Proceedings of the 2025 Winter Simulation Conference
E. Azar, A. Djanatliev, A. Harper, C. Kogler, V. Ramamohan, A. Anagnostou, and S. J. E. Taylor, eds.

A REINFORCEMENT LEARNING-BASED DISCRETE EVENT SIMULATION APPROACH FOR STREAMLINING JOB-SHOP PRODUCTION LINE UNDER UNCERTAINTY

Jia-Min Chen[1], Bimal Nepal[1], and Amarnath Banerjee[1]

[1]Dept. of Industrial and Systems Eng., Texas A&M University, College Station, TX, USA

ABSTRACT

Streamlining the order release strategy for a job-shop production system under uncertainty is a complex problem. The system is likely to have a number of stochastic parameters contributing to the problem complexity. These factors make it challenging to develop optimal job-shop schedules. This paper presents a Reinforcement Learning-based discrete-event simulation approach that streamlines the policy for releasing orders in a job-shop production line under uncertainty. A digital twin (DT) was developed to simulate the job-shop production line, which facilitated the collection of process and equipment data. A reinforcement learning algorithm is connected to the DT environment and trains with the previously collected data. Once the training is complete, its solution is evaluated in the DT using experimental runs. The method is compared with a few popular heuristic-based rules. The experimental results show that the proposed method is effective in streamlining the order release in a job-shop production system with uncertainty.

1 INTRODUCTION

The Job Shop Problem (JSP) is a challenging problem in manufacturing due to the fact that a factory typically produces several product types, each of them having their unique production sequence with selection choices among several machines to perform each task. In addition, there are requirements for additional resources at each step that have stochastic elements. This class of problems has been well studied over the years with a good overview available in Xiong et al. (2022).

Many approaches have been developed and implemented to solve complex JSPs. Each approach has its strengths and weaknesses. One such approach involves developing metaheuristic algorithms which aim to reduce the makespan. Another approach is Advanced Production Scheduling (APS) systems, designed to handle different types of scheduling problems, including those involving single machines, two machines, and job shops. Simple heuristic rules are used to guide scheduling decisions. As JSPs become more complex, it becomes difficult to apply traditional optimization methods. One of the major challenges is stochastic processing times that are modeled using statistical distributions. The uncertainty in the process makes it harder to obtain optimal and reliable schedules. The stochastic processing times complicate the process of adjusting constraints and objectives within metaheuristic algorithms. Furthermore, JSPs are susceptible to unexpected events, such as equipment breakdowns. These factors create production bottlenecks while subsequent maintenance and repair activities require additional time for rescheduling. These issues require new approaches to model and solve JSPs. One such approach is Reinforcement Learning (RL).

Recent research studies have shown that RL is a promising alternative for addressing complex JSP scenarios, particularly those with variable processing times and machine reliability issues. Unlike traditional methods, RL enables an agent to learn from experience and adapt to changing circumstances. RL algorithms can learn to make scheduling decisions in the presence of uncertainty and can also learn to respond to disrupting events. By learning to optimize in dynamic and unpredictable environments, RL has the potential to significantly improve the efficiency and resilience of job-shop production systems (Wang et al. 2023). The capabilities of RL can be extended to Digital Twins (DTs).

This paper studies the application of RL in enhancing the productivity of a simulated job-shop production line by optimizing its order-releasing policy. FlexSim is used to demonstrate the concept with the availability of a socket function to integrate with Python as shown by Leon et al. (2022). The RL-based optimization method can be developed in Python integrating with the JSP problem modeled in FlexSim.

Section 2 provides a brief literature review of related works. Section 3 explains how the construction of the JSP system. In Section 4, we introduce the decision-making process of our RL-based method. Section 5 discusses how the RL algorithm is connected to FlexSim using a Python socket interface. Section 6 discusses and performs a quantitative analysis of the experimental results with different heuristic scheduling rules. Finally, Section 7 summarizes the key findings and contributions of the study.

2 LITERATURE REVIEW

There are four major areas in this research study: (i) the DT software used for JSP; (ii) heuristic algorithm for scheduling rules; (iii) stochastic parameters of a system; and (iv) the main RL tool.

2.1 Selection of Simulator for the JSP

One objective of this study is to develop a simulation-based digital twin of a job-shop production system that enables a convenient and accessible connection to external algorithms, specifically RL agents. The simulation platform must not only support detailed modeling of job-shop production dynamics—such as machine routing, resource constraints, and stochastic processing times—but also allow real-time interaction with external decision-making agents. This connectivity is essential to enable closed-loop experimentation, where intelligent control policies can be dynamically evaluated and refined.

Several discrete-event simulation (DES) tools have been used for modeling manufacturing environments, including job-shop systems. A comprehensive review by Qiao and Wang (2021) highlights the application of DES tools, such as Arena, AnyLogic, Simio, and FlexSim in production system design, flow analysis, and resource management. Each supports DES modeling with different emphases and capabilities. AnyLogic combines DES with agent-based and system dynamics modeling, offering flexibility for simulating decentralized or hybrid systems. Guizzi et al. (2020) used AnyLogic to model a job-shop environment and evaluate performance under alternative dispatching policies. Simio supports object-oriented modeling and simulation-based scheduling, and has been applied to complex production environments as shown by Thiers et al. (2016). Arena has been used for DES-based analysis. Islam et al. (2019) developed an Arena-based job-shop production model to compare performance under various scheduling rules.

While these platforms are each capable of modeling job-shop production, they differ in their ability to support the integration objective of this study. AnyLogic's use of Java as the primary scripting language provides modeling flexibility but adds complexity when integrating with Python-based RL agents. Real-time data exchange typically requires substantial custom development. Simio, though effective for production system analysis, does not include built-in mechanisms for real-time communication with external algorithms and may require third-party middleware for integration. Arena is well-established and intuitive for DES modeling but lacks native support for external API connectivity and offers limited extensibility for interfacing with external learning frameworks. These constraints present barriers to establishing a direct, dynamic link between simulation and external learning agents.

FlexSim was selected as the simulation environment for this study due to its compatibility of creating a real-time interactive DT for job-shop production. FlexSim offers an object-oriented DES platform with a visual interface and prebuilt industrial components for modeling job-shop systems. Crucially, it provides a native Python API that supports two-way communication with external scripts during simulation runtime. This capability allows external algorithms to interact with the model in real time—observing system states, selecting control actions, and receiving performance feedback—without requiring extensive middleware or language translation. The selection of FlexSim is thus grounded in its alignment with the integration and experimentation needs of this research study.

In this study, FlexSim serves as both the modeling environment for the job-shop production system and the execution platform to test learning-based control strategies. Its capabilities have been previously validated. Wu et al. (2018) used FlexSim to simulate and improve manufacturing operations. Zhu et al. (2014) applied it in the context of cold-chain logistics to identify process bottlenecks. Leon et al. (2022) demonstrated integration of FlexSim with Python-based algorithms to enable intelligent optimization. These examples support FlexSim's applicability to real-world production environments and its role as a simulation framework capable of interacting with external control logic. This is the basis for using FlexSim as the choice for creating a DT that reflects job-shop operations while supporting real-time algorithmic control and experimentation.

2.2 Stochastic Parameters of a System

Reliability modeling is an essential component in simulating job-shop production systems, where equipment availability directly affects scheduling and throughput. Two standard metrics used in this context are Mean Time Between Failures (MTBF) and Mean Time to Repair (MTTR). MTBF represents the expected operational time before a failure occurs, while MTTR captures the average time required to restore functionality. In simulation practice, both metrics are commonly modeled as stochastic variables, reflecting the unpredictable nature of real-world machine behavior. Pusztai et al. (2022) identify MTBF and MTTR as core indicators in production system simulation and emphasize the importance of probabilistic modeling for accurate system representation. Rather than assigning fixed values, simulation studies often sample from distributions to reflect observed variability in failure and repair processes (Nelson (2003)). Jardine et al. (2006) further recommend the use of historical maintenance records or condition-monitoring data to inform these models, especially when applying reliability-based decision-making strategies.

In addition to failure and repair uncertainty, stochastic processing times are a well-documented source of variability in job-shop operations. These arise from differences in operator efficiency, part complexity, or setup conditions. Law and Kelton (2007) indicate that ignoring such variability can lead to misleading simulation outputs, particularly when estimating queue times, utilization rates, or makespan.

Together, MTBF, MTTR, and processing time variability represent key sources of operational uncertainty. Their inclusion in DT models enhances realism and supports a more robust evaluation of scheduling and control strategies. Modeling these parameters stochastically enables simulation of dynamic conditions that more closely reflect real production environments.

2.3 Heuristic Scheduling Rule

Heuristic scheduling rules, also known as dispatching rules, are widely used in job-shop production systems to provide fast and practical scheduling decisions. These rules prioritize jobs based on predefined criteria and are valuable when computational resources are limited or when quick decisions are required on the shop floor (Lödding 2013). Common rules include First Come First Served (FCFS), Shortest Processing Time (SPT), Earliest Due Date (EDD), Longest Processing Time (LPT), Least Work Remaining (LWKR), and Most Work Remaining (MWKR). Each rule targets different performance objectives—such as reducing makespan, minimizing tardiness, or balancing machine utilization. Lödding (2013) offers a detailed taxonomy and performance classification of these rules based on empirical studies and production contexts.

Despite their simplicity and ease of implementation, static heuristic rules have limited flexibility. Studies show that their effectiveness often declines in stochastic environments, where disturbances like machine breakdowns, variable processing times, or urgent job arrivals can disrupt planned sequences (Chan et al. 2017). Researchers have proposed methods to tune or combine heuristics adaptively, but the base logic of these rules remains fixed once chosen.

In this study, heuristic rules are used as benchmark strategies in the job-shop simulation. Their performance serves as a reference point for evaluating improvements achieved through more adaptive scheduling policies.

2.4 Reinforcement Learning (RL) Tool

Reinforcement Learning (RL) is a learning method where an agent interacts with a system and learns how to make better decisions over time. The agent observes the current state of the system, takes an action, and receives a reward based on how good that action was. As Sutton and Barto (1998) discussed, the agent learns which actions lead to better outcomes by trying different strategies and adjusting based on feedback.

In job-shop production systems, RL has been used to help machines or scheduling agents make better decisions in real time. Unlike fixed rules, RL can adjust to unexpected events like machine breakdowns or urgent job arrivals. For example, Waschneck et al. (2018) applied RL in a semiconductor factory and showed it could improve global production scheduling. Similarly, Cronrath et al. (2019) used RL in a digital twin setting to help the system learn from real-time operations. Zhang et al. (2020) developed a deep RL approach that learns dispatching rules directly from job-shop data. Their method performed well even as problem sizes increased and outperformed traditional rules such as FCFS or SPT.

Mao et al. (2016) demonstrate that time-based reward signals, such as job delays, can effectively guide RL agents in dynamic scheduling environments. This approach supports our use of completion-time gaps as a reward signal to optimize order release in job-shop systems. This study builds on that foundation, but focuses specifically on teaching an RL agent to optimize order-releasing policies and decide when and which jobs should enter the system. The RL agent is implemented using the OpenAI Gym interface (Brockman et al. 2016) and interacts with a shop system modeled using FlexSim.

3 JSP DIGITAL TWIN MODELING

This section introduces the details of the JSP system.

3.1 JSP Digital Twin Overview

We model a JSP system using FlexSim. In this model, there are ten machines that make up the production system. Products are transported between the machines. Products are initially stored in a queue (input buffer) near the entry point of the production system. There are three transporters for moving products between different machines and transfer queues. Upon completion, a transporter moves the completed products to the sink where they exit the system. Figure 1 shows the production system layout in FlexSim.

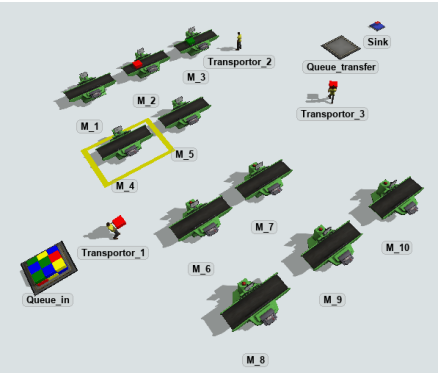


Figure 1: Screenshot of JSP production system simulated in FlexSim.

3.2 Processing Sequence

The JSP system considered in this study includes four product types, each with a predefined processing route composed of one or more sequential operations (see Table 1). Each row in the table corresponds to a product type, while the columns represent successive operations required to complete that product. For each operation, one or more alternative machines may be listed, indicating routing flexibility at that stage.

In this model, each machine assignment constitutes a single, indivisible operation, hereafter referred to as an atomic step. If multiple machines are available for a given operation, only one is selected at runtime based on machine availability and system status. This structure reflects a flexible job-shop environment, where routing decisions are made dynamically.

The segmentation of the processing route into steps is based on the sequence of required operations rather than on the number of unique machines. For example, Type 1 products can be processed by either M_1 or M_9 in Step 1. Products may revisit the same machine in later steps or share machines across different steps with other products, depending on the process flow design. Step 4 denotes the completion of the processing steps and movement to the sink for departure.

Table 1: Routing table for product types through processing steps.

Product Type	**Step 1**	**Step 2**	**Step 3**	**Step 4**
Type 1	M_1, M_9	M_1, M_4, M_6	M_2, M_4, M_{10}	Sink
Type 2	M_2, M_{10}	M_3, M_5, M_7	M_2, M_8	Sink
Type 3	M_1, M_8	M_2, M_4, M_7	M_3, M_{10}	Sink
Type 4	M_2	M_8	M_3, M_9	Sink

3.3 Random Events: Processing Time and Equipment Failure

In the JSP system, processing times for each machine operation are represented using truncated normal distributions, denoted as $N(\mu, \sigma)$, where μ is the mean processing time and σ represents the standard deviation, capturing the variability during the processing operations at each step.

Random breakdown and recovery times are configured in the DT environment. As discussed in Section 2.2, these stochastic characteristics can add real-world uncertainties to the JSP system. In the process, RL can learn how to process such uncertainties by making better decisions and increasing rewards. Machine repair times are uniformly distributed. Up-times, representing how long a machine will be able to run before it goes into a breakdown state, are exponentially distributed to capture the wide variability associated with typical equipment failure occurrences.

3.4 Job-Shop Production Flow

The product flow in the JSP is modeled using the *Process Flow* module in FlexSim. This module enables flexible and customizable system logic design. Algorithm 1 provides a general version of the product routing process, from creation of the new job to completion of the job at the sink.

4 REINFORCEMENT LEARNING FORMULATION

4.1 Problem Description

The production scheduling problem is modeled as a job-shop system consisting of n machines, multiple product types, and stochastic operational conditions. Each product type follows a specific routing sequence with processing time variability and may be affected by machine availability, random breakdowns, and queuing delays. The objective is to train a RL agent to dynamically select product types to release into the system to minimize total production time and improve throughput.

4.2 Markov Decision Process Model

The problem is formulated as a Markov Decision Process (MDP) defined by a state space $\mathscr{S}$, an action space $\mathscr{A}$, a transition function $\mathscr{P}(s'|s,a)$, and a reward function $R(s,a)$. The RL agent learns a policy $\pi_\theta(a|s)$ to maximize expected cumulative rewards.

Algorithm 1 Job operations are dynamically routed based on machine availability and process requirements.

```
Initialize:
p ← new product with assigned type
p.route ← processing sequence (list of machine options)
n ← number of steps in p.route
while simulation is running do
    for i = 1 to n do
        M ← p.route[i]                                  ▷ Candidate machines
        m ← select available machine from M
        send p to m and wait for processing
    end for
    send p to sink
end while
```

4.3 State Space

At each decision step t, the state $S_t \in \mathscr{S}$ provides a comprehensive snapshot of the system's current configuration. The state representation consists of the following components:

- **Machine Status Vector:** For each machine M_i, where $M_i \in \mathscr{M}$ and $\mathscr{M}$ is the set of all machines, the operational status is encoded as:
$$S_{\text{status}}(M_i) \in \mathscr{C},$$
where $\mathscr{C} = \{\text{idle}, \text{busy}, \text{down}, \text{repair}\}$ denotes the set of possible machine conditions.
- **Available Product Type at the Source:** Let $\mathscr{P}$ denote the set of all product types. The product type currently available for release at the source is represented by:
$$S_{\text{source}} \in \mathscr{P}.$$
- **Machine Processing Assignments:** Define a processing vector $\mathbf{P}_t = [P_t^{(i)}]_{M_i \in \mathscr{M}}$, where $P_t^{(i)} \in \mathscr{P} \cup \{0\}$ denotes the product type being processed on machine M_i at time t, or 0 if the machine is idle. Formally:
$$P_t^{(i)} = \begin{cases} 0, & \text{if } M_i \text{ is not processing any product,} \\ p, & \text{if product of type } p \in \mathscr{P} \text{ is assigned to } M_i. \end{cases}$$

4.4 Action Space

The action at each decision step is denoted by $A_t \in \mathscr{A} = \{1, 2, 3, \ldots, p\}$, where each element corresponds to a specific product type that can be released into the JSP system. The RL agent selects an action based on the current system state S_t, which captures relevant information such as machine availability, queue status, and system utilization.

The central objective of the RL agent is to **maximize total system throughput** by dynamically optimizing order-release policies. Rather than following static dispatch rules, the agent learns to make real-time decisions that respond to the dynamic state of the system. At each step, the agent determines which product type to release, directly affecting machine workloads, queue lengths, and system congestion. These affected factors have an impact on how rewards will perform.

By formulating the action space around product type selection, the model allows the RL agent to learn how different release patterns influence performance under stochastic conditions.

4.5 State Transitions

The transitions depend on multiple dynamic factors, including machine availability, product routing rules, processing durations, and equipment breakdowns or recoveries. These transitions are governed by the simulation model built in FlexSim, which acts as the digital twin of the job-shop system.

4.6 Reward and Reward Function

The RL agent is trained using a reward function designed to promote high throughput in the JSP system. The immediate reward is based on the time interval between consecutive product completions at the system sink. Let C_t represent the completion time of the t^{th} product; then the reward is defined as:

$$R_t = -\Delta T_t = -(C_t - C_{t-1}).$$

This formulation encourages the agent to minimize idle time between product completions, thereby promoting faster and more consistent production flow. Smaller time gaps result in less negative rewards, guiding the agent to increase completion frequency over time. In order to improve reward stability and reduce potential bias toward short-processing-time jobs, rewards are accumulated over batches of five completed products:

$$R_k = \sum_{j=t-k+1}^{t} R_j = -(C_t - C_{t-k}).$$

This batch-based reward structure serves two purposes. First, it smoothens fluctuations caused by stochastic events such as machine breakdowns or processing time variability. Second, it reduces the incidence of the RL agent over-prioritizing shorter jobs, encouraging a more balanced scheduling behavior that improves overall throughput. The choice of batch size $k = 5$ was determined empirically as a compromise between reward sensitivity and variance reduction.

The agent's policy π_θ is updated using the Proximal Policy Optimization (PPO) algorithm with a discount factor $\gamma = 0.99$, which ensures long-term scheduling quality while retaining responsiveness to local fluctuations. This reward design reflects the central goal of the study: to construct a simulation-based DT of a JSP system that enables a convenient and accessible interface to external RL algorithms. Performance metrics such as tardiness, slack time, fairness indices, or priority weights are initially excluded to simplify system dynamics and maintain compatibility with modular learning interfaces. Products types are introduced stochastically, and queue capacities are constrained to prevent excessive backlog, which further stabilizes agent learning.

The design aligns with RL strategies used in production systems, where time-based signals have been found effective in guiding learning in stochastic environments, as shown by Zhang et al. (2020) and Mao et al. (2016).

4.7 Interaction between RL Agent and DT Environment

At each time step t, the RL agent observes the current state S_t from the environment and selects an action A_t according to its policy $\pi(A_t|S_t)$. The environment responds with the next state S_{t+1} and an immediate reward R_t, which is used to update the agent's policy. Figure 2 illustrates the RL loop between the agent and the DT environment in FlexSim.

4.8 RL Agent Logic

The RL agent is designed to optimize order release decisions in a stochastic job-shop production environment. Its primary objective is to increase system throughput by dynamically selecting which product type to release into the system. The agent makes decisions based on the observed state of the system, which includes machine status, product availability at the source, and the current processing status of jobs.

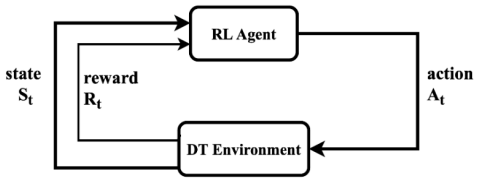


Figure 2: Interaction between RL Agent and DT Environment.

At each decision step, the agent selects an action $A_t \in \mathscr{A} = \{1, 2, \ldots, p\}$, representing the type of product to release, based on the current state S_t. The action is executed in the FlexSim DT environment. The environment monitors the time interval between successive product completions using the attribute `Sink.TimeDiff`. This time difference $\Delta T_t = C_t - C_{t-1}$ is used to compute the immediate reward:

$$R_t = -\Delta T_t = -(C_t - C_{t-1}).$$

A smaller ΔT_t corresponds to faster consecutive product completions and indicates higher throughput. Thus, minimizing ΔT_t aligns with the agent's learning goal. A batch-based reward policy is employed to stabilize learning in the presence of randomness. The agent accumulates rewards over every five completed products. When five completions are reached, the cumulative reward is calculated and used to update the agent's policy using the Proximal Policy Optimization (PPO) algorithm. This approach helps smoothen out transient fluctuations in performance and provides a more reliable learning signal. The complete decision-making and learning logic is summarized in Algorithm 2.

Algorithm 2 RL Agent Logic: State Observation, Action Selection, and Reward Accumulation.

Initialize: Reward buffer $\mathscr{R} \leftarrow \emptyset$
for each decision step t **do**
  Observe state S_t
  Select action $A_t \sim \pi_\theta(A_t|S_t)$
  Release product of type A_t
  Retrieve reward: $R_t = -$`Sink.TimeDiff`
  Reset: `Sink.TimeDiff` $\leftarrow 0$
  Append R_t to $\mathscr{R}$
  if `CompleteProduct` ≥ 5 **then**
    Compute $R_5 = \sum \mathscr{R}$; update policy π_θ
    Reset $\mathscr{R} \leftarrow \emptyset$; `CompleteProduct` $\leftarrow 0$
  end if
end for

5 CONNECTION BETWEEN RL ALGORITHM AND FLEXSIM VIA SOCKET

A real-time interaction between the RL agent and the JSP simulation environment is enabled via a socket-based architecture. The Python-based RL agent communicates with FlexSim using a TCP/IP socket, allowing decoupled yet synchronous data exchange. This setup supports modular experimentation by maintaining FlexSim as the DT environment while enabling external policy learning and decision-making via Python-based RL.

At each decision point, the FlexSim model sends the current state—encoded as a JSON object—to the RL agent, which responds with a selected product type to release. FlexSim processes this input and returns the immediate reward and updated state back to the agent. This loop continues throughout the simulation, enabling the agent to iteratively improve its policy. Such a design supports generalization across algorithms and aligns with prior studies that adopt modular learning architectures for production system control. Figure 3 depicts the communication architecture showing the two-way communication.

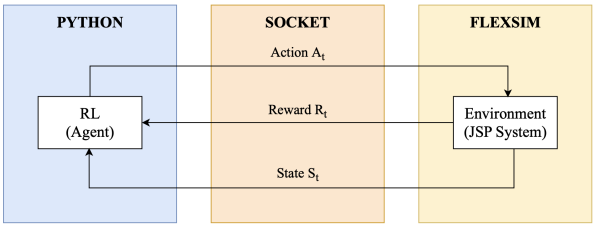


Figure 3: Communication loop between the RL agent and the environment via socket interface.

6 EXPERIMENTAL RESULTS

Here, we discuss some of the results from the experiments. We compared our proposed RL-based solution to the JSP with the following heuristic scheduling rules: FCFS (First come, first served), LWKR (Least Work Remaining Rule) and MWKR (Most Work Remaining Rule). The FCFS scheduling rule generates product types in order from 1 to 4 by default. In the FCFS policy, the machines process products that are sent in the order in which they are received. The LWKR policy aims to prioritize the job with the least processing time remaining, and the MWKR policy intends to release products that currently have the most total processing times remaining.

6.1 Hourly Throughput Comparison

Throughput was the primary performance measure that was used for comparison between the four scheduling policies. There were 10 random simulation replications with each replication running for 28,800 seconds in FlexSim. The heuristic dispatching rules—First Come First Served (FCFS), Least Work Remaining (LWKR), and Most Work Remaining (MWKR)—were selected to represent a variety of commonly used and theoretically significant scheduling logics. FCFS serves as a basic benchmark, while LWKR and MWKR offer contrasting workload-based prioritization strategies. These rules are frequently applied in the literature on job shop scheduling as previously discussed (Lödding 2013) and serve as appropriate baselines to assess the proposed RL-based approach developed in this paper.

Figure 4 illustrates the average hourly throughput across the four policies. The RL agent significantly outperforms the heuristics, consistently achieving higher throughput with a median around 135 and peaks above 150, compared to 110–125 for the other approaches. This result demonstrates the RL agent's capability to dynamically optimize order releases under stochastic and resource-constrained conditions.

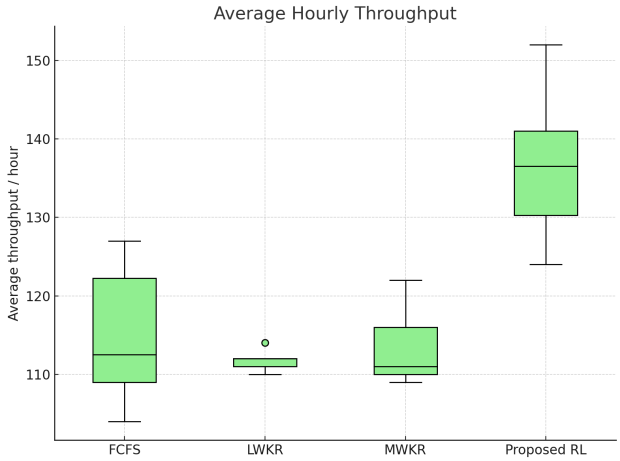


Figure 4: Average hourly throughput comparison among different scheduling rules.

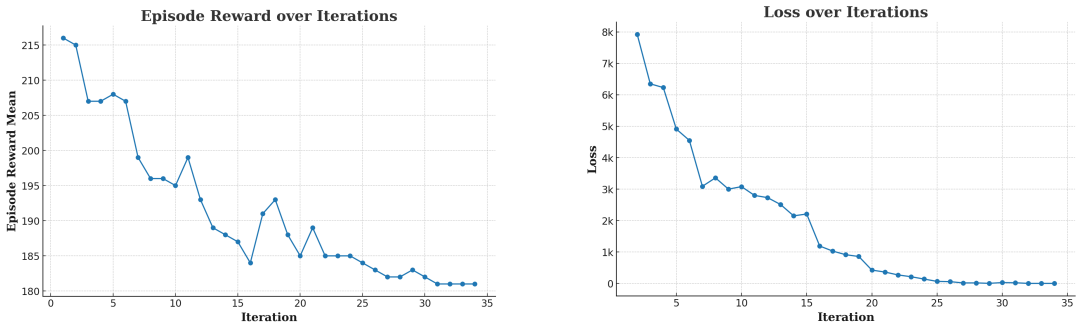
6.2 RL Training Performance

In order to track the improvement of the RL agent's performance over time, we recorded the average episode reward and the training loss during each iteration. These two metrics help us to understand the learning performance of the agent.

Figure 5a shows the average episode reward across 34 training iterations. Since the reward is defined as the negative time gap between product completions, a lower values implies faster production or higher throughput. The plot shows a clear downward trend, which implies the agent is learning to reduce delays and improve scheduling. Figure 5b displays the training loss over the same period. The loss starts high and steadily drops close to zero, showing that the policy and value networks are learning effectively.

6.3 Sensitivity Analysis

In order to understand the most significant state features affecting the agent's performance, we performed a sensitivity analysis using permutation importance (PI) from a trained Random Forest model. The PI approach can capture feature importance in the context of all other features. This is crucial for an effective JSP where the relationships between machine status, queues, and product types are often complex and non-linear (Breiman 2001). The sensitivity analysis is performed with 500 observations of states and rewards.



(a) Episode reward mean change.

(b) Training loss convergence.

Figure 5: Reinforcement learning performance indicators.

6.3.1 Top 3 Most Influential States

The results show that **State 12**—the number of products in the input queue at the system entry point—has the strongest influence on reward. As shown in Figure 1, all product types go through the shared input buffer *Queue_in*. When this queue is long, the product release is delayed, which increases time gaps between completions and lowers the reward. **States 16 and 17**, which represent the status of $\mathbf{M_3}$ and $\mathbf{M_4}$, also have a strong impact. These machines are part of several product routes and often handle longer processing tasks as shown earlier in Table 1. Their availability directly affects throughput and reward. Table 2 lists the top three most influential state features based on the analysis.

6.3.2 Top 3 Least Influential States

The sensitivity analysis also identified the least influential state features on the reward, specifically States 5, 6, and 7. These correspond to the product types currently being processed on Machines M_5, M_6, and M_7, respectively.

Machine M_5 is only used in Step 2 for processing Type 2 products. M_6 appears in Step 2 for product Type 1, while M_7 is active in a limited number of paths, specifically for product Types 2 and 3. Compared

Table 2: Top 3 most influential state features on reward.

State Index	Description	Importance Score
12	Number of items stored in the input queue before entering the production system	1.1996
16	Operational status of Machine M_3	0.0610
17	Operational status of Machine M_4	0.0513

to other machines, these are relatively underutilized and serve fewer product types across fewer steps. Additionally, they do not appear to be part of bottleneck-prone stages in the job-shop system. As a result, the product types being processed on these machines have minimal influence on the time gap between product completions—the primary component of the reward function. This explains their low sensitivity scores, as reflected in the PI results. A summary of these bottom three features is provided in Table 3.

Table 3: Top 3 least influential state features on reward.

State Index	Description	Importance Score
5	Product type processed at Machine M_5	0.0000
6	Product type processed at Machine M_6	0.0000
7	Product type processed at Machine M_7	-0.0003

7 CONCLUSION

This paper presents a reinforcement learning (RL)-based scheduling framework for optimizing throughput in a stochastic job-shop production environment modeled as a digital twin in FlexSim. Unlike conventional heuristic-based dispatching rules, our approach enables real-time, adaptive decision-making by learning from system dynamics.

We formulate the scheduling problem as a Markov Decision Process (MDP), where the RL agent observes machine states, queue levels, and product types to decide which product to release next for processing. The reward function is defined based on the time gap between consecutive product completions, encouraging faster production or higher throughput. Through extensive experimentation, we compare the proposed RL policy with traditional heuristics such as FCFS, LWKR, and MWKR. Results show that the RL agent consistently outperforms these baselines in terms of average throughput per hour. Sensitivity analysis further reveals that input queue length and the status of key machines (e.g., M_3, M_4) are the most influential factors affecting system performance. Training curves show improvement in episode rewards and policy convergence, confirming the effectiveness of the learning process.

While the RL-based method performs well under the tested conditions, it assumes that the mix of product types and system settings remains stable. In practice, production plans may change due to shifting demand, which could impact the agent's decision-making. As the current model is trained offline, it may require periodic retraining or adjustments to remain effective. Future work could explore online learning or model updates to improve the agent's ability to adapt to changing production conditions. These additions would make the system more robust and useful in real-world manufacturing settings.

REFERENCES

Breiman, L. 2001. "Random Forests". *Machine Learning* 45:5–32.

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang *et al*. 2016. "OpenAI Gym". *arXiv Preprint arXiv:1606.01540*.

Chan, F. T. S., N. Li, S. H. Chung, and M. Saadat. 2017. "Management of Sustainable Manufacturing Systems—A Review on Mathematical Problems". *International Journal of Production Research* 55(4):1210–1225.

Cronrath, C., A. R. Aderiani, and B. Lennartson. 2019. "Enhancing Digital Twins Through Reinforcement Learning". In *Proceedings of the IEEE International Conference on Automation Science and Engineering (CASE)*. August 22–26, Vancouver, BC, Canada, 293–298.

Guizzi, G., S. Vespoli, A. Grassi, and L. C. Santillo. 2020. "Simulation-Based Performance Assessment of a New Job-Shop Dispatching Rule for the Semi-Heterarchical Industry 4.0 Architecture". In *2020 Winter Simulation Conference (WSC)*, 1664–1675 https://doi.org/10.1109/WSC48552.2020.9383981.

Islam, M., T. Palit, and A. Mukaddes. 2019. "Development of an ARENA Simulation Model for Scheduling Problems in Job Shop Production". In *Proceedings of the International Conference on Mechanical, Industrial and Materials Engineering (ICMIME2019)*. December 17–19, Rajshahi, Bangladesh, 150–156.

Jardine, A. K. S., D. Lin, and D. Banjevic. 2006. "A Review on Machinery Diagnostics and Prognostics Implementing Condition-Based Maintenance". *Mechanical Systems and Signal Processing* 20(7):1483–1510.

Law, A. M., and W. D. Kelton. 2007. *Simulation Modeling and Analysis*. 3rd ed. New York: McGraw-Hill.

Leon, J. F., P. Marone, M. Peyman, Y. Li, L. Calvet, M. Dehghanimohammadabadi *et al*. 2022. "A Tutorial on Combining FlexSim with Python for Developing Discrete-Event Simheuristics". In *2022 Winter Simulation Conference (WSC)*, 1386–1400 https://doi.org/10.1109/WSC57314.2022.10015309.

Lödding, H. 2013. *Handbook of Manufacturing Control: Fundamentals, Description, Configuration*. 1st ed. Berlin and Heidelberg: Springer.

Mao, H., M. Alizadeh, I. Menache, and S. Kandula. 2016. "Resource Management with Deep Reinforcement Learning". In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets 2016)*. November 9–10, Philadelphia, PA, USA, 50-56.

Nelson, W. B. 2003. *Applied Life Data Analysis*. 1st ed. New York: John Wiley & Sons.

Pusztai, L. P., L. Nagy, and I. Budai. 2022, May. "Selection of Production Reliability Indicators for Project Simulation Model". *Applied Sciences* 12(10):5012.

Qiao, D., and Y. Wang. 2021. "A Review of the Application of Discrete Event Simulation in Manufacturing". *Journal of Physics: Conference Series* 1802(2):022066.

Sutton, R. S., and A. G. Barto. 1998. *Reinforcement Learning: An Introduction*. 1st ed. Cambridge, MA, USA: MIT Press.

Thiers, G., T. Sprock, L. McGinnis, A. Graunke, and M. Christian. 2016. "Automated Production System Simulations Using Commercial Off-the-Shelf Simulation Tools". In *2016 Winter Simulation Conference (WSC)*, 1036–1047 https://doi.org/10.1109/WSC.2016.7822163.

Wang, B., Y. Sun, H. Jung, L. D. Nguyen, N. S. Vo, and T. Q. Duong. 2023. "Digital Twin-Enabled Computation Offloading in UAV-Assisted MEC Emergency Networks". *IEEE Wireless Communications Letters* 12(9):1588–1592.

Waschneck, B., A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp *et al*. 2018. "Optimization of Global Production Scheduling with Deep Reinforcement Learning". *Procedia CIRP* 72:1264–1269.

Wu, G., L. Yao, and S. Yu. 2018, June. "Simulation and Optimization of Production Line Based on FlexSim". In *Proceedings of the 30th Chinese Control and Decision Conference (CCDC 2018)*, 3358–3363. Shenyang, China. June 9-11, 2018.

Xiong, H., S. Shi, D. Ren, and J. Hu. 2022. "A Survey of Job Shop Scheduling Problem: The Types and Models". *Computers & Operations Research* 142:105731.

Zhang, C., W. Song, Z. Cao, J. Zhang, P. S. Tan, and C. Xu. 2020. "Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning". In *Advances in Neural Information Processing Systems 33*, edited by H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin, 1621–1632. Red Hook, NY, USA: Curran Associates, Inc.

Zhu, X., R. Zhang, F. Chu, Z. He, and J. Li. 2014. "A FlexSim-Based Optimization for the Operation Process of Cold-Chain Logistics Distribution Centre". *Journal of Applied Research and Technology* 12(2):270–288.

AUTHOR BIOGRAPHIES

JIA-MIN CHEN is a master's student in the Wm Michael Barnes '64 Department of Industrial & Systems Engineering, Texas A&M University in College Station, TX, USA. His research interests include digital twin technology and discrete event simulation. His email address is chenjiamin85917@tamu.edu and his website is https://www.linkedin.com/in/jia-min-chen-94058a284/.

BIMAL NEPAL is a Professor and J.R. Thompson Department Head Chair in the Department of Engineering Technology and Industrial Distribution at Texas A&M University, College Station, TX, USA. His research interests include integration of supply chain management with new product development decisions, manufacturing systems optimization, supply chain resiliency, distributor service portfolio optimization, and engineering education. His e-mail address is nepal@tamu.edu and his website is https://engineering.tamu.edu/etid/profiles/bnepal.html.

AMARNATH BANERJEE is a Professor in the Wm Michael Barnes '64 Department of Industrial & Systems Engineering, Texas A&M University in College Station, TX, USA. His research interests include modeling, simulation and visualization, with applications in health care systems, information systems, manufacturing systems and Industry 4.0. His e-mail address is banerjee@tamu.edu and his website is https://engineering.tamu.edu/industrial/profiles/abanerjee.html.