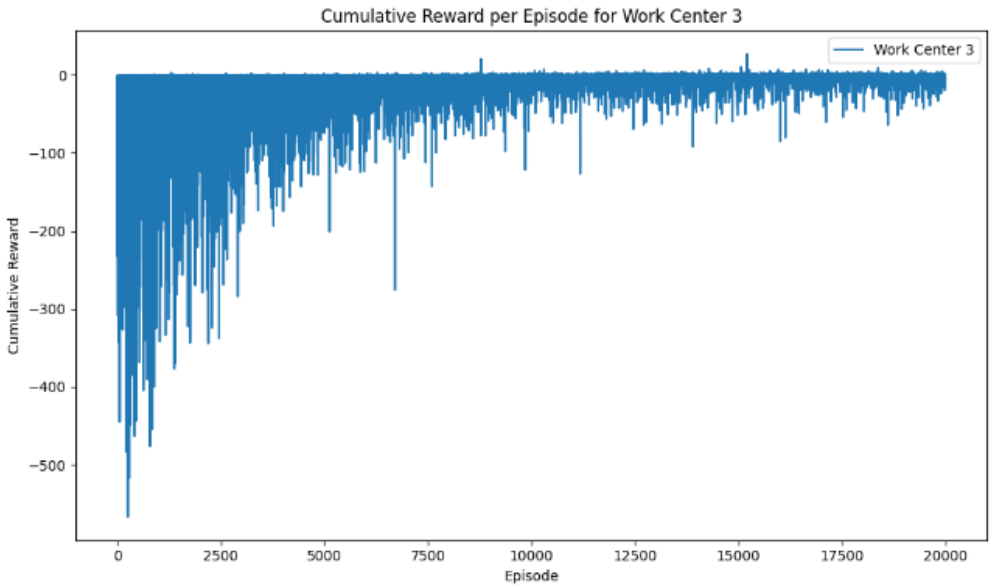
(a)


Cumulative Reward per Episode for Work Center 1
Cumulative Reward
0
−100
−200
−300
−400
−500
−600
0
2500
5000
7500
10000
12500
15000
17500
20000
Episode
Work Center 1

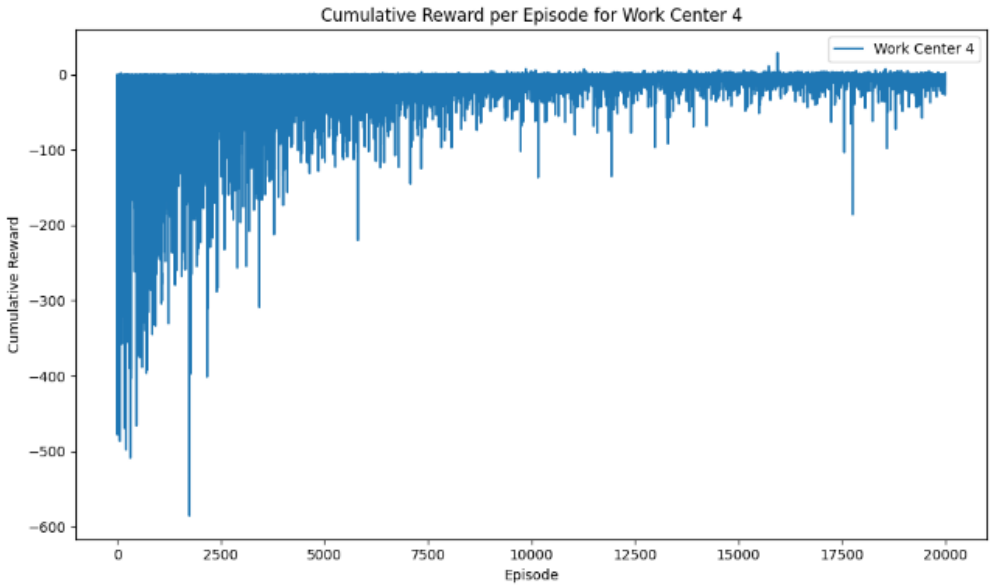
(b)


Cumulative Reward per Episode for Work Center 2
Work Center 2
Cumulative Reward
0
−100
−200
−300
−400
−500
−600
−700
0
2500
5000
7500
10000
12500
15000
17500
20000
Episode

(c)


Cumulative Reward per Episode for Work Center 3
Work Center 3
Cumulative Reward
0
−100
−200
−300
−400
−500
0
2500
5000
7500
10000
12500
15000
17500
20000
Episode

(d)



(e)

Figure 6.2. Cumulative reward per episode for all five work centers.

Despite the inherent stochasticity in job arrivals, machine breakdowns, and sequence-dependent setup times, all UVFA agents successfully learned scheduling policies that consistently increase cumulative reward. The observed convergence trends demonstrate that the two-stream network architecture, in conjunction with reward weight conditioning, enables each agent to robustly adapt to its local work center's scheduling complexity.

6.3.5. Testing Results

This section presents the evaluation results of the trained UVFA-based scheduling agents under dynamic and multi-objective conditions. A total of four distinct test sets were constructed, each reflecting a different reward formulation strategy as described in **Section 6.2.2.3**. These test sets are designed to explore the impact of reward shaping and baseline referencing on the quality and trade-off characteristics of the learned policies. The definition of the four test sets is shown in **Table 6.3**.

Table 6.3. The definition of four test sets based on two reward methods.

Set ID	Tardiness Calculation Method	Reference Sequence for Reward Difference
Set 1	factor $= 10$: time to due $= 10 \cdot (\mathrm{ECT} - d_i)$, if $\mathrm{ECT} \leq d_i$	Compared with the original waiting job sequence
Set 2	time to due $= 0$, if $\mathrm{ECT} \geq d_i$	Compared with the original waiting job sequence
Set 3	factor $= 10$: time to due $= 10 \cdot (\mathrm{ECT} - d_i)$, if $\mathrm{ECT} \leq d_i$	Compared with the benchmark rule (SSAPT for tardiness, SST for setup)
Set 4	time to due $= 0$, if $\mathrm{ECT} \geq d_i$	Compared with the benchmark rule (SSAPT for tardiness, SST for setup)

In each test set, the performance of a fixed trained policy is assessed across 51 sampled objective weight vectors $\mathbf{w} = [w_1, w_2]$, satisfying $w_1 + w_2 = 1$.The values of w_1 range from 0.00 to 1.00 in increments of 0.02. This allows for an approximation of the learned policy's response across the Pareto surface without retraining. The key variations among the sets lie in the penalty structure applied to tardiness and the baseline used for calculating relative improvements in reward signals. In all cases, setup time is considered during evaluation, though its treatment in the reward function may vary based on the reference comparison.

- **Set 1** employs a shaped time-to-due reward (penalty factor = 10) and compares the candidate schedule against the original job queue at the decision point. This represents the standard reward configuration used throughout training.
- **Set 2** uses an unshaped (or tardiness-focused) time-to-due formulation that provides no reward for early completion, only late jobs incur penalties, while still comparing against the original job queue.
- **Set 3** retains the shaped time-to-due reward (factor = 10) but computes both the time-to-due and setup-time differentials relative to fixed heuristic baselines: SSAPT for tardiness and SST for setup. This baseline-referenced design constitutes the key methodological innovation of this work.

- **Set 4** combines the unshaped time-to-due logic of **Set 2** with the benchmark-referenced comparison framework of **Set 3**, serving as a hybrid test case to isolate the effect of reward shaping under heuristic anchoring.

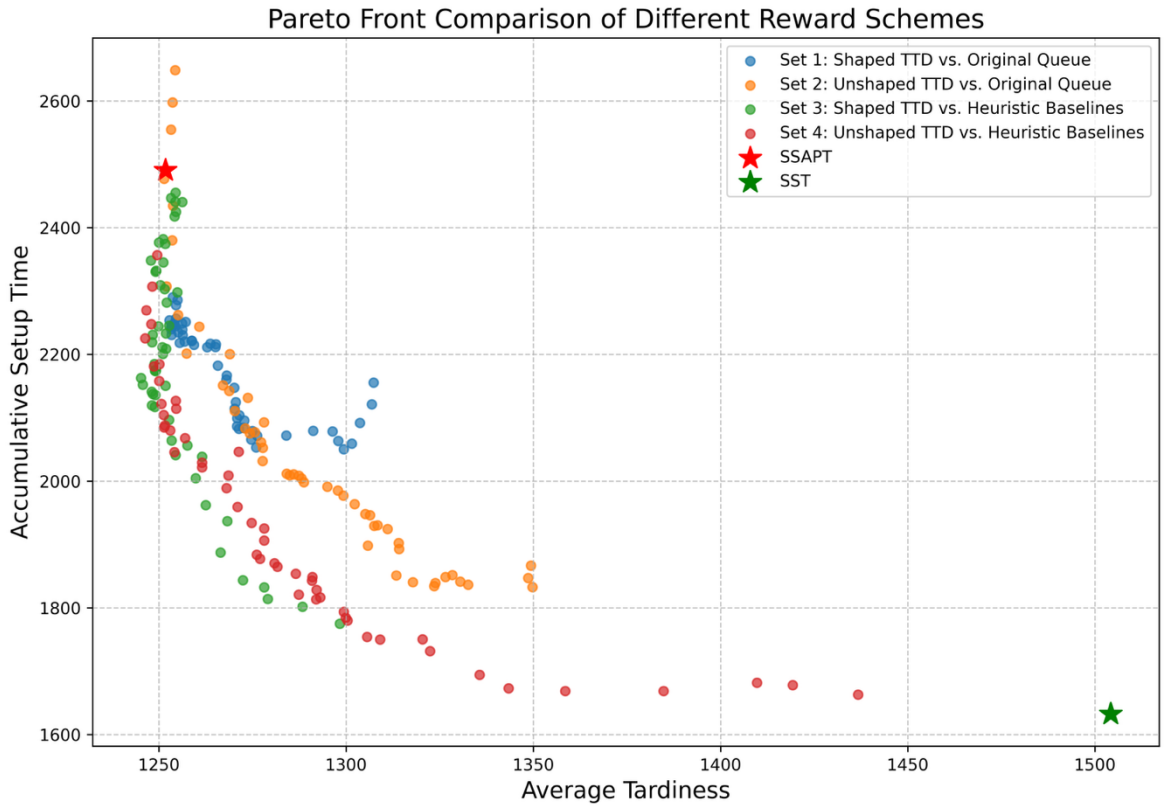


Figure 6.3. The comparison of two objectives, tardiness and setup time, under different reward settings.

The performance of all four test sets is plotted in the tardiness–setup time objective space, as shown in **Figure 6.3**. Each point represents the performance of a policy tested under a specific weight vector. Benchmark dispatching rules (Shortest Setup Time and SSAPT) are plotted as reference stars.

From the visual distribution:

- **Set 1** demonstrates broad coverage across the trade-off space, forming a smooth approximation of the Pareto front. It serves as a balanced benchmark using shaped time-

to-due penalties and queue-based comparisons.

- **Set 2,** which uses an unshaped (tardiness-focused) time-to-due formulation, achieves strong tardiness reduction but incurs higher setup times. This occurs because the absence of early-completion rewards weakens the tardiness signal, shifting the policy's emphasis toward minimizing transitions—a behavior amplified by the original queue comparison.
- **Set 3,** employing both shaped time-to-due penalties and comparisons against SSAPT/SST baselines, shows marked improvements over Set 1, particularly in regions where heuristic policies dominate. This confirms that anchoring rewards to meaningful heuristics enhances policy quality.
- **Set 4** combines the unshaped time-to-due logic with baseline-referenced comparisons. While it underperforms Set 3 in overall balance due to weaker tardiness shaping, it still outperforms queue-based methods, demonstrating that heuristic anchoring can improve scheduling even without explicit early-completion incentives.

These results highlight that both the structure of the reward function and the reference used for computing performance improvements play a pivotal role in shaping learning outcomes. While **Figure 6.3** offers intuitive visual insights, it lacks a formal quantification of solution quality and distribution. Therefore, in the next section, a more rigorous comparison of these test sets is conducted using multi-objective performance metrics, including Generational Distance (GD), Inverted Generational Distance (IGD), and Spread (Δ).

6.3.6. Multi-objective Metric Analysis

To complement the visual and qualitative assessments presented in **Section 6.3.5**, this section provides a quantitative comparison of the four test sets using established multi-objective performance indicators. Specifically, Generational Distance (GD), Inverted Generational Distance (IGD), and Spread (Δ) are computed for each test set. These metrics provide

insights into convergence accuracy, Pareto front coverage, and solution diversity.

Metric Overview

- **Generational Distance (GD):**

  Measures the average distance from each point in the solution set to the nearest point on the true Pareto front. Lower values indicate better convergence.

- **Inverted Generational Distance (IGD):**

  Measures how well the true Pareto front is represented by the solution set. A lower IGD reflects better coverage of the front.

- **Spread (Δ):**

  Assesses the distribution of solutions along the Pareto front. Lower values indicate a more uniform spread, which is critical for representing diverse trade-off solutions.

Table 6.4. The numerical results of four different test sets.

Set	GD	IGD	Spread
Set 1: Original Reward (Factor 10)	54.7875	161.2822	0.7399
Set 2: Original Reward (Tardiness Only)	80.6408	58.4600	0.9658
Set 3: Modified Reward (Factor 10)	81.6450	24.6090	0.7738
Set 4: Modified Reward (Tardiness Only)	22.4185	5.2975	0.7268

Convergence (GD):

Set 4, which combines a tardiness-only reward (factor = ∞) with baseline-referenced comparison, achieves the lowest GD. This suggests that computing performance improvements relative to external baselines (SSAPT/SST), even in a scalarized reward setting, can guide the agent toward high-quality convergence. In contrast, Set 3 is also referenced to the same baseline but exhibits poor performance in GD, implying that the shaped penalty (factor value set to 10) may introduce a conflict with the fixed reference, thereby impairing learning stability. Set 2 (pure tardiness vs. original queue) and Set 1 (full

reward vs. original queue) show similarly high GD, implying that reward comparison to the current job sequence alone may be insufficient for robust convergence.

Coverage (IGD):

Again, Set 4 performs best, demonstrating superior coverage across the approximated Pareto front. Set 3 also improves over the queue-based sets (Set 1 and Set 2), reinforcing the idea that benchmark-referenced evaluation encourages the exploration of more globally effective solutions. The worst coverage is observed in Set 1, suggesting that training reward aligned solely to the local job queue leads to narrow policy development.

Diversity (Spread):

Set 4 achieves the most even distribution of solutions, benefiting from both a sharp reward structure and meaningful reference anchors. Conversely, Set 2, while simple, yields the most unbalanced spread, likely due to converging toward a small set of tardiness-minimizing patterns without considering solution diversity. Set 1 and Set 3 occupy the midrange, but Set 3 again benefits slightly from referencing external baselines.

These updated results further underscore the importance of how reward is defined and what it is compared against:

- **Set 4 (Tardiness-Only + Benchmark-Referenced)** delivers the strongest overall performance across GD, IGD, and Spread. This validates the proposed methodological innovation of computing reward differences relative to fixed heuristic policies (SSAPT for tardiness, SST for setup). Even with a scalar reward structure, this approach leads to robust convergence, high coverage, and well-distributed solutions.
- **Set 3 (Full Reward + Benchmark-Referenced)** shows strong coverage and diversity, but its convergence is less stable, possibly due to complex interactions between the

shaped penalty and fixed reference baseline.

- **Set 1 (Full Reward + Original Queue)** represents the standard training baseline and performs weakest overall in both convergence and coverage, suggesting that purely local comparisons hinder global policy improvement.
- **Set 2 (Tardiness-Only + Original Queue)** offers marginally better convergence than **Set 1**, but suffers from poor diversity, likely due to oversimplification and lack of multi-objective guidance.

6.3.7. SHAP Analysis

1) Setup-Time-Only Reward

Table 6.5. SHAP-based feature importance under setup-only reward weights ($\boldsymbol{w_{tardiness}}$= 0.0, $\boldsymbol{w_{setup}}$= 1.0) in Case 3. Rows = features, columns = work centers (WC0–WC4). Each cell reports mean absolute SHAP value and within-WC rank as "value (rank)".

Feature	WC0	WC1	WC2	WC3	WC4
Potential_Job_Type_Number_in_WC	0.064 (1)	0.295 (2)	0.062 (4)	0.180 (2)	0.178 (2)
Num_Waiting_Jobs_in_WC	0.037 (2)	0.314 (1)	0.384 (1)	0.336 (1)	0.218 (1)
Std_Setup_Time	0.034 (4)	0.103 (8)	0.035 (5)	0.125 (4)	0.108 (4)
Min_Operations	0.034 (3)	0.098 (9)	0.000 (34)	0.000 (32)	0.046 (10)
Mean_Operations	0.033 (6)	0.078 (12)	0.000 (34)	0.000 (32)	0.027 (19)
Mean_Remaining_Operations	0.033 (5)	0.015 (28)	0.009 (18)	0.015 (23)	0.016 (27)
Mean_Processing_Time_in_WC	0.032 (7)	0.059 (14)	0.007 (22)	0.084 (9)	0.024 (21)
Mean_Flow_Time	0.031 (8)	0.084 (10)	0.015 (12)	0.071 (11)	0.043 (12)
Std_Remaining_Processing_Time	0.028 (9)	0.014 (29)	0.005 (25)	0.011 (25)	0.042 (14)

Min_Processing_Time_in_WC	0.027 (10)	0.012 (30)	0.007 (21)	0.002 (31)	0.034 (17)
Min_Flow_Time	0.026 (12)	0.116 (6)	0.015 (13)	0.114 (6)	0.046 (9)
Max_Flow_Time	0.026 (11)	0.152 (4)	0.025 (7)	0.140 (3)	0.060 (6)
Max_Processing_Time_in_WC	0.023 (13)	0.078 (13)	0.074 (3)	0.046 (14)	0.011 (29)
Max_Remaining_Operations	0.021 (14)	0.025 (22)	0.004 (28)	0.000 (32)	0.043 (11)
Mean_Setup_Time	0.019 (15)	0.034 (18)	0.008 (19)	0.051 (13)	0.060 (7)
Max_Setup_Time	0.018 (18)	0.033 (19)	0.008 (20)	0.044 (15)	0.020 (25)
Std_Processing_Time_in_WC	0.018 (17)	0.290 (3)	0.150 (2)	0.095 (7)	0.074 (5)
Std_Operations	0.018 (16)	0.113 (7)	0.000 (34)	0.000 (32)	0.026 (20)
Max_Remaining_Processing_Time	0.017 (19)	0.025 (23)	0.009 (16)	0.023 (20)	0.020 (24)
Mean_Time_to_Due_Date	0.014 (20)	0.028 (21)	0.005 (26)	0.019 (21)	0.047 (8)
Min_Time_to_Due_Date	0.012 (21)	0.029 (20)	0.009 (17)	0.030 (17)	0.010 (31)
Max_Time_to_Due_Date	0.010 (25)	0.047 (16)	0.006 (24)	0.078 (10)	0.028 (18)
Min_Remaining_Processing_Time	0.010 (24)	0.008 (34)	0.016 (11)	0.024 (19)	0.023 (23)
Min_Remaining_Operations	0.010 (23)	0.049 (15)	0.021 (9)	0.038 (16)	0.034 (16)
Std_Remaining_Operations	0.010 (22)	0.009 (33)	0.023 (8)	0.051 (12)	0.042 (13)
Max_Waiting_Time	0.008 (26)	0.150 (5)	0.030 (6)	0.120 (5)	0.024 (22)
Mean_Remaining_Processing_Time	0.007 (27)	0.017 (27)	0.013 (14)	0.026 (18)	0.036 (15)
Mean_Waiting_Time	0.006 (28)	0.079 (11)	0.019 (10)	0.086 (8)	0.012 (28)
Min_Setup_TIme	0.003 (31)	0.018 (25)	0.011 (15)	0.011 (24)	0.122 (3)

Std_Waiting_Time	0.003 (30)	0.017 (26)	0.003 (30)	0.011 (26)	0.006 (34)
Max_Operations	0.003 (29)	0.037 (17)	0.000 (34)	0.000 (32)	0.018 (26)
Max_Earliest_Machine_Available_Time	0.001 (37)	0.003 (36)	0.002 (31)	0.002 (30)	0.002 (37)
Std_Flow_Time	0.001 (36)	0.006 (35)	0.007 (23)	0.005 (28)	0.010 (32)
Std_Time_to_Due_Date	0.001 (35)	0.020 (24)	0.001 (33)	0.005 (29)	0.011 (30)
Mean_Earliest_Machine_Available_Time	0.001 (34)	0.010 (31)	0.004 (29)	0.009 (27)	0.004 (35)
Std_Earliest_Machine_Available_Time	0.001 (33)	0.000 (37)	0.002 (32)	0.000 (32)	0.002 (36)
Min_Earliest_Machine_Available_Time	0.001 (32)	0.009 (32)	0.004 (27)	0.016 (22)	0.006 (33)
Num_Work_Centers	0.000 (38)	0.000 (37)	0.000 (34)	0.000 (32)	0.000 (38)
Num_Machines_in_WC	0.000 (38)	0.000 (37)	0.000 (34)	0.000 (32)	0.000 (38)
Num_Job_Types_in_System	0.000 (38)	0.000 (37)	0.000 (34)	0.000 (32)	0.000 (38)
Min_Waiting_Time	0.000 (38)	0.000 (37)	0.000 (34)	0.000 (32)	0.000 (38)
Weights_of_time_to_due	0.000 (38)	0.000 (37)	0.000 (34)	0.000 (32)	0.000 (38)
Weights_of_Setup_time	0.000 (38)	0.000 (37)	0.000 (34)	0.000 (32)	0.000 (38)

The SHAP analysis under a setup-dominated reward structure reveals how the agent internalizes industrial best practices for changeover reduction. The prominence of *Potential_Job_Type_Number_in_WC* as the top feature in WC0, WC3, and WC4 reflects a learned strategy of job-type clustering. By minimizing the diversity of job types in the immediate queue, the agent reduces the frequency of costly setups. In real-world manufacturing, especially in batch production or job shops, this mirrors the common practice of grouping similar parts together to avoid repeated tooling changes, fixture adjustments, or material swaps.

Moreover, the elevated importance of setup time statistics (*Std_Setup_Time*, *Mean_Setup_Time*, *Min_Setup_Time*) across all work centers indicates that the agent does not merely count setups but actively evaluates their magnitude and variability. High setup time variance signals unpredictability in changeover duration, which can disrupt scheduling stability. Avoiding such transitions, therefore becomes critical, demonstrating adaptive reasoning grounded in operational risk.

One surprising observation is the near-complete disappearance of due-date-related features. Even *Min_Time_to_Due_Date* ranks below 20 in most centers. While expected under a setup-focused objective, the extent of this drop is striking. It suggests the agent is willing to sacrifice short-term due-date adherence to preserve long-term throughput stability through setup reduction. In high-mix, low-volume environments where changeovers dominate cycle time, this trade-off is economically rational—delaying a job by a few hours may be less costly than losing half a shift to reconfiguration.

Conversely, the persistent relevance of *Num_Waiting_Jobs_in_WC* (ranked first in WC1 and second in WC2) is noteworthy. Even when setup is prioritized, raw queue length remains a key signal. This is likely because a long queue increases the opportunity to batch similar jobs. Thus, the agent uses waiting job count not as a tardiness proxy but as a resource for sequencing optimization.

Finally, the low importance of processing time features (for example, *Mean_ Processing_Time_in_WC* ranks below 15 everywhere) confirms that, under this reward scheme, machine utilization takes a back seat to transition efficiency. This challenges the traditional focus on keeping machines busy and instead supports a flow-centric philosophy where smooth, uninterrupted sequences trump individual operation speed.

In summary, the setup-oriented policy does not simply ignore due dates. It redefines efficiency around changeover economics, revealing a sophisticated understanding of

bottleneck dynamics in flexible manufacturing systems. This behavior is not only interpretable but also directly actionable for production engineers seeking to minimize non-value-added time.

2) Tardiness-Only Reward

Table 6.6. SHAP-based feature importance under tardiness-only reward weights ($\boldsymbol{w_{\text{tardiness}}}$=1.0, $\boldsymbol{w_{\text{setup}}}$=0.0) in Case 3. Rows = features, columns = work centers (WC0–WC4). Each cell reports mean absolute SHAP value and within-WC rank as "value (rank)".

Feature	WC0	WC1	WC2	WC3	WC4
Mean_Processing_Time_in_WC	0.109 (1)	0.056 (6)	0.024 (4)	0.118 (3)	0.023 (12)
Std_Processing_Time_in_WC	0.074 (2)	0.073 (4)	0.059 (2)	0.074 (5)	0.025 (11)
Min_Operations	0.058 (3)	0.005 (32)	0.000 (34)	0.000 (32)	0.013 (20)
Num_Waiting_Jobs_in_WC	0.057 (4)	0.391 (1)	0.125 (1)	0.173 (1)	0.124 (1)
Potential_Job_Type_Number_in_WC	0.052 (5)	0.137 (2)	0.017 (5)	0.145 (2)	0.056 (2)
Max_Processing_Time_in_WC	0.035 (6)	0.025 (16)	0.038 (3)	0.061 (6)	0.011 (21)
Max_Remaining_Operations	0.029 (8)	0.006 (30)	0.008 (15)	0.000 (32)	0.026 (9)
Min_Processing_Time_in_WC	0.029 (7)	0.025 (17)	0.007 (18)	0.003 (30)	0.039 (3)
Mean_Setup_Time	0.028 (9)	0.054 (8)	0.008 (14)	0.031 (16)	0.015 (18)
Mean_Operations	0.025 (10)	0.028 (15)	0.000 (34)	0.000 (32)	0.026 (10)
Mean_Flow_Time	0.017 (12)	0.033 (13)	0.006 (21)	0.038 (11)	0.034 (5)
Mean_Remaining_Operations	0.017 (11)	0.008 (28)	0.005 (24)	0.007 (25)	0.020 (13)
Std_Remaining_Processing_Time	0.015 (13)	0.014 (20)	0.007 (17)	0.011 (24)	0.016 (15)

Min_Flow_Time	0.013 (14)	0.030 (14)	0.006 (23)	0.032 (15)	0.030 (7)
Max_Setup_Time	0.012 (15)	0.008 (26)	0.004 (25)	0.017 (18)	0.006 (28)
Max_Flow_Time	0.011 (16)	0.043 (10)	0.015 (7)	0.039 (10)	0.027 (8)
Min_Time_to_Due_Date	0.010 (17)	0.060 (5)	0.011 (10)	0.053 (7)	0.009 (22)
Std_Setup_Time	0.009 (18)	0.056 (7)	0.007 (16)	0.078 (4)	0.039 (4)
Std_Operations	0.008 (20)	0.022 (18)	0.000 (34)	0.000 (32)	0.006 (27)
Mean_Time_to_Due_Date	0.008 (19)	0.013 (22)	0.006 (19)	0.045 (8)	0.015 (17)
Min_Remaining_Operations	0.007 (22)	0.000 (36)	0.012 (9)	0.014 (21)	0.016 (16)
Mean_Remaining_Processing_Time	0.007 (21)	0.008 (27)	0.008 (12)	0.012 (22)	0.008 (24)
Max_Waiting_Time	0.005 (25)	0.078 (3)	0.006 (20)	0.036 (13)	0.008 (23)
Min_Remaining_Processing_Time	0.005 (24)	0.000 (35)	0.014 (8)	0.016 (19)	0.019 (14)
Max_Time_to_Due_Date	0.005 (23)	0.038 (11)	0.009 (11)	0.038 (12)	0.006 (26)
Std_Remaining_Operations	0.004 (26)	0.005 (31)	0.008 (13)	0.036 (14)	0.014 (19)
Mean_Waiting_Time	0.003 (31)	0.037 (12)	0.006 (22)	0.045 (9)	0.008 (25)
Max_Operations	0.003 (30)	0.009 (25)	0.000 (34)	0.000 (32)	0.003 (30)
Std_Waiting_Time	0.003 (29)	0.011 (23)	0.001 (33)	0.014 (20)	0.001 (34)
Mean_Earliest_Machine_Available_Time	0.003 (28)	0.007 (29)	0.002 (29)	0.011 (23)	0.001 (36)
Max_Remaining_Processing_Time	0.003 (27)	0.048 (9)	0.004 (26)	0.003 (27)	0.004 (29)
Max_Earliest_Machine_Available_Time	0.002 (34)	0.002 (34)	0.001 (31)	0.003 (29)	0.000 (37)
Min_Earliest_Machine_Available_Time	0.002 (33)	0.013 (21)	0.003 (27)	0.019 (17)	0.002 (32)

Min_Setup_TIme	0.002 (32)	0.016 (19)	0.017 (6)	0.001 (31)	0.033 (6)
Std_Flow_Time	0.001 (36)	0.002 (33)	0.002 (28)	0.004 (26)	0.001 (33)
Std_Earliest_Machine_Available_Time	0.001 (35)	0.000 (36)	0.001 (30)	0.000 (32)	0.001 (35)
Num_Work_Centers	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Num_Machines_in_WC	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Num_Job_Types_in_System	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Min_Waiting_Time	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Weights_of_time_to_due	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Weights_of_Setup_time	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Std_Time_to_Due_Date	0.000 (37)	0.009 (24)	0.001 (32)	0.003 (28)	0.002 (31)

The SHAP results under a pure tardiness-oriented reward reflect a policy that aligns closely with traditional due-date-driven scheduling logic, while adapting to local workload conditions. In Work Center 0, the high importance of *Mean_Processing_Time_in_WC* and *Std_Processing_Time_in_WC* indicates sensitivity to both average load and processing time variability. In practice, high variability can lead to unpredictable job completions, making it a legitimate concern when minimizing tardiness. Similarly, the emphasis on *Max_Waiting_Time* in Work Centers 1 and 2 suggests a focus on preventing jobs from waiting excessively—an approach consistent with the oldest-job-first heuristic commonly used to avoid severe delays.

An expected but noteworthy observation is the consistently low importance of setup-related features across all work centers. Metrics such as *Std_Setup_Time*, *Mean_Setup_Time*, and *Min_Setup_Time* rank near the bottom, and the feature *Weights_of_Setup_time* is uniformly among the least influential. This is consistent with the reward structure, which assigns zero

weight to setup costs. Consequently, the agent has no incentive to consider changeover effort, even if reducing setups might indirectly support on-time performance by improving machine availability. In real manufacturing settings, this behavior highlights a limitation of single-objective formulations: they may overlook operational interdependencies that affect overall system efficiency.

It is also worth noting that direct due-date features like *Min_Time_to_Due_Date* do not always rank highest. Instead, the agent often relies on proxy indicators such as waiting time or processing load. This suggests that, in dynamic environments with stochastic arrivals and processing times, system-level state variables provide more robust signals for predicting and preventing tardiness than static deadline information alone.

Overall, the analysis confirms that the agent effectively optimizes for the given objective of minimizing tardiness, but does so in a way that is narrowly focused. The results reinforce the value of multi-objective reward design in practical scheduling systems, where balancing timeliness with transition efficiency can lead to more resilient and realistic policies.

3) Balanced Reward

Table 6.7. SHAP-based feature importance under balanced reward weights ($\boldsymbol{w_{\text{tardiness}}}$= 0.5, $\boldsymbol{w_{\text{setup}}}$= 0.5) in Case 3. Rows = features, columns = work centers (WC0–WC4). Each cell reports mean absolute SHAP value and within-WC rank as "value (rank)".

Feature	WC0	WC1	WC2	WC3	WC4
Num_Waiting_Jobs_in_WC	0.632 (1)	0.443 (1)	0.499 (1)	0.260 (1)	0.405 (1)
Potential_Job_Type_Number_in_WC	0.164 (2)	0.267 (2)	0.057 (4)	0.244 (2)	0.186 (2)
Std_Processing_Time_in_WC	0.162 (3)	0.131 (4)	0.247 (2)	0.117 (5)	0.105 (5)
Min_Operations	0.122 (4)	0.004 (33)	0.000 (34)	0.000 (32)	0.026 (20)

Max_Processing_Time_in_WC	0.108 (5)	0.082 (8)	0.144 (3)	0.049 (14)	0.018 (28)
Mean_Processing_Time_in_WC	0.104 (6)	0.094 (6)	0.009 (21)	0.130 (3)	0.026 (21)
Max_Waiting_Time	0.082 (7)	0.180 (3)	0.028 (8)	0.122 (4)	0.062 (12)
Max_Flow_Time	0.067 (8)	0.074 (10)	0.055 (5)	0.091 (7)	0.100 (6)
Min_Flow_Time	0.062 (9)	0.055 (12)	0.033 (7)	0.061 (10)	0.068 (10)
Mean_Remaining_Processing_Time	0.061 (10)	0.011 (29)	0.017 (13)	0.022 (21)	0.080 (8)
Mean_Remaining_Operations	0.058 (11)	0.011 (30)	0.018 (12)	0.012 (24)	0.018 (27)
Std_Operations	0.048 (12)	0.035 (17)	0.000 (34)	0.000 (32)	0.030 (17)
Max_Remaining_Operations	0.042 (13)	0.010 (31)	0.009 (22)	0.000 (32)	0.116 (4)
Mean_Waiting_Time	0.035 (14)	0.079 (9)	0.019 (11)	0.094 (6)	0.028 (19)
Mean_Operations	0.034 (15)	0.047 (15)	0.000 (34)	0.000 (32)	0.039 (14)
Min_Time_to_Due_Date	0.030 (17)	0.051 (13)	0.016 (15)	0.065 (9)	0.021 (22)
Std_Setup_Time	0.030 (16)	0.086 (7)	0.024 (9)	0.081 (8)	0.076 (9)
Mean_Flow_Time	0.027 (18)	0.033 (18)	0.034 (6)	0.027 (17)	0.059 (13)
Min_Earliest_Machine_Available_Time	0.019 (19)	0.020 (22)	0.006 (27)	0.025 (20)	0.010 (33)
Max_Remaining_Processing_Time	0.018 (21)	0.051 (14)	0.009 (23)	0.011 (25)	0.020 (23)
Mean_Earliest_Machine_Available_Time	0.018 (20)	0.013 (28)	0.005 (28)	0.014 (22)	0.006 (35)
Mean_Setup_Time	0.016 (23)	0.100 (5)	0.010 (18)	0.053 (12)	0.085 (7)
Std_Remaining_Processing_Time	0.016 (22)	0.018 (24)	0.007 (24)	0.014 (23)	0.029 (18)
Max_Earliest_Machine_Available_Time	0.013 (25)	0.003 (34)	0.002 (31)	0.003 (31)	0.003 (36)

Min_Remaining_Operations	0.013 (24)	0.000 (36)	0.017 (14)	0.027 (18)	0.019 (24)
Min_Setup_TIme	0.012 (26)	0.029 (19)	0.014 (16)	0.003 (30)	0.172 (3)
Mean_Time_to_Due_Date	0.011 (27)	0.039 (16)	0.006 (26)	0.034 (15)	0.065 (11)
Min_Processing_Time_in_WC	0.010 (29)	0.019 (23)	0.004 (29)	0.008 (27)	0.019 (25)
Std_Remaining_Operations	0.010 (28)	0.014 (27)	0.013 (17)	0.052 (13)	0.038 (16)
Std_Flow_Time	0.008 (30)	0.006 (32)	0.009 (20)	0.007 (28)	0.011 (32)
Max_Time_to_Due_Date	0.007 (31)	0.067 (11)	0.007 (25)	0.054 (11)	0.038 (15)
Max_Setup_Time	0.006 (32)	0.024 (20)	0.010 (19)	0.026 (19)	0.014 (30)
Std_Waiting_Time	0.005 (35)	0.020 (21)	0.001 (33)	0.010 (26)	0.016 (29)
Std_Earliest_Machine_Available_Time	0.005 (34)	0.000 (36)	0.002 (32)	0.000 (32)	0.002 (37)
Min_Remaining_Processing_Time	0.005 (33)	0.000 (35)	0.021 (10)	0.029 (16)	0.009 (34)
Max_Operations	0.004 (36)	0.016 (25)	0.000 (34)	0.000 (32)	0.019 (26)
Std_Time_to_Due_Date	0.003 (37)	0.015 (26)	0.003 (30)	0.005 (29)	0.013 (31)
Num_Work_Centers	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Num_Machines_in_WC	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Num_Job_Types_in_System	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Min_Waiting_Time	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Weights_of_time_to_due	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)
Weights_of_Setup_time	0.000 (38)	0.000 (36)	0.000 (34)	0.000 (32)	0.000 (38)

The SHAP analysis under balanced rewards reveals not only expected trade-offs but also nuanced strategic behaviors that reflect real-world scheduling dilemmas.

First, the persistent dominance of *Num_Waiting_Jobs_in_WC* across all work centers， even under balanced objectives， highlights a fundamental truth in production systems: queue congestion is the most immediate and universal pressure, regardless of whether the goal is to meet due dates or reduce setups. In practice, shop-floor supervisors instinctively prioritize clearing backlogs to prevent cascading delays, and the agent mirrors this operational urgency.

Second, the co-emergence of setup-related features (e.g., *Min_Setup_Time*, *Std_Setup_Time*) and tardiness proxies (e.g., *Mean_Flow_Time*, *Max_Processing_Time_in_WC*) in top ranks, especially in WC1 and WC4, demonstrates the agent's ability to integrate conflicting objectives into a unified decision logic. For instance, choosing a job with slightly longer processing time but much shorter setup can simultaneously reduce machine idle time (improving flow) and avoid costly changeovers. This hybrid reasoning aligns with lean manufacturing principles, where setup reduction directly enables responsiveness.

However, one surprising finding is the relatively low importance of explicit due-date features such as *Min_Time_to_Due_Date* and *Mean_Time_to_Due_Date*, which rank outside the top 15 in most centers. Given that tardiness is half the reward, this seems counterintuitive. Yet, it makes sense in context: flow time and waiting job count are stronger proxies for tardiness in dynamic environments. A long queue or high workload inherently increases the risk of missing future due dates, even without knowing exact deadlines. The agent thus learns to control systemic causes of tardiness rather than reacting to individual due dates—a more robust and scalable strategy.

Conversely, the high importance of *Min_Setup_Time* in WC4 (ranking 3rd with 0.172) is striking. This suggests that in certain work centers, perhaps those with highly heterogeneous job types, the agent prioritizes setup minimization as a primary lever to maintain throughput and effectively treats setup time as a hidden bottleneck. This insight could guide real-world investments: if setup dominates decision-making in a WC, then automation or

standardization of fixtures may yield greater gains than adding machines.

In summary, the balanced policy does not merely "average" two objectives; it discovers synergistic features (like queue length and setup variability) that jointly influence both tardiness and efficiency. This emergent intelligence underscores the value of UVFA's context-aware representation in multi-objective scheduling.

4) Trend Analysis

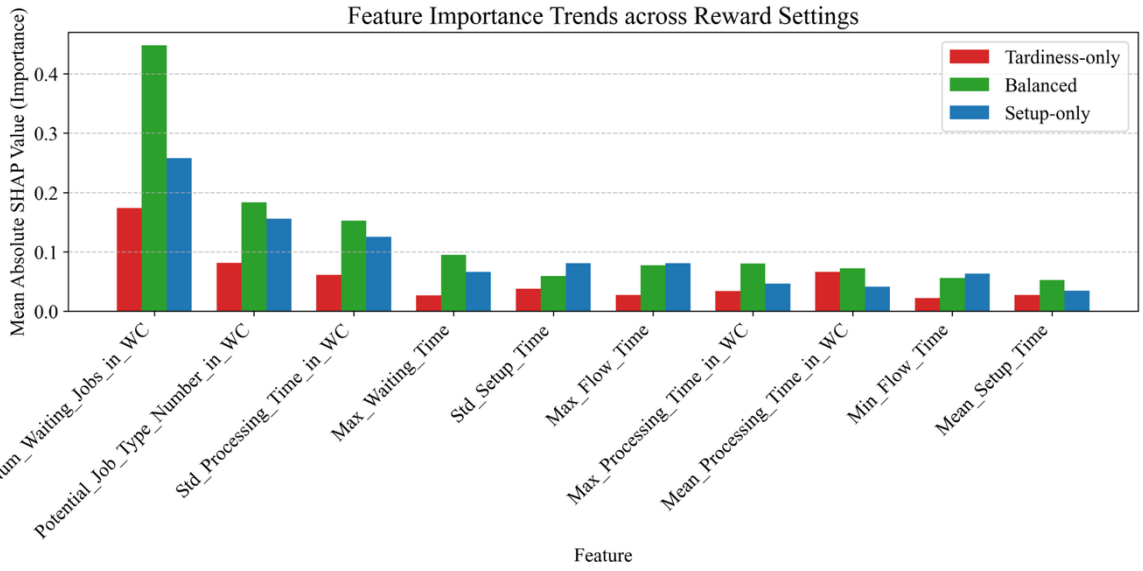


Figure 6.4. Feature importance trends across reward settings.

The SHAP analysis in **Figure 6.4** not only confirms the agent's responsiveness to reward design but also reveals deeper insights into how scheduling decisions align or diverge from real-world operational logic.

In practice, *Num_Waiting_Jobs_in_WC* consistently ranks as the most important feature across all reward settings, making intuitive sense: shop-floor supervisors universally monitor queue length as a primary indicator of congestion and delivery risk. A long queue signals both immediate workload pressure and potential downstream bottlenecks, regardless of

whether the goal is to meet due dates or reduce setups. The agent's reliance on this signal demonstrates its grounding in fundamental production dynamics.

Similarly, the prominence of *Potential_Job_Type_Number_in_WC* under the setup-only objective mirrors established lean manufacturing practices. In high-mix environments, frequent job-type changes incur significant non-value-added time through tooling swaps, calibration, or material handling. By prioritizing sequences that minimize job-type diversity, the agent effectively implements a form of group technology, which is a strategy widely used in industry to reduce changeover costs. This behavior is not only rational but also directly transferable to real scheduling policies.

One notable observation is the relatively modest importance of explicit due-date features such as *Min_Time_to_Due_Date*, even in the tardiness-focused setting. While initially counterintuitive, this reflects a pragmatic adaptation to uncertainty: in dynamic job shops with stochastic arrivals and processing times, rigid adherence to static deadlines can be less effective than managing system-level flow. Instead, the agent leverages proxy indicators like *Max_Waiting_Time* and *Num_Waiting_Jobs_in_WC*, which capture emergent delay risks more robustly than individual due dates. This aligns with modern predictive scheduling approaches that prioritize workload balancing over myopic deadline chasing.

Conversely, the low relevance of processing time statistics (e.g., *Mean_Processing_Time_in_WC*) was somewhat unexpected. Given that longer jobs contribute more to flow time and tardiness, one might anticipate stronger sensitivity to these metrics. However, their moderate SHAP values suggest the agent treats processing time as secondary to queue state. This choice may reflect the environment's structure, where job arrival patterns and sequencing flexibility dominate over individual operation duration.

Perhaps most revealing is what the agent ignores. Under the setup-only policy, due-date features drop to near-zero importance, confirming that single-objective optimization can lead

to strategic blind spots. In reality, completely neglecting delivery performance, even when optimizing for efficiency, could damage customer trust. This underscores a key limitation of narrow reward formulations and supports the adoption of multi-objective or hierarchical rewards in practical applications.

Overall, the SHAP results demonstrate that the UVFA agent does not merely memorize rules but learns context-aware heuristics that echo real-world trade-offs. The alignment between learned feature importance and industrial best practices increases confidence in the model's interpretability and deployability，while also highlighting where human oversight remains essential to balance competing priorities.

6.4. Implications of the Study

The findings of this chapter carry significant implications for both the theoretical advancement of multi-objective reinforcement learning in manufacturing and its practical implementation in dynamic scheduling systems:

- **Generalization across Multiple Objectives:** By integrating UVFA into the DDQN framework, this study presents a generalizable solution that allows a single policy to approximate multiple objective preferences without retraining.
- **Enhanced Decision-Making under Setup Complexity:** The inclusion of sequence-dependent setup times introduces a layer of real-world complexity often overlooked in prior scheduling research. The study demonstrates that a UVFA-enhanced DRL agent can effectively manage trade-offs between conflicting objectives by leveraging extended state representations and setup-aware dispatching rules.
- **Reward Design and Policy Robustness:** Comparative results show that shaping reward functions relative to strong heuristic benchmarks leads to higher-quality Pareto-optimal solutions and improved generalization under dynamic disturbances.

- **Interpretability and Adaptability:** The SHAP analysis reveals that the learned policies dynamically shift their focus according to reward settings, reflecting a genuine understanding of multi-objective trade-offs. Agents prioritize setup-related or tardiness-related features depending on the optimization goal, while maintaining a stable sensitivity to universal indicators like queue pressure.
- **Scalable and Deployable Framework:** The decentralized structure ensures that policies trained with UVFA-DDQN remain scalable and responsive to real-time system events. This makes the approach well-suited for Industry 4.0 production systems that demand high responsiveness, adaptability, and multi-objective optimization capabilities.

6.5. Summary

This chapter presented **Problem 3**, which extends the dynamic scheduling problem into a multi-objective framework incorporating sequence-dependent setup times. To address the challenge of balancing conflicting objectives in real-time, the study introduces a DDQN-based reinforcement learning framework augmented by UVFA. The following key conclusions can be drawn:

- **Effective Multi-Objective Optimization:** The UVFA-enabled DDQN agent successfully learns a spectrum of Pareto-efficient scheduling policies across varying objective weights. Without requiring retraining for different trade-offs, the agent flexibly adapts to changing preferences between minimizing tardiness and minimizing setup time.
- **Performance Superiority across Metrics:** Among four tested reward strategies, the baseline-referenced and scalarized reward structure (**Set 4**) demonstrated the best overall performance in terms of Generational Distance (GD), Inverted GD (IGD), and Spread (Δ). This confirms that comparing performance against strong reference policies enhances learning outcomes and Pareto front coverage.

- **SHAP-Based Interpretability:** Feature importance analysis confirms that the learned agents dynamically reweight scheduling features according to the objective configuration. Setup time–related features dominate when the objective emphasizes setup time minimization, whereas urgency-related features prevail under tardiness minimization. This demonstrates that the UVFA-based framework effectively internalizes objective-aware scheduling behavior.
- **Scalable Reinforcement Learning for Complex Scheduling:** The proposed architecture demonstrates robustness across stochastic job arrivals, machine breakdowns, and setup-sensitive environments. Its decentralized and event-driven nature ensures real-time decision-making with minimal computational overhead, making it suitable for practical deployment in smart manufacturing contexts.

Overall, this chapter confirms that UVFA-enhanced reinforcement learning provides a viable, interpretable, and scalable framework for tackling complex, multi-objective scheduling challenges in Industry 4.0 systems. This sets a foundation for future exploration into goal-conditioned, decentralized DRL in high-dimensional manufacturing optimization problems.

Chapter 7. Conclusion and Future Work

This final chapter consolidates the key findings of the thesis, reflecting on the theoretical and practical contributions made across the three core case studies. It summarizes how the research objectives have been addressed through a progression of increasingly complex scheduling environments and reinforcement learning techniques. Beyond synthesizing the results, the chapter discusses the broader implications of the proposed methods for intelligent manufacturing systems. It also outlines the limitations of the current work and suggests promising directions for future study, particularly in the areas of real-world deployment, multi-objective learning, and adaptive decision-making in dynamic production systems.

7.1. Overview of the Thesis

This thesis investigates the problem of DFJSP in the context of smart manufacturing systems. As production environments become more complex and unpredictable, with random job arrivals, occasional machine failures, and changing scheduling goals, traditional static and rule-based methods often fail to maintain good performance. In response, this research proposes a series of RL-based frameworks designed to enable scalable, real-time, and adaptive scheduling in decentralized, event-driven environments.

The primary objective of the thesis is to explore how distributed RL agents can autonomously learn effective scheduling policies in dynamic production systems. By modeling the scheduling process as an MDP and embedding RL agents within each work center, the proposed frameworks allow the system to continuously respond to real-time events while minimizing key operational objectives such as job tardiness and setup time. The research progresses through three core case studies, each adding layers of complexity to the scheduling environment and learning architecture.

- In **Problem 1**, a baseline RL-based scheduling system using DDQN agents is introduced in a distributed, event-driven architecture. Each work center independently selects scheduling rules in response to stochastic job arrivals.
- In **Problem 2**, the limitations of rule selection are addressed by introducing a continuous prioritization mechanism based on PPO. This enables finer control and more robust responses, especially under disruptions such as machine breakdowns.
- In **Problem 3**, the problem is extended into a multi-objective setting by incorporating sequence-dependent setup times. And UVFA is introduced into the DDQN framework, enabling the agent to generalize across varying trade-offs between conflicting objectives.

Each case builds incrementally upon the previous, collectively contributing to the design of intelligent, interpretable, and resilient scheduling systems suitable for Industry 4.0/5.0 manufacturing contexts. This final chapter synthesizes these findings, highlights the key contributions, and outlines potential directions for future research.

7.2. Comparative Summary

This thesis explores the application of RL in DFJSP through a progressive series of three case studies. Each case increases the complexity of the scheduling environment and strengthens the capacity of the learning model, forming a coherent trajectory from simple rule-based decision-making to generalized multi-objective optimization. The study not only reveals how RL adapts to different scheduling contexts but also reflects the evolution of decision architecture, action space design, and reward modeling.

Problem 1 forms the foundation of this progression. It evaluates the feasibility and adaptability of RL-based rule selection across different levels of routing complexity, under stochastic job arrivals but without resource breakdowns or setup considerations.

In **Case 1A**, the system uses fixed benchmark instances derived from classical job shop

datasets. Each job follows a fixed routing that visits all work centers in a predefined sequence, and processing times are deterministic. This provides a controlled environment to examine whether RL can effectively select dispatching rules in an event-driven, distributed architecture. **Case 1B** introduces variability by allowing stochastic processing times. While all jobs still pass through all work centers in order, the uncertainty in processing time increases the complexity of scheduling decisions. This tests the agent's ability to learn generalizable rule-selection strategies across different episode realizations. **Case 1C** further extends the problem to a flexible routing configuration, where different job types follow different operation sequences and may skip certain work centers altogether. This increases the diversity of job behavior and reduces the symmetry in state transitions, posing greater challenges for rule selection. Across all three sub-cases, the RL agent uses a DDQN framework to select a job priority rule at each decision point. The learning focus is on how to adapt rule selection policies to varying levels of routing and processing complexity within a distributed, event-triggered system.

Problem 2 builds upon the architecture of **Problem 1** by introducing machine breakdowns, which significantly increase uncertainty and require more responsive policies. The RL model shifts from discrete rule selection to continuous prioritization to address this. A PPO agent is used to generate a vector of weights applied to job and machine features. These weighted scores are used to rank candidate job-machine assignments, effectively replacing rule selection with a data-driven scoring mechanism. This continuous control structure allows the agent to fine-tune its decision-making with higher resolution and respond more effectively to real-time disturbances. It also opens the door for feature-based interpretability and a smoother training landscape compared to discrete actions.

In **Problem 3**, the focus shifts from environmental disruptions to multi-objective optimization. Sequence-dependent setup times are added to the system, introducing a natural trade-off between minimizing job tardiness and reducing total setup time. To accommodate

this, the DDQN model is extended with a UVFA, which conditions the value function on an objective weight vector. This architecture allows the agent to generalize across different preferences by learning how the value of actions varies under different reward combinations. The reward function is also decomposed into two components to separately represent tardiness and setup penalties, providing more informative signals during training. **Problem 3** represents the most advanced configuration in the study. In this chapter, the ability of RL to adapt to conflicting goals and support Pareto-efficient scheduling strategies is demonstrated.

Collectively, the three cases illustrate a clear methodological trajectory: from selecting heuristic rules, to learning continuous prioritization, and ultimately to generalized value function approximation for multi-objective decision-making. The models evolve alongside the complexity of the scheduling problem, moving from static environments to those with failures and multiple trade-offs. This progression confirms that RL not only improves performance in each isolated scenario but also provides a scalable framework for intelligent, decentralized scheduling in increasingly complex manufacturing systems.

7.3. Summary of Research Contributions

Through the structured progression of **Problem 1** to **Problem 3**, this thesis makes several methodological, theoretical, and practical contributions to the field of intelligent scheduling in dynamic flexible job shop environments.

(1) A Progressive Reinforcement Learning Framework for Dynamic Scheduling

This thesis proposes a unified but adaptable RL framework that evolves through increasing levels of problem complexity. It begins with rule-based decision-making (DDQN) in **Problem 1**, transitions to feature-weighted continuous prioritization (PPO) in **Problem 2**, and culminates in multi-objective optimization with generalization (DDQN + UVFA) in

Problem 3. This progression reflects how RL architectures can be systematically scaled and adapted to address various types of uncertainty and operational trade-offs in manufacturing. Each learning agent is embedded within a distributed, event-driven environment, which enhances modularity and supports scalable real-time scheduling.

(2) Generalization Across Routing Structures and Uncertainty Types

The modeling strategy in **Case 1A–1C** highlights the flexibility of the proposed architecture in adapting to increasingly generalized job routing scenarios:

- **Case 1A** uses deterministic job shop instances with fixed routing and processing times.
- **Case 1B** introduces stochastic processing durations.
- **Case 1C** removes routing uniformity, allowing job types to skip certain work centers.
- **Problem 2** adds dynamic machine failures to the environment, modeled through MTBF and MTTR, creating time-varying machine availability that forces the agent to consider equipment reliability in scheduling decisions.
- **Problem 3** introduces sequence-dependent setup times and shifts the problem from single-objective to multi-objective optimization. This requires the model to explicitly capture operation sequences, job types, and setup transition patterns, which significantly increases temporal and combinatorial complexity.

Despite these increasingly generalized and dynamic configurations, the proposed RL framework remains structurally stable and computationally scalable. It does not require fundamental changes to the decision-making architecture when moving from one scenario to the next. This modularity in modeling design supports adaptability across different industries, production lines, and scheduling priorities.

(3) Transition from Discrete Rule Selection to Continuous Priority Learning

One of the key innovations is the transformation of the action space from a limited set of

dispatching rules to a high-dimensional feature-weighted prioritization function. In **Problem 2**, a PPO agent learns continuous weights over job and machine features, allowing more fine-grained control and policy expressions. This shift improves responsiveness to localized disruptions (e.g., machine breakdowns) while also supporting better generalization across episodes. The feature-weighted prioritization strategy offers a new hybrid between interpretable scheduling logic and learning-based adaptability.

(4) Multi-Objective Scheduling through UVFA-Conditioned Value Learning

In **Problem 3**, the thesis addresses the challenge of conflicting objectives, specifically, tardiness versus setup time, by using a UVFA. By conditioning the value function on an objective preference vector $\mathbf{w}$, the agent can learn a generalized policy capable of operating under varying reward configurations. This allows a single agent to generate Pareto-approximating scheduling policies that adapt to different business priorities without retraining. The agent's reward signal is explicitly decomposed into objective-specific terms, improving training feedback and interpretability.

(5) Interpretability through SHAP-Based Feature Attribution

To enhance transparency and build trust in learned scheduling policies, this thesis employs SHAP-based feature attribution to interpret the PPO-WF agent's local decision logic. Specifically, it examines how state features influence the selection of the next dispatching action at each decision epoch. Using a background dataset of 2000 representative states sampled from the converged policy's replay buffer, the analysis reveals consistent patterns across diverse environments. In static settings (**Cases 1A–1C**), "*Num_Machines_in_WC*" is consistently the most influential feature, outweighing queue length and due-date information, suggesting the agent prioritizes structural capacity over transient congestion to maintain system fluidity and reduce tardiness. In dynamic, failure-prone environments (**Case 2**), machine health indicators such as time since last repair gain importance, reflecting risk-

aware behavior aligned with predictive maintenance. Under multi-objective rewards (**Case 3**), the agent treats "*Num_Waiting_Jobs_in_WC*" as a universal signal of operational pressure while selectively elevating setup-related features like "*Potential_Job_Type_Number_in_WC*" when changeover costs matter. Global system attributes show negligible impact across all cases, confirming that high performance can be achieved with purely local observability, which enhances scalability and robustness. Although input features are statistically dependent, this is a known limitation of standard SHAP. Despite this, the consistency of rankings across work centers, cases, and training runs, together with their alignment with domain knowledge, supports these attributions as practical approximations of the agent's reasoning. This framework enables production engineers to audit specific decisions, verify behavioral plausibility, and identify blind spots such as the underweighting of due dates in setup-dominated regimes, making SHAP a critical bridge between reinforcement learning and human-supervised, safety-conscious manufacturing operations.

7.4. Limitations and Future Directions

While this study proposes a scalable and interpretable reinforcement learning framework for dynamic flexible job shop scheduling, several challenges remain, offering opportunities for further enhancement:

(1) Bridging Simulation and Real-World Deployment

All experiments were conducted in simulation environments, which—despite modeling realistic factors such as job arrivals, routing variability, and machine breakdowns—cannot fully capture the unpredictability, latency, and operational noise of actual factories. Future work should explore integration with manufacturing execution systems (MES) [30], [154] or digital twins, enabling physical validation under real-time sensor data, hardware constraints, and communication delays.

(2) Reducing Computational Cost and Accelerating Learning

Training PPO and UVFA-based agents requires substantial computational resources and long training times to achieve stable convergence, potentially limiting deployment in resource-constrained settings. Research into transfer learning [26], curriculum learning [27], policy distillation [155], or meta-learning could significantly shorten training while enabling rapid adaptation to new production scenarios with minimal retraining [31], [156].

(3) Expanding and Dynamically Adjusting Objectives

The current model focuses on tardiness and sequence-dependent setup time, omitting other important manufacturing objectives such as energy efficiency, labor cost, machine wear, or carbon footprint. Future extensions should broaden the objective set and incorporate mechanisms for dynamically adjusting objective weights in real time, enabling adaptive scheduling strategies in response to changing business priorities.

(4) Enhancing Decentralized Coordination and Interpretability

Agents operate independently within work centers, lacking explicit coordination. While this supports modularity, it may lead to suboptimal global performance. Cooperative multi-agent reinforcement learning and communication protocols could improve workload balancing and bottleneck anticipation. Additionally, interpretability—currently based on local SHAP analysis—could be enhanced through trajectory-level explanations, causal reasoning, or attention-based visualization to better capture policy evolution and build user trust.

(5) Improving Robustness and Online Adaptation

The offline-trained agents cannot adapt during deployment, making them susceptible to performance degradation under evolving production conditions. Incorporating online or continual learning mechanisms could enable ongoing policy updates, maintaining

performance under concept drift. Furthermore, robustness should be extended beyond job arrivals and machine breakdowns to cover setup time variability, job cancellations, operator delays, and data anomalies.

List of Publication

[1] C. Ngwu, Y. Liu, and R. Wu, “Reinforcement learning in dynamic job shop scheduling: a comprehensive review of AI-driven approaches in modern manufacturing,” *Journal of Intelligent Manufacturing*, 2025, doi: 10.1007/s10845-025-02585-6.

References

[1] Y. Lu, “Industry 4.0: A survey on technologies, applications and open research issues,” *Journal of Industrial Information Integration*, vol. 6, pp. 1–10, Jun. 2017, doi: 10.1016/j.jii.2017.04.005.

[2] IoT Analytics, “Industry 4.0 and smart manufacturing,” *IoT Analytics*, Nov. 14, 2018. [Online]. Available: https://iot-analytics.com/industry-4-0-and-smart-manufacturing/. [Accessed: Aug. 15, 2025].

[3] S. Nahavandi, “Industry 5.0-a human-centric solution,” *Sustainability*, vol. 11, no. 16, p. 4371, Aug. 2019, doi: 10.3390/su11164371.

[4] O. Sribnyak, “Mini Cooper: Current Marketing Strategy, Digital Marketing Approach, the Brand & Ethical Values,” *Bulletin - Prague College Centre for Research and Interdisciplinary Studies*, vol. 2012, no. 2, Aug. 2012, doi: 10.2478/v10284-012-0001-3.

[5] R. Cigolini, S. Franceschetto, and A. Sianesi, “Shop floor control in the VLSI circuit manufacturing: a simulation approach and a case study,” *International Journal of Production Research*, vol. 60, no. 18, pp. 5450–5467, 2022, doi: 10.1080/00207543.2021.1959954.

[6] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 5th ed. New York, NY, USA: Springer, 2016.

[7] M. L. Pinedo, *Planning and Scheduling in Manufacturing and Services*, 2nd ed. New

York, NY, USA: Springer, 2009.

[8] S. Ikli, C. Mancel, M. Mongeau, X. Olive, and E. Rachelson, “The aircraft runway scheduling problem: A survey,” *Computers and Operations Research*, vol. 132. 2021, doi: 10.1016/j.cor.2021.105336.

[9] P. Yuan, W. Han, X. Su, J. Liu, and J. Song, “A dynamic scheduling method for carrier aircraft support operation under uncertain conditions based on rolling horizon strategy,” *Applied Sciences*, vol. 8, no. 9, 2018, doi: 10.3390/app8091546.

[10] J. A. Bennell, M. Mesgarpour, and C. N. Potts, “Dynamic scheduling of aircraft landings,” *European Journal of Operational Research*, vol. 258, no. 1, pp. 315–327, 2017, doi: 10.1016/j.ejor.2016.08.015.

[11] C. F. Daganzo, “The crane scheduling problem,” *Transportation Research Part B*, vol. 23, no. 3, 1989, doi: 10.1016/0191-2615(89)90001-5.

[12] K. H. Kim and Y. M. Park, “A crane scheduling method for port container terminals,” *European Journal of Operational Research*, vol. 156, no. 3, 2004, doi: 10.1016/S0377-2217(03)00133-4.

[13] F. Yao, A. Demers, and S. Shenker, “Scheduling model for reduced CPU energy,” in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, Milwaukee, WI, USA, 1995, pp. 374–382, doi: 10.1109/sfcs.1995.492493.

[14] A. Lieder and R. Stolletz, “Scheduling aircraft take-offs and landings on interdependent and heterogeneous runways,” *Transportation Research Part E: Logistics and Transportation Review*, vol. 88, 2016, doi: 10.1016/j.tre.2016.01.015.

[15] D. Ronen, “Cargo ships routing and scheduling: Survey of models and problems,” *European Journal of Operational Research*, vol. 12, no. 2. 1983, doi: 10.1016/0377-2217(83)90215-1.

[16] L. Zhang, L. Wang, Z. Li, C. Liu, and Y. Wang, “Deep reinforcement learning for dynamic flexible job shop scheduling problem considering variable processing times,” *Journal of Manufacturing Systems*, vol. 71, 2023, doi: 10.1016/j.jmsy.2023.09.009.

[17] G. E. Vieira, J. W. Herrmann, and E. Lin, “Rescheduling manufacturing systems: A framework of strategies, policies, and methods,” *Journal of Scheduling*, 2003, doi: 10.1023/A:1022235519958.

[18] D. Ouelhadj and S. Petrovic, “A survey of dynamic scheduling in manufacturing systems,” *Journal of Scheduling*, vol. 12, no. 4. pp. 417–431, Aug. 2009, doi: 10.1007/s10951-008-0090-8.

[19] I. Sabuncuoglu and M. Bayiz, “Analysis of reactive scheduling problems in a job shop environment,” *European Journal of Operational Research*, vol. 126, no. 3, 2000, doi: 10.1016/S0377-2217(99)00311-2.

[20] M. Zandieh and M. Gholami, “An immune algorithm for scheduling a hybrid flow shop with sequence-dependent setup times and machines with random breakdowns,” *International Journal of Production Research*, vol. 47, no. 24, pp. 6999–7027, 2009, doi: 10.1080/00207540802400636.

[21] Y. N. Sotskov and F. Werner, “Sequence-dependent setup and clean-up times in a two-machine job-shop with minimizing makespan,” in *Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing (INCOM)*, Saint-

Etienne, France, 2006, pp. 197–202, doi: 10.3182/20060517-3-fr-2903.00034.

[22] H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, "Executing production schedules in the face of uncertainties: A review and some future directions," *European Journal of Operational Research*, 2005, vol. 161, no. 1, doi: 10.1016/j.ejor.2003.08.027.

[23] W. Shen, D. H. Norrie, and J.-P. A. Barthès, *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*. London, U.K.: Taylor & Francis, 2001.

[24] J. Kletti, "Integrated Industry: MES enables decentralisation," *Productivity Management*, vol. 20, no. 2, pp. 34–37, 2015.

[25] N. Aissani, A. Bekrar, D. Trentesaux, and B. Beldjilali, "Dynamic scheduling for multi-site companies: A decisional approach based on reinforcement multi-agent learning," *Journal of Intelligent Manufacturing*, vol. 23, no. 6, pp. 2513–2529, Dec. 2012, doi: 10.1007/s10845-011-0580-y.

[26] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10. 2010, doi: 10.1109/TKDE.2009.191.

[27] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9. 2022, doi: 10.1109/TPAMI.2021.3069908.

[28] R. Liu, R. Piplani, and C. Toro, "Deep reinforcement learning for dynamic scheduling of a flexible job shop," *International Journal of Production Research*, vol. 60, no. 13,

pp. 4049–4069, 2022, doi: 10.1080/00207543.2022.2058432.

[29] S. Luo, L. Zhang, and Y. Fan, “Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning,” *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 4, pp. 3020–3038, Oct. 2022, doi: 10.1109/TASE.2021.3104716.

[30] A. Shojaeinasab, Y. Zhang, and J. Wang, “Intelligent manufacturing execution systems: A systematic review,” *Journal of Manufacturing Systems*, vol. 62. 2022, doi: 10.1016/j.jmsy.2022.01.004.

[31] S. Tian, L. Li, W. Li, H. Ran, X. Ning, and P. Tiwari, “A survey on few-shot class-incremental learning,” *Neural Networks*, vol. 169. 2024, doi: 10.1016/j.neunet.2023.10.039.

[32] G. Dantzig, R. Fulkerson, and S. Johnson, “Solution of a large-scale traveling-salesman problem,” *Journal of the Operations Research Society of America*, vol. 2, no. 4, 1954, doi: 10.1287/opre.2.4.393.

[33] H. Xiong, S. Shi, D. Ren, and J. Hu, “A survey of job shop scheduling problem: The types and models,” *Computers and Operations Research*, vol. 142. Elsevier Ltd, Jun. 01, 2022, doi: 10.1016/j.cor.2022.105731.

[34] J. Shahrabi, M. A. Adibi, and M. Mahootchi, “A reinforcement learning approach to parameter estimation in dynamic job shop scheduling,” *Computers and Industrial Engineering*, vol. 110, pp. 75–82, 2017, doi: 10.1016/j.cie.2017.05.026.

[35] Y. Gu, M. Chen, and L. Wang, “A self-learning discrete salp swarm algorithm based

on deep reinforcement learning for dynamic job shop scheduling problem," *Applied Intelligence*, 2023, doi: 10.1007/s10489-023-04479-7.

[36] N. E. D. A. Said, Y. Samaha, E. Azab, L. A. Shihata, and M. Mashaly, "An online reinforcement learning approach for solving the dynamic flexible job-shop scheduling problem for multiple products and constraints," in *Proceedings of the 2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, 2021, pp. 134–139, doi: 10.1109/CSCI54926.2021.00095.

[37] H. Kim, D. E. Lim, and S. Lee, "Deep learning-based dynamic scheduling for semiconductor manufacturing with high uncertainty of automated material handling system capability," *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 1, pp. 13–22, Feb. 2020, doi: 10.1109/TSM.2020.2965293.

[38] J. Chang, D. Yu, Y. Hu, W. He, and H. Yu, "Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival," *Processes*, vol. 10, no. 4, Apr. 2022, doi: 10.3390/pr10040760.

[39] C. D. Hubbs, C. Li, N. V. Sahinidis, I. E. Grossmann, and J. M. Wassick, "A deep reinforcement learning approach for chemical production scheduling," *Computers and Chemical Engineering*, vol. 141, Oct. 2020, doi: 10.1016/j.compchemeng.2020.106982.

[40] H. Zhu, M. Li, Y. Tang, and Y. Sun, "A deep-reinforcement-learning-based optimization approach for real-time scheduling in cloud manufacturing," *IEEE Access*, vol. 8, pp. 9987–9997, 2020, doi: 10.1109/ACCESS.2020.2964955.

[41] S. Qu, J. Wang, and J. Jasperneite, “Dynamic scheduling in large-scale stochastic processing networks for demand-driven manufacturing using distributed reinforcement learning,” in *Proceedings of the 23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Turin, Italy, Sep. 2018, pp. 433–440,doi: 10.1109/ETFA.2018.8502508.

[42] M. Zandieh and M. A. Adibi, “Dynamic job shop scheduling using variable neighbourhood search,” *International Journal of Production Research*, 2010, doi: 10.1080/00207540802662896.

[43] F. Liu, S. Wang, Y. Hong, and X. Yue, “On the robust and stable flowshop scheduling under stochastic and dynamic disruptions,” *IEEE Transactions on Engineering Management*, vol. 64, no. 4, pp. 539–553, 2017, doi: 10.1109/TEM.2017.2712611.

[44] C. Duan, C. Deng, and B. Wang, “Optimal maintenance policy incorporating system level and unit level for mechanical systems,” *International Journal of Systems Science*, vol. 49, no. 5, pp. 1074–1087, 2018, doi: 10.1080/00207721.2018.1432782.

[45] R. Laggoune, A. Chateauneuf, and D. Aissani, “Preventive maintenance scheduling for a multi-component system with non-negligible replacement time,” *International Journal of Systems Science*, vol. 41, 2010, doi: 10.1080/00207720903230765.

[46] M. H. Shu, L. Y. Hsu, and B. M. Hsu, “Dynamic performance and cost measures with setup determination under state-age-dependent deterioration,” *International Journal of Systems Science*, vol. 41, no. 9, pp. 1121–1131, 2010, doi: 10.1080/00207720903244113.

[47] X. Zhang and S. van de Velde, “On-line two-machine open shop scheduling with time

lags," *European Journal of Operational Research*, vol. 204, no. 1, pp. 14–19, 2010, doi: 10.1016/j.ejor.2009.09.023.

[48] M. Hopf, C. Thielen, and O. Wendt, "Competitive algorithms for multistage online scheduling," *European Journal of Operational Research*, vol. 260, no. 2, pp. 468–481, 2017, doi: 10.1016/j.ejor.2016.12.047.

[49] S. C. Horng and S. S. Lin, "Ordinal optimization based metaheuristic algorithm for optimal inventory policy of assemble-to-order systems," *Applied Mathematical Modelling*, vol. 42, pp. 43–57, 2017, doi: 10.1016/j.apm.2016.10.002.

[50] R. Ramezanian and M. Saidi-Mehrabad, "Hybrid simulated annealing and MIP-based heuristics for stochastic lot-sizing and scheduling problem in capacitated multi-stage production system," *Applied Mathematical Modelling*, vol. 37, no. 7, pp. 5134–5147, 2013, doi: 10.1016/j.apm.2012.10.024.

[51] N. Mebarki and A. Shahzad, "Correlation among tardiness-based measures for scheduling using priority dispatching rules," *International Journal of Production Research*, 2013, doi: 10.1080/00207543.2012.762131.

[52] M. Thürer, M. Stevenson, T. Qu, and M. Godinho Filho, "The design of simple subcontracting rules for make-to-order shops: An assessment by simulation," *European Journal of Operational Research*, 2014, doi: 10.1016/j.ejor.2014.06.018.

[53] R. Germs and N. D. Van Foreest, "Order acceptance and scheduling policies for a make-to-order environment with family-dependent lead and batch setup times," *International Journal of Production Research*, 2013, doi: 10.1080/00207543.2012.693638.

[54] S. Lee and Y. Yih, “Reducing patient-flow delays in surgical suites through determining start-times of surgical cases,” *European Journal of Operational Research*, vol. 238, no. 2, pp. 620–629, 2014, doi: 10.1016/j.ejor.2014.03.043.

[55] S. A. Ali and M. A. El-Baz, “A fuzzy logic based algorithm for non-permutation flow shop scheduling with static and dynamic environment,” in *Proceedings of the 47th International Conference on Computers and Industrial Engineering (CIE)*, Lisbon, Portugal, Oct. 2017, pp. 1–6, doi: 10.1109/ICCIE.2017.8167785.

[56] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, “Optimization and approximation in deterministic sequencing and scheduling: A survey,” in *Discrete Optimization II*, P. L. Hammer, E. L. Johnson, and B. H. Korte, Eds. Amsterdam, The Netherlands: North-Holland, 1979, pp. 287–326.

[57] K. R. Baker, “Sequencing rules and due-date assignments in a job shop,” *Management Science*, vol. 30, no. 9, pp. 1093–1104, Sep. 1984, doi: 10.1287/mnsc.30.9.1093.

[58] E. Balas, “Duality in discrete programming: II. the quadratic case,” *Management Science*, vol. 16, no. 1, 1969, doi: 10.1287/mnsc.16.1.14.

[59] L. Meng, Y. Ren, B. Zhang, J. Q. Li, H. Sang, and C. Zhang, “MILP modeling and optimization of energy-efficient distributed flexible job shop scheduling problem,” *IEEE Access*, vol. 8, pp. 191191–191203, 2020, doi: 10.1109/ACCESS. 2020.3032548.

[60] A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier, “Staff scheduling and rostering: A review of applications, methods and models,” *European Journal of Operational*

Research, 2004, vol. 153, no. 1, doi: 10.1016/S0377-2217(03)00095-X.

[61] P. Baptiste, C. Le Pape, and W. Nuijten, *Constraint-based scheduling : applying constraint programming to scheduling problems*. Boston, MA, USA: Springer, 2001.

[62] P. D. D. Dominic, S. Kaliyamoorthy, and M. S. Kumar, "Efficient dispatching rules for dynamic job shop scheduling," *International Journal of Advanced Manufacturing Technology*, vol. 24, no. 1–2, pp. 70–75, 2004, doi: 10.1007/s00170-002-1534-5.

[63] J. H. Holland, "Genetic algorithms," *Scientific American*, vol. 267, no. 1, pp. 66–73, Jul. 1992, doi: 10.2307/24939139.

[64] M. Gen and L. Lin, "Multiobjective evolutionary algorithm for manufacturing scheduling problems: State-of-the-art survey," *Journal of Intelligent Manufacturing*, 2014, vol. 25, no. 5, pp. 849–866, doi: 10.1007/s10845-013-0804-4.

[65] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: a survey with a unified framework," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 41–66, Mar. 2017, doi: 10.1007/s40747-017-0036-x.

[66] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983, doi: 10.1126/science.220.4598.671.

[67] F. Garza-Santisteban, L. Cruz-Reyes, N. R. Pérez, and H. Terashima-Marín, "A simulated annealing hyper-heuristic for job shop scheduling problems," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC)*, Wellington, New Zealand, Jun. 2019, pp. 1560–1567, doi:

10.1109/CEC.2019.8790296.

[68] G. Mohammadi, A. Karampourhaghghi, and F. Samaei, “A multi-objective optimisation model to integrating flexible process planning and scheduling based on hybrid multi-objective simulated annealing,” *International Journal of Production Research*, vol. 50, no. 18, pp. 5063–5076, 2012, doi: 10.1080/00207543.2011.631602.

[69] F. Glover and M. Laguna, *Tabu Search*. Boston, MA, USA: Kluwer Academic Publishers, 1997.

[70] S. Mahmud, A. Abbasi, R. K. Chakrabortty, and M. J. Ryan, “Multi-operator communication based differential evolution with sequential Tabu Search approach for job shop scheduling problems,” *Applied Soft Computing*, vol. 108, 2021, doi: 10.1016/j.asoc.2021.107470.

[71] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: Optimization by a colony of cooperating agents,” *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol. 26, no. 1, pp. 29–41, Feb. 1996, doi: 10.1109/3477.484436.

[72] M. F. Uslu, S. Uslu, and F. Bulut, “An adaptive hybrid approach: Combining genetic algorithm and ant colony optimization for integrated process planning and scheduling,” *Applied Computing and Informatics*, vol. 18, no. 1–2, 2022, doi: 10.1016/j.aci.2018.12.002.

[73] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, Perth, WA, Australia, Nov./Dec. 1995, pp. 1942–1948, doi: 10.1109/ICNN.1995.488968.

[74] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, “A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems,” *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4. Institute of Electrical and Electronics Engineers Inc., pp. 904–916, Jul. 01, 2019, doi: 10.1109/JAS.2019.1911540.

[75] Z. Lian, “A united search particle swarm optimization algorithm for multiobjective scheduling problem,” *Applied Mathematical Modelling*, vol. 34, no. 11, pp. 3518–3526, 2010, doi: 10.1016/j.apm.2010.03.001.

[76] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez et al., “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017, doi: 10.1038/nature24270.

[77] J. Kober, J. A. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *International Journal of Robotics Research*, vol. 32, no. 11, 2013, doi: 10.1177/0278364913495721.

[78] Y. Wei, X. Jiang, P. Hao, and K. Gu, “Pattern driven dynamic scheduling approach using reinforcement learning,” in *Proceedings of the 2009 IEEE International Conference on Automation and Logistics, ICAL 2009*, 2009, pp. 514–519, doi: 10.1109/ICAL.2009.5262867.

[79] M. E. Aydin and E. Öztemel, “Dynamic job-shop scheduling using reinforcement learning agents,” *Robotics and Autonomous Systems*, vol. 33, no. 2–3, pp. 169–178, Nov. 2000, doi: 10.1016/S0921-8890(00)00087-3.

[80] J. Wang, S. Qu, J. Wang, J. O. Leckie, and R. Xu, “Real-time decision support with

reinforcement learning for dynamic flowshop scheduling," in *Proceedings of the Smart SysTech 2017 – European Conference on Smart Objects, Systems and Technologies*, Dortmund, Germany, Jun. 2017, pp. 1–9.

[81] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," arXiv:1312.5602 [cs.LG], Dec. 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[82] L. Hu, Z. Liu, W. Hu, Y. Wang, J. Tan, and F. Wu, "Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network," *Journal of Manufacturing Systems*, vol. 55, pp. 1–14, Apr. 2020, doi: 10.1016/j.jmsy.2020.02.004.

[83] S. Yang, J. Wang, L. Xin, and Z. Xu, "Real-time and concurrent optimization of scheduling and reconfiguration for dynamic reconfigurable flow shop using deep reinforcement learning," *CIRP Journal of Manufacturing Science and Technology*, vol. 40, pp. 243–252, Jan. 2023, doi: 10.1016/j.cirpj.2022.12.001.

[84] S. Yang, J. Wang, and Z. Xu, "Real-time scheduling for distributed permutation flowshops with dynamic job arrivals using deep reinforcement learning," *Advanced Engineering Informatics*, vol. 54, Oct. 2022, doi: 10.1016/j.aei.2022.101776.

[85] Y. Zhang, H. Zhu, D. Tang, T. Zhou, and Y. Gui, "Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems," *Robotics and Computer-Integrated Manufacturing*, vol. 78, Dec. 2022, doi: 10.1016/j.rcim.2022.102412.

[86] X. Lin and J. Chen, "Deep reinforcement learning for dynamic scheduling of two-

stage assembly flowshop," in *Proceedings of the 27th International Conference on Neural Information Processing (ICONIP)*, Bangkok, Thailand, Nov. 2020, published in Lecture Notes in Computer Science, vol. 12690, pp. 263–271, 2021, doi: 10.1007/978-3-030-78811-7_25.

[87] J. Liu, F. Qiao, M. Zou, J. Zinn, Y. Ma, and B. Vogel-Heuser, "Dynamic scheduling for semiconductor manufacturing systems with uncertainties using convolutional neural networks and reinforcement learning," *Complex and Intelligent Systems*, vol. 8, no. 6, pp. 4641–4662, Dec. 2022, doi: 10.1007/s40747-022-00844-0.

[88] X. Jing, X. Yao, M. Liu, and J. Zhou, "Multi-agent reinforcement learning based on graph convolutional network for flexible job shop scheduling," *Journal of Intelligent Manufacturing*, vol. 35, no. 1, pp. 75–93, Jan. 2024, doi: 10.1007/s10845-022-02037-5.

[89] C. L. Liu, C. C. Chang, and C. J. Tseng, "Actor-critic deep reinforcement learning for solving job shop scheduling problems," *IEEE Access*, vol. 8, pp. 71752–71762, 2020, doi: 10.1109/ACCESS.2020.2987820.

[90] T. Zhou, D. Tang, H. Zhu, and Z. Zhang, "Multi-agent reinforcement learning for online scheduling in smart factories," *Robotics and Computer-Integrated Manufacturing*, vol. 72, Dec. 2021, doi: 10.1016/j.rcim.2021.102202.

[91] B. Waschneck, A. Reichstaller, L. Belzner, and T. Bauernhansl, "Optimization of global production scheduling with deep reinforcement learning," in *Proceedings of the 51st CIRP Conference on Manufacturing Systems (CIRP CMS)*, Stockholm, Sweden, May 2018, published in *Procedia CIRP*, vol. 72, pp. 1264–1269, 2018, doi: 10.1016/j.procir.2018.03.212.

[92] M. Caramia and P. Dell'Olmo, *Multi-objective Management in Freight Logistics: Increasing Capacity, Service Level, Sustainability, and Safety with Optimization Algorithms*. Cham, Switzerland: Springer, 2020.

[93] J. Long, Z. Zheng, and X. Gao, "Dynamic scheduling in steelmaking-continuous casting production for continuous caster breakdown," *International Journal of Production Research*, vol. 55, no. 11, pp. 3197–3216, 2017, doi: 10.1080/00207543.2016.1268277.

[94] J. Chongwatpol and R. Sharda, "RFID-enabled track and traceability in job-shop scheduling environment," *European Journal of Operational Research*, vol. 228, no. 2, pp. 374–386, Jul. 2013, doi: 10.1016/j.ejor.2013.01.009.

[95] J. Fan and D. Feng, "Design of cellular manufacturing system with quasi-dynamic dual resource using multi-objective GA," *International Journal of Production Research*, vol. 51, no. 14, pp. 4134–4154, 2013, doi: 10.1080/00207543.2012.748228.

[96] A. Rossi, A. Pandolfi, and M. Lanzetta, "Dynamic set-up rules for hybrid flow shop scheduling with parallel batching machines," *International Journal of Production Research*, vol. 52, no. 13, pp. 3842–3857, 2014, doi: 10.1080/00207543.2013.835496.

[97] V. Sels, N. Gheysen, and M. Vanhoucke, "A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions," *International Journal of Production Research*, vol. 50, no. 15, pp. 4255–4270, 2012, doi: 10.1080/00207543.2011.611539.

[98] A. Baykasoğlu and F. S. Karaslan, "Solving comprehensive dynamic job shop scheduling problem by using a GRASP-based approach," *International Journal of*

Production Research, vol. 55, no. 11, pp. 3308–3325, 2017, doi: 10.1080/00207543.2017.1306134.

[99] M. Rabbani, M. Monshi, and H. Rafiei, "A new AATP model with considering supply chain lead-times and resources and scheduling of the orders in flowshop production systems: A graph-theoretic view," *Applied Mathematical Modelling*, vol. 38, no. 24, pp. 6098–6107, 2014, doi: 10.1016/j.apm.2014.05.011.

[100] I. Sabuncuoglu and O. B. Kizilisik, "Reactive scheduling in a dynamic and stochastic FMS environment," *International Journal of Production Research*, vol. 41, no. 17, pp. 4211–4231, 2003, doi: 10.1080/0020754031000149202.

[101] B. Shnits, "Methods for activating the decision-making process that is used for controlling flexible manufacturing systems," *International Journal of Production Research*, 2010, doi: 10.1080/00207540903436703.

[102] J. B. C. Tajan, A. I. Sivakumar, and S. B. Gershwin, "Heuristic control of multiple batch processors with incompatible job families and future job arrivals," *International Journal of Production Research*, vol. 50, no. 15, pp. 4206–4219, 2012, doi: 10.1080/00207543.2011.601342.

[103] V. Vinod and R. Sridharan, "Simulation-based metamodels for scheduling a dynamic job shop with sequence-dependent setup times," *International Journal of Production Research*, 2009, doi: 10.1080/00207540701486082.

[104] K. Wang, H. Qin, Y. Huang, M. Luo, and L. Zhou, "Surgery scheduling in outpatient procedure centre with re-entrant patient flow and fuzzy service times," *Omega*, vol. 102, 2021, doi: 10.1016/j.omega.2020.102350.

[105] M. Ebrahimi, S. M. T. Fatemi Ghomi, and B. Karimi, “Hybrid flow shop scheduling with sequence dependent family setup time and uncertain due dates,” *Applied Mathematical Modelling*, vol. 38, no. 9–10, pp. 2490–2504, 2014, doi: 10.1016/j.apm.2013.10.061.

[106] J. B. Wang, C. J. Hsu, and D. L. Yang, “Single-machine scheduling with effects of exponential learning and general deterioration,” *Applied Mathematical Modelling*, vol. 37, no. 4, pp. 2293–2299, 2013, doi: 10.1016/j.apm.2012.05.022.

[107] B. C. Choi, K. Lee, J. Y. T. Leung, and M. L. Pinedo, “Flow shops with machine maintenance: Ordered and proportionate cases,” *European Journal of Operational Research*, vol. 207, no. 1, pp. 97–104, 2010, doi: 10.1016/j.ejor.2010.04.018.

[108] K. Rafiee, M. Rabbani, H. Rafiei, and A. Rahimi-Vahed, “A new approach towards integrated cell formation and inventory lot sizing in an unreliable cellular manufacturing system,” *Applied Mathematical Modelling*, vol. 35, no. 4, pp. 1810–1819, 2011, doi: 10.1016/j.apm.2010.10.011.

[109] P. Georgiadis and C. Michaloudis, “Real-time production planning and control system for job-shop manufacturing: A system dynamics analysis,” *European Journal of Operational Research*, vol. 216, no. 1, pp. 94–104, 2012, doi: 10.1016/j.ejor.2011.07.022.

[110] W. Weng and S. Fujimura, “Control methods for dynamic time-based manufacturing under customized product lead times,” *European Journal of Operational Research*, vol. 218, no. 1, pp. 86–96, 2012, doi: 10.1016/j.ejor.2011.10.014.

[111] M. Sheikhalishahi, N. Eskandari, A. Mashayekhi, and A. Azadeh, “Multi-objective

open shop scheduling by considering human error and preventive maintenance," *Applied Mathematical Modelling*, vol. 67, pp. 573–587, 2019, doi: 10.1016/j.apm.2018.11.015.

[112] S. Pradhan, P. Damodaran, and K. Srihari, "Predicting performance measures for Markovian type of manufacturing systems with product failures," *European Journal of Operational Research*, vol. 184, no. 2, pp. 725–744, 2008, doi: 10.1016/j.ejor.2006.11.016.

[113] T. N. Wong, C. W. Leung, K. L. Mak, and R. Y. K. Fung, "Integrated process planning and scheduling/rescheduling - An agent-based approach," *International Journal of Production Research*, vol. 44, no. 18–19, pp. 3627–3655, 2006, doi: 10.1080/00207540600675801.

[114] Y. Li, W. Gu, M. Yuan, and Y. Tang, "Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources using hybrid deep Q-network," *Robotics and Computer-Integrated Manufacturing*, vol. 74, Apr. 2022, doi: 10.1016/j.rcim.2021.102283.

[115] T. Zhou, D. Tang, H. Zhu, and L. Wang, "Reinforcement learning with composite rewards for production scheduling in a smart factory," *IEEE Access*, vol. 9, pp. 752–766, 2021, doi: 10.1109/ACCESS.2020.3046784.

[116] S. Luo, L. Zhang, and Y. Fan, "Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning," *Computers & Industrial Engineering*, vol. 159, p. 107489, Sep. 2021, doi: 10.1016/j.cie.2021.107489.

[117] T. Hickling, A. Zenati, N. Aouf, and P. Spencer, "Explainability in deep

reinforcement learning: A review into current methods and applications," *ACM Computing Surveys*, vol. 56, no. 5, 2023, doi: 10.1145/3623377.

[118] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, Long Beach, CA, USA, Dec. 2017, pp. 4768–4777.

[119] C. J. Ejiyi, O. A. Adeola, M. O. Ogunleye, and O. S. Adewumi, "A robust predictive diagnosis model for diabetes mellitus using Shapley-incorporated machine learning algorithms," *Healthcare Analytics*, vol. 3, 2023, doi: 10.1016/j.health.2023.100166.

[120] L. O. Hjelkrem and P. E. de Lange, "Explaining deep learning models for credit scoring with shap: A case study using open banking data," *Journal of Risk and Financial Management*, vol. 16, no. 4, 2023, doi: 10.3390/jrfm16040221.

[121] S. Bhattacharjee, Y. B. Hwang, K. Ikromjanov, R. I. Sumon, H. C. Kim, and H. K. Choi, "An explainable computer vision in histopathology: Techniques for interpreting black box model," in *Proceedings of the 4th International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Fukuoka, Japan, Feb. 2022, pp. 188–193, doi: 10.1109/ICAIIC54071.2022.9722656.

[122] K. Aas, M. Jullum, and A. Løland, "Explaining individual predictions when features are dependent: More accurate approximations to Shapley values," *Artificial Intelligence*, vol. 298, 2021, doi: 10.1016/j.artint.2021.103502.

[123] R. Liessner, J. Dohmen, and M. Wiering, "Explainable reinforcement learning for longitudinal control," in *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART)*, Vienna, Austria, Feb. 2021, pp. 874–881, doi:

10.5220/0010256208740881.

[124] C. V. S. R. Syavasya and A. L. Muddana, “Optimization of autonomous vehicle speed control mechanisms using hybrid DDPG-SHAP-DRL-stochastic algorithm,” *Advances in Engineering Software*, vol. 173, 2022, doi: 10.1016/j.advengsoft.2022.103245.

[125] A. K. Raz, J. M. Dotterweich, J. D. Musgrave, and T. H. Chung, “Test and evaluation of reinforcement learning via robustness testing and explainable AI for high-speed aerospace vehicles,” in *Proceedings of the 2022 IEEE Aerospace Conference*, Big Sky, MT, USA, Mar. 2022, pp. 1–11, doi: 10.1109/AERO53065.2022.9843563.

[126] X. Dai, S. Cheng, and A. Chong, “Deciphering optimal mixed-mode ventilation in the tropics using reinforcement learning with explainable artificial intelligence,” *Energy and Buildings*, vol. 278, 2023, doi: 10.1016/j.enbuild.2022.112629.

[127] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere, “Explainable reinforcement learning through a causal lens,” in *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI)*, New York, NY, USA, Feb. 2020, pp. 2493–2500, doi: 10.1609/aaai.v34i03.5631.

[128] D. Janzing, L. Minorics, and P. Blöbaum, “Feature relevance quantification in explainable AI: A causal problem,” in *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS)*, Online, Aug. 2020, pp. 2907–2916, doi: 10.48550/arXiv.1910.13413.

[129] C. Ngwu, Y. Liu, and R. Wu, “Reinforcement learning in dynamic job shop scheduling: a comprehensive review of AI-driven approaches in modern

manufacturing," *Journal of Intelligent Manufacturing*, 2025, doi: 10.1007/s10845-025-02585-6.

[130] L. Wang, Z. Zhang, D. Zhang, Y. Qiao, and X. Liu, "Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning," *Computer Networks*, vol. 190, May 2021, doi: 10.1016/j.comnet.2021.107969.

[131] P. Tassel, M. Gebser, and K. Schekotihin, "A reinforcement learning environment for job-shop scheduling," *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 12, 2021.

[132] T. Zhou, Y. Wang, H. Li, and J. Zhang, "Reinforcement learning for online optimization of job-shop scheduling in a smart manufacturing factory," *Advances in Mechanical Engineering*, vol. 14, no. 3, Mar. 2022, doi: 10.1177/16878132221086120.

[133] F. A. M. do Nascimento and F. Hessel, "A decentralized federated learning architecture for intrusion detection in IoT systems," in *Proceedings of the 14th International Conference on Network and Service Management (CNSM)*, Izmir, Turkey, Oct. 2021, pp. 265–273, published in *Lecture Notes in Networks and Systems*, vol. 450, 2022, doi: 10.1007/978-3-030-99587-4_22.

[134] J. C. Palacio, Y. M. Jiménez, L. Schietgat, B. Van Doninck, and A. Nowé, "A Q-learning algorithm for flexible job shop scheduling in a real-world manufacturing scenario," in *Proceedings of the 54th CIRP Conference on Manufacturing Systems (CIRP CMS)*, Budapest, Hungary, Jun. 2021, pp. 1279–1284, published in *Procedia CIRP*, vol. 106, 2022, doi: 10.1016/j.procir.2022.02.183.

[135] L. Wang, Z. Pan, and J. Wang, “A review of reinforcement learning based intelligent optimization for manufacturing scheduling,” *Complex System Modeling and Simulation*, vol. 1, no. 4, pp. 257–270, Dec. 2021, doi: 10.23919/CSMS.2021.0027.

[136] E. Gerstl, B. Mor, and G. Mosheiov, “Scheduling on a proportionate flowshop to minimise total late work,” *International Journal of Production Research*, vol. 57, no. 2, pp. 531–543, 2019, doi: 10.1080/00207543.2018.1456693.

[137] G. L. Ragatz and V. A. Mabert, “A simulation analysis of due date assignment rules,” *Journal of Operations Management*, vol. 5, no. 1, 1984, doi: 10.1016/0272-6963(84)90005-6.

[138] Y. Pochet and L. A. Wolsey, “Production planning by mixed integer programming,” New York, NY, USA: Springer, 2006.

[139] E. Taillard, “Benchmarks for basic scheduling problems,” *European Journal of Operational Research*, 1993, doi: 10.1016/0377-2217(93)90182-M.

[140] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-Learning,” in *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*, Phoenix, AZ, USA, Feb. 2016, pp. 2094–2100, doi: 10.1609/aaai.v30i1.10295.

[141] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, Chia Laguna, Sardinia, Italy, May 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html

[142] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” In *Proceedings of the ICML 2013 Workshop on Deep Learning for Audio, Speech and Language Processing*, Atlanta, GA, USA, Jun. 2013. [Online]. Available: https://ai.stanford.edu/~amaas/papers/relu_icml2013_final.pdf

[143] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, May 2015. [Online]. Available: https://arxiv.org/abs/1412.6980

[144] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan et al., “PyTorch: An imperative style, high-performance deep learning library,” in *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS)*, Vancouver, BC, Canada, Dec. 2019, pp. 8024–8035.

[145] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, “Trust region policy optimization,” in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, Lille, France, Jul. 2015, pp. 1889–1897.

[146] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” in *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, San Juan, PR, USA, May 2016. [Online]. Available: https://arxiv.org/abs/1506.02438

[147] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *Proceedings of the 13th Annual Conference on Neural Information Processing Systems (NeurIPS)*, Denver, CO, USA, Nov./Dec. 1999, pp. 1057–1063, published in *Adv. Neural Inf. Process.*

Syst., vol. 12, 2000.

[148] R. C. Wilson and C. H. White, "Sequence dependent set-up times and job sequencing," *International Journal of Production Research*, vol. 15, no. 2, 1977, doi: 10.1080/00207547708943117.

[149] S. Tri Windras Mara, E. Sutoyo, R. Norcahyo, and A. Pratama Rifai, "The job sequencing and tool switching problem with sequence-dependent set-up time," *Journal of King Saud University - Engineering Sciences*, vol. 35, no. 1, 2023, doi: 10.1016/j.jksues.2021.02.015.

[150] J. Chang, D. Yu, Z. Zhou, W. He, and L. Zhang, "Hierarchical reinforcement learning for multi-objective real-time flexible scheduling in a smart shop floor," *Machines*, vol. 10, no. 12, 2022, doi: 10.3390/machines10121195.

[151] Z. Hajiakhondi-Meybodi, A. Mohammadi, and J. Abouei, "Deep reinforcement learning for trustworthy and time-varying connection scheduling in a coupled UAV-based femtocaching architecture," *IEEE Access*, vol. 9, 2021, doi: 10.1109/ACCESS.2021.3060323.

[152] J. Leng, Y. Zhang, D. Wang, M. Liu, and X. Chen, "A multi-objective reinforcement learning approach for resequencing scheduling problems in automotive manufacturing systems," *International Journal of Production Research*, vol. 61, no. 15, 2023, doi: 10.1080/00207543.2022.2098871.

[153] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal value function approximators," in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, Lille, France, Jul. 2015, pp. 1312–1320.

[154] S. Mantravadi and C. Møller, “An overview of next-generation manufacturing execution systems: How important is MES for industry 4.0?,” in *Proceedings of the 18th International Conference on Metal Forming (Metal Forming)*, Copenhagen, Denmark, Sep. 2018, pp. 597–604, published in *Procedia Manufacturing*, vol. 30, 2019, doi: 10.1016/j.promfg.2019.02.083.

[155] A. A. Rusu, S. G. Colmenarejo, C. Rothörl, and R. Pascanu, “Policy distillation,” in *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, San Juan, PR, USA, May 2016. [Online]. Available: https://arxiv.org/abs/1511.06295

[156] Y. Feng, J. Chen, J. Xie, T. Zhang, H. Lv, and T. Pan, “Meta-learning as a promising approach for few-shot cross-domain fault diagnosis: Algorithms, applications, and prospects,” *Knowledge-Based Systems*, vol. 235, 2022, doi: 10.1016/j.knosys.2021.107646.