Manufacturing and Science Engineering Conference
MSEC2024
June 17–June 21, 2024, Knoxville, TN

MSEC2024-XXXX

A DEEP REINFORCEMENT LEARNING APPROACH FOR PRODUCTION SCHEDULING IN COMPUTER SERVER INDUSTRY

Azzam Radman
University of Louisville
Louisville, KY

Faisal Aqlan[*]
University of Louisville
Louisville, KY

Pratik Parikh
University of Louisville
Louisville, KY

Md. Noor-E-Alam
Northeastern University
Boston, MA

ABSTRACT

Computer Server Industry is characterized by extensive test processes to ensure high quality and reliability of the servers. Computer Server Industry production systems utilize Configure-To-Order (CTO), also known as fabrication/fulfillment, strategy which provides an effective balance between demand and supply by synchronizing the flow of materials, equipment, and labor throughout the production process. In the fabrication stage, components or sub-assemblies are produced, tested, and assembled based on a projected production plan. They are then kept in stock until an actual order is received from a customer. In the fulfillment stage, final products are assembled according to actual customer orders. Assignment of products to test cells during the fulfillment stage can be a challenging task due to high quality requirement and limited resources. Current practices tend to assign products to test cells based on a specific criterion such as on-time shipment or maximum test cell occupancy, which can result in higher levels of energy consumption or delayed orders. This paper introduces a Deep Reinforcement Learning (DRL) approach to effectively assign servers to test cells considering a multi-objective reward function that combines multiple criteria. A proposed simulation model serves as the environment with which the DRL agent interacts, learning a policy that develops a test schedule for the products. The proposed approach is tested with a case study from a high-end server manufacturing environment. Sensitivity analysis is conducted to analyze the impact of the different values of the system's variables on its performance.

Keywords: Deep Reinforcement Learning, Reward Function, Production Scheduling, Computer Server Industry.

[*]Corresponding author: faisal.aqlan@louisville.edu
Documentation for `asmeconf.cls`: Version 1.34, March 11, 2024.

1. INTRODUCTION

Production scheduling in manufacturing systems focuses on determining the sequence of operations, allocating resources, and setting timelines to ensure optimal utilization of available resources while minimizing costs and maximizing productivity. In Configure-To-Order (CTO) manufacturing environments, a production schedule should achieve an effective balance between demand and supply by synchronizing the flow of materials, equipment, and labor throughout the production process. A CTO production environment consists of hybrid build-to-plan and assemble-to-order operations [1]. This configuration is also known as the fabrication/fulfillment strategy, in which products are customized and configured based on customer specifications within a predefined set of options and variations. Unlike make-to-stock or make-to-order strategies, where products are either produced in advance or manufactured completely from scratch, the fabrication/fulfillment strategy allows for a higher degree of customization while maintaining some level of standardization. In the fabrication stage, components or sub-assemblies are produced, tested, and assembled based on a projected production plan. They are then kept in stock until an actual order is received from a customer. In the fulfillment stage, final products are only assembled according to actual customer orders. There are two key production planning and scheduling decisions in a CTO system: (1) optimizing the build-to-plan subsystem (for components and sub-assemblies) by determining the production and inventory levels for a given forecast (exogenously provided), and (2) assembly and test planning and scheduling in the assemble-to-order subsystem in which orders are allocated to assembly and test processes.

In CTO manufacturing systems, developing solutions to optimize the production schedule has been an active area of research due to the high impact of these solutions on real-world applications [2]. Previous studies employed both heuristics and numerical optimization methods to approach the production scheduling task. However, these methodologies exhibit inherent limitations that restrict their applicability to specific domains or contexts. Heuristics such as Early Due Date First often yield solutions that are intuitive and easily comprehensible; however, their feasibility is constrained by their limited applicability across diverse domains or contexts [3]. Conversely, numerical solutions necessitate a more complex implementation process, involving a comprehensive mathematical description of the environment and objectives alongside the high computational intensiveness. Lenstra et al. [4] demonstrated that a significant portion of scheduling problems similar to those encountered in industrial fields are characterized as NP-complete. Nonetheless, they have the advantage of yielding optimal solutions despite their increased complexity [5]. Furthermore, the existing complexities posed by factors such as increased product diversity and environmental uncertainties emphasize the need for novel methodologies in production scheduling [6]. These approaches should be capable of effectively adapting to abrupt variations in such dynamic environments, thereby generating timely and viable solutions that meet the required criteria [7].

Another class of approaches commonly referred to as data-driven production scheduling has recently emerged as an alternative to traditional methods [8, 9]. The emergence of big data, machine learning (ML), and information technologies has prompted a growing interest among researchers in employing the potential of manufacturing data. These methods involve a three-step process. Initially, the state features are filtered, and only the relevant ones are selected. Second, optimal samples are selected according to predefined criteria. These selected samples encompass both the input features and the scheduling decisions. Finally, an ML model is trained to establish a mapping between the input features and the optimal decisions.

Although data-driven approaches offer the advantage of simplifying the system formulation compared to optimization techniques, they exhibit several drawbacks, including the highly iterative nature of feature and optimal sample selection [10]. *DRL* presents itself as a means to address these challenges, enabling both automatic feature selection and engineering. Moreover, *DRL* emerges as a promising solution to these limitations, as it facilitates both automatic feature selection and engineering. Furthermore, *DRL* eliminates the requirement for labeled data, which might be infeasible in specific contexts either due to the high complexity of the system where the outcomes are hard to predict or the lack of field experts who are capable of building such datasets. Rather, *DRL* directly interacts with production environments, and this interaction, in turn, streamlines the collection of the data required by the *DRL* agent to learn the policy [10].

In this study, we explore the potential of *DRL* for conducting scheduling tasks within a high-end server manufacturing setting. We start by establishing the Markov Decision Process (MDP) in which both the agent and the environment are explicitly defined. The latter is represented via a discrete event simulation (DES) model that resembles the current state of the system and yields a change when an action is taken by the agent. The DES model facilitates the representation of uncertainties and serves as a training platform for the *DRL* agent. This process, in turn, aids in the development of an agent capable of functioning effectively in a dynamic environment that encompasses both failures and uncertainties. The *DRL* agent, on the other hand, is a Deep Neural Network (DNN) that receives its inputs from the simulation model and generates action outputs, which are then applied within the same environment. As a result of these actions, the environment's state undergoes updates, resulting in the acquisition of a reward for achieving that particular state.

2. LITERATURE REVIEW

Reinforcement Learning (*RL*) has emerged as a highly promising and viable alternative approach for addressing production scheduling challenges, particularly in light of its impressive performance showcased in video game domains [11, 12]. An early work investigated the potential of employing *RL* to automatically infer domain-specific heuristics to find short, conflict-free schedules [13]. The Temporal Difference (TD) algorithm was employed to train a neural network to learn a heuristic evaluation function over states. This proposed approach outperformed the best existing scheduling algorithm at the time, namely Zweben's iterative repair method based on simulated annealing. The work presented *RL* as a high-performance method to solve scheduling tasks [14].

Another early study employed neural networks to learn near-optimal policies to solve job shop scheduling problems [15]. The primary focus of the study was directed towards the minimization of the cumulative tardiness observed in two distinct task scenarios, namely the single resource case and the multi-resource case. The proposed methodology has demonstrated a notable enhancement when compared to all existing heuristic approaches in both experimental settings. Furthermore, it has exhibited superior generalization abilities when tested on previously unseen scenarios, surpassing the performance of these heuristics.

[16] have conceptualized the production scheduling problem within the framework of multi-agent reinforcement learning. Each resource in the system is associated with an adaptive agent, entrusted with acquiring knowledge to formulate its own dispatching policies in an autonomous manner, disregarding the actions of other agents. The introduced methodology has exhibited competitive performance compared to alternative heuristic techniques, indicating its potential for further advancement to enhance its performance.

Lin et al. [17] have devised a multi-class Deep Q-Network (DQN) approach to address the job shop scheduling problem, specifically in the context of a semiconductor manufacturing system. The study assumes the presence of M machines, each assigned J jobs. A deep neural network (DNN) was trained to incorporate order inputs in addition to system inputs, enabling the prediction of the optimal dispatching rule from a set of seven predefined rules, namely FIFO, SPT, LPT, MOPNR, LOPT, SQN, and LQN. Once the selected dispatching rule was determined, it was transmitted to the corresponding edge devices responsible for implementing the rule on the associated machine. In cases where the generated schedule proved to be infeasible, the authors adopted a two-step approach for schedule repair, utilizing the MOPNR and SPT rules. The selection of actions following the calculation of values for each of the predefined rules was executed through the employment of an epsilon-greedy policy. The primary objective of the study was to optimize the makespan of the manufacturing system. The experimental outcomes of the proposed methodology exhibited superior performance when compared to the individual utilization of all other dispatching rules.

Liu et al. [18] used multi-agent actor-critic deep reinforcement learning (*DRL*) to address job shop scheduling. They employed two CNN-based models: one to evaluate states (critic network) and the other to select actions (actor network). The approach outperformed simple dispatching rules; meanwhile, optimization methods performed better. Nonetheless, the authors emphasized that the optimization techniques used in their study were constrained to deterministic environments where the entire scenario was predefined. In contrast, *DRL* demonstrated robustness in dynamic settings, rendering it a more viable choice for real-world applications.

In their work, Baer et al. [19] introduced a conceptual multi-agent *DRL* model as a method for online scheduling in flexible manufacturing systems (FMS). The study started by formulating the scheduling problem as a Markov Decision Process (MDP) [20]. To model the FMS, a Petri net was employed [21]. In the proposed multi-agent approach, each agent is assigned to control an individual product at a given time. This approach offers several advantages over a single global agent, including reductions in both the state and action spaces. The benefits of this method alleviate the complexities associated with using a single large DNN to handle such dimensions, as it would require extensive computational resources and exponentially extend the training time. Following that, all agents underwent individual training and subsequently were optimized together towards a shared global goal.

After that, Baer et al. [22] implemented their conceptual model outlined in their earlier work [19]. The authors developed a multi-agent *DRL* model to tackle the online scheduling problem in flexible manufacturing systems (FMS). In this study, rather than assigning machines to agents, the products themselves were allocated to agents. Each agent pursued its own optimization goals, which might differ from those of other agents. The agents shared their parameters during training, enabling the training of a single Q-value function. The evaluation of the model in this work encompassed its comprehension of tasks, robustness to variations, scalability, and generalization capabilities. The proposed method demonstrated its proficiency in all four aspects, achieving near-optimal solutions even in the presence of failures and variations in the system.

Hubbs et al. [23] utilized *DRL* in the context of chemical production scheduling. The study assessed the performance of the *DRL* model in the presence of uncertainties, where the agent was tasked with conducting online, dynamic scheduling for chemical processes. The *DRL* model exhibited superior performance compared to the mixed-integer linear programming (MILP) model and produced competitive results when compared to a shrinking horizon MILP approach in terms of the percentage mean optimal gap. The findings of this work highlighted the real-time optimized schedules generated by the *DRL* method, underscoring its advantages over traditional techniques in the chemical production scheduling domain.

In their work, Zhu et al. [24] tackled the resource scheduling problem (RSP) in the context of cloud manufacturing (CMfg) through the utilization of *DRL*. The proposed framework (Sharer) displayed notable advantages over leading heuristics, including the MSQL algorithm [25], NSGA-II [26], and the ICPSO algorithm [27]. The performance evaluation was based on three key metrics: storage utilization, average completion time, and the average cost-satisfaction ratio. These results underscored the efficacy of the *DRL* approach in addressing the resource scheduling challenges in the field of cloud manufacturing. Moreover, the study highlights *DRL*'s adaptability to scheduling tasks in dynamically changing conditions, where the agent's training environment was a simulation model designed to emulate the uncertainties encountered in real-world cloud manufacturing scenarios.

In this work, we leverage *DRL* to tackle a highly complex task in server assembly systems, i.e., server-to-test cell assignment. The approach starts with the development of a simulation model that closely mimics the environmental factors essential for the *DRL* model. This simulation model aims to encapsulate the complex dynamics of the problem, ensuring that the *DRL* model is trained on a highly realistic environment. Furthermore, since the *DRL* agent is trained using the simulation model, it exhibits the ability to generate schedules that are resilient to uncertainties, such as variations in arrival and processing times. This adaptability makes our model robust and more suitable for real-world deployment compared to static scenario-based scheduling techniques. In this work, we aim at exploring the ability of DRL to produce efficient server schedules in the complex server environment. The ability of DRL in this work will be focused on monitoring compatibility in assignment along with utilization rate.

3. PROBLEM DESCRIPTION

In the computer server industry, the fulfillment test assignment policies usually rely on simple dispatching rules (e.g., assigning products to test cells based on the plant scheduled ship date (PSSD)) [28] or heuristics [29]. These policies assume the knowledge of all the assigned orders in advance, and based on that, the orders are allocated to test cells. Prior to the assignment, forecasting for server arrival is carried out, and the assignment takes place based on the forecasted scenarios. Despite the ability of these simple dispatching rules to assign servers, these rules usually focus on single objectives such as increasing utilization rate or minimizing makespan. Handling more complex multiple objectives cannot be performed by such dispatching rules and that calls for more intelligent solutions.

In this study, we approach this task in a more realistic manner, mirroring real-world conditions. First, we assume orders arrive sequentially, and prior forecasting for the arrivals is not employed. Then, upon server arrival, the requirements of the servers alongside the test environment features are fed to the model to determine the best cell for the incoming server.

This approach offers several advantages. First, it better represents real-world test environments where servers arrive sequentially, making the solution more practical. Second, the DRL method is based on a DNN that acts as the decision-maker agent. The inference time for such a DNN is considerably low, al- lowing for attaining near-real-time scheduling solutions. Finally, the training environment is a simulation model that encompasses stochasticity in both arrival and processing times, enabling the agent to learn to solve stochastic environments rather than solving a predefined deterministic scenario.

4. METHODOLOGY

Figure 1 shows the proposed research framework, which comprises two primary components: the environment and the agent. The agent is essentially a DNN model responsible for making decisions based on the inputs it receives. These inputs come from two distinct sources: one set originates from the incoming server, while the other set is extracted from the environment. The agent's actions cause a change in the environment and subsequently update its state, which results in generating a reward value for the agent for taking that action. The following two subsections provide a comprehensive description of the agent and environment.

To test the integrated *DRL*-DES approach, a case study from a CTO manufacturing system is considered. The system typically follows a periodic review inventory policy, considering inventory holding, order fulfillment, and stockout costs (versus service level). A safety stock is used to cover uncertainty in demand and processing capacity while also considering transit time variability. Work-in-process (WIP) inventory also exists due to production lead times and testing requirements. A high level of fill rate is of utmost importance because revenue loss from missing a customer order exceeds inventory holding costs. The case study represents computer server manufacturing processes, which include fabrication assembly, fabrication test, dekitting and storage, followed by fulfillment assembly and test, and finally packaging and shipping.

4.1 Environment Description

The environment in this study represents the fulfilment test process of a manufacturing system that produces high-end computer servers. The manufacturing system under study comprises 56 test cells distributed across four test banks, each containing 14 cells. These test cells have different cooling (water-based, air-based, or both) and voltage (high or low) specifications that must align with the requirements of the tested servers. Despite their varying specifications, test cells exhibit roughly similar power consumption, measured as the energy required per unit time. Moreover, each test bank has a dual cooling system: primary - computer room air conditioner (CRAC) and secondary - cooling distribution unit (CDU). Notably, the CRAC has a power consumption of 176 kW, while the CDU operates at 150 kW. Activation of these cooling units is contingent upon the number of concurrently active cells within the test bank. The CRAC can handle up to seven active cells before the CDU is activated should one more cell be added to the test bank.

4.2 Simulation Model

The overall simulation process is shown in Figure 2. The simulation involves monitoring three primary events generated by three distinct sources. These events consist of server arrivals, test initiations, and test completions, which are triggered by the server generator, the queue, and the test bank, respectively. The server generator creates servers at intervals following an exponential distribution characterized by the parameter λ. Upon server generation, their cooling and voltage specifications, as well as their service times, are drawn from both uniform distributions.

The subsequent event detector identifies the next event as the one with the soonest upcoming time. Depending on the type of the next event, the corresponding process is activated, leading to a system update. Regardless of the nature of the next event, and before activating the process corresponding to the next system, the monitored system metrics get updated, including simulation time, power consumption, total response time for the servers in the system, and cell utilization. Subsequently, the corresponding event is processed, leading to system updates.

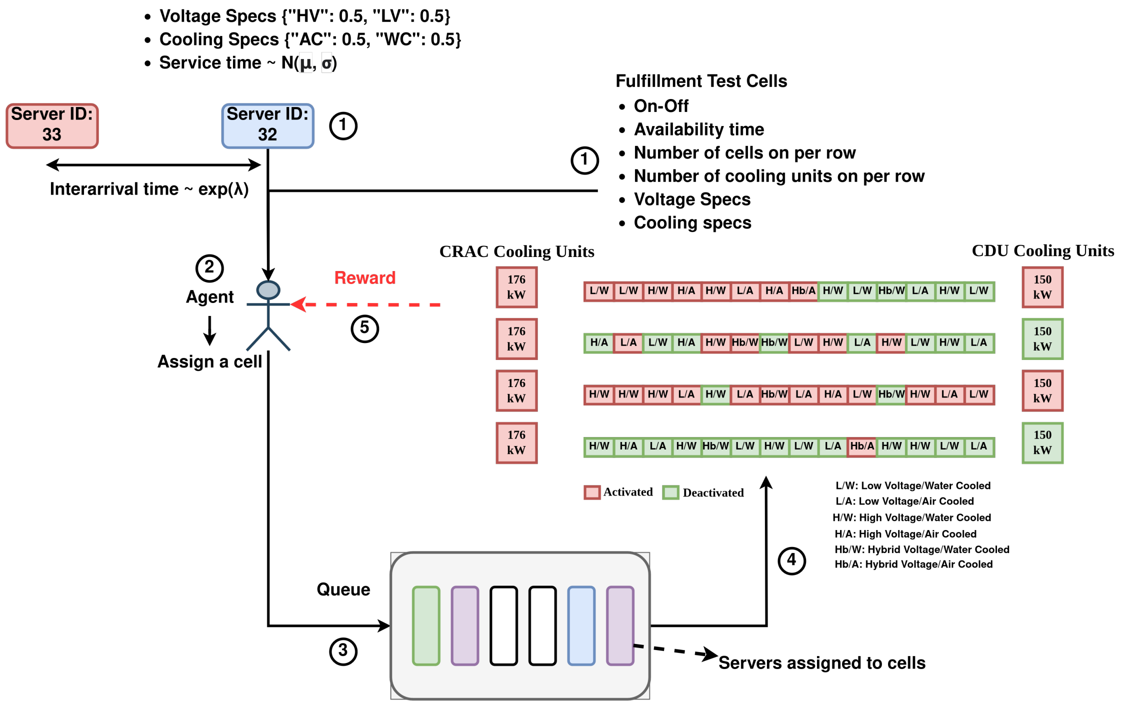
In the case of the arrival event, both counters for the number of servers in the system and the number of arrivals increment by one. Then, the server features are transmitted to the *DRL* agent along with the current environment features. The agent assigns a cell for the server to be tested on, which then joins the waiting queue until the designated cell becomes available. Following this, the next server is generated after a time interval sampled from an exponential distribution.

For the service initiation event, the server leaves the queue, is directed to the assigned testing cell, and triggers an update in the state of the bank that manages operating cells. For the service completion, on the other hand, the server vacates the testing cell, resulting in an update of the cell operation states, along with an adjustment to the counters tracking the number of departures and the number of servers in the system.

The simulation model keeps track of critical metrics, including response time, queue duration, power consumption, and resource utilization. Starting at the initiation of the simulation, the simulation clock updates whenever a new event takes place. The arrival time of each server is recorded alongside the time in the queue, enabling the computing of the response time for each individual server. Moreover, test cell utilization is calculated as test cell occupancy over total simulation time, which can then be used to compute the overall utilization of the system. Additionally, the durations for which the cooling units are in operation are monitored. The power consumption of the cooling units is computed by multiplying these durations by their respective power consumption rates.

4.3 *DRL* Agent

The architecture of the *DRL* agent is designed to handle all types of input originating from both the test environment and the incoming server. In terms of dimensionality, there are two types of inputs: 1D and 2D inputs. The 2D inputs serve to depict the state of the test cells, encompassing factors like occupancy status, cooling and voltage specifications, compatibility, and the time of availability. On the other hand, the 1D vector inputs are employed to represent other features, including the count of active cells and cooling units within each bank, the voltage and cooling requirements of the incoming server, and the time needed for test completion. The state inputs to the model are explained in detail



Voltage Specs {"HV": 0.5, "LV": 0.5}
Cooling Specs {"AC": 0.5, "WC": 0.5}
Service time ~ N(μ, σ)
Server ID: 33
Server ID: 32
1
Interarrival time ~ exp(λ)
1
Fulfillment Test Cells
On-Off
Availability time
Number of cells on per row
Number of cooling units on per row
Voltage Specs
Cooling specs
CRAC Cooling Units
CDU Cooling Units
2
Agent
Assign a cell
Reward
5
176 kW
150 kW
L/W L/W H/W H/A H/W L/A H/A Hb/A H/W L/W Hb/W L/A H/W L/W
H/A L/A L/W H/A H/W Hb/W Hb/W L/W H/W L/A H/W L/W H/W L/A
H/W H/W H/W L/A H/W L/A Hb/W L/A H/A L/W Hb/W H/W L/A L/W
H/W H/A L/A H/W Hb/W L/W H/W L/W L/A Hb/A H/W H/W L/W L/A
Activated
Deactivated
L/W: Low Voltage/Water Cooled
L/A: Low Voltage/Air Cooled
H/W: High Voltage/Water Cooled
H/A: High Voltage/Air Cooled
Hb/W: Hybrid Voltage/Water Cooled
Hb/A: Hybrid Voltage/Air Cooled
4
Queue
3
Servers assigned to cells

FIGURE 1: DRL-DES INTEGRATED FRAMEWORK.

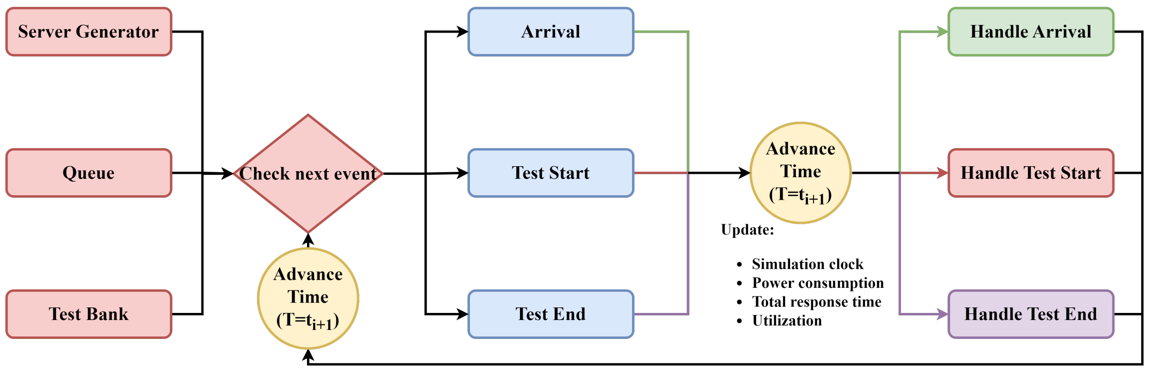


FIGURE 2: FLOWCHART OF THE OVERALL SIMULATION PROCESS.

in Table 1.

The four 2D input arrays are stacked to form a 4 × 4 × 14 input tensor, then undergo processing through a convolutional neural network (CNN). The CNN is composed of two sequential 2D convolutional layers, each followed by 2D maximum pooling layers. The output of the latter is projected to 1D via the 2D global average pooling layer. On the other hand, the 1D input side of the network is processed through fully connected (FC) layers. The outputs of both sides of the network are then concatenated and passed to an output layer containing the same number of test cells (56). In this work, the deep Q-network (DQN) algorithm [30] is adopted to learn the agent. In DQN, there are two identical networks in terms of architecture: the Q-network and the target Q-network, both trained simultaneously during the learning process. The Q-network is used to generate values for the state action pairs at the given state and based on that, an action is taken. The target Q-network, on the other hand, computes the values of the next state-action pairs which are used as targets to update the weights of the value network. The learning process starts by enabling the agent to interact with the environment and collect data in the form of tuples containing states, actions, rewards, next states, and termination states. The accumulated data points are stored in a replay memory that handles the most recent M number of samples, where M is a parameter defined by the user. Each time an N number of new samples are collected, where N represents the number of iterations for each update and is a predefined hyperparameter, the Q-network undergoes a gradient descent step according to the Bellman equation 1.

$$\begin{aligned} Q'(s_t, a_t) \leftarrow\, & Q(s_t, a_t) + \\ & \alpha \times [r_{t+1} + \gamma \times \max_a (Q_T(s_{t+1}, a)) - Q(s_t, a_t)] \end{aligned} \quad (1)$$

where Q' is the updated Q-network, Q is the current Q-network, Q_T is the target Q-network, t it the current time-step, s is the state, a is the action, r is the reward value, α is the learning rate for the gradient descent, and γ is the discount factor. The ϵ-greedy method is employed during the learning process which enables the agent to explore the environment. ϵ represents the probability of selecting an action at random, meanwhile 1-ϵ is the probability of selecting the action with the highest value, i.e. the greedy action. The value of ϵ is chosen to be high, close to 1, at the beginning of the training, then it decreases during the course of training allowing the model to exploit its knowledge effectively.

5. EXPERIMENTAL ANALYSIS

In this study, we explored the use of two reward functions. The first function provides rewards or penalties to the agent depending on its proficiency in assigning compatible cells to incoming servers. The second function incorporates a time penalty into the reward mechanism, wherein the server receives diminishing rewards as it spends more time in the queue. For this second configuration, we experimented with three different weighting settings. The upcoming section describes the evaluation metrics used to evaluate the performance of the baseline method and the two reward functions which are discussed in more detail afterwards.

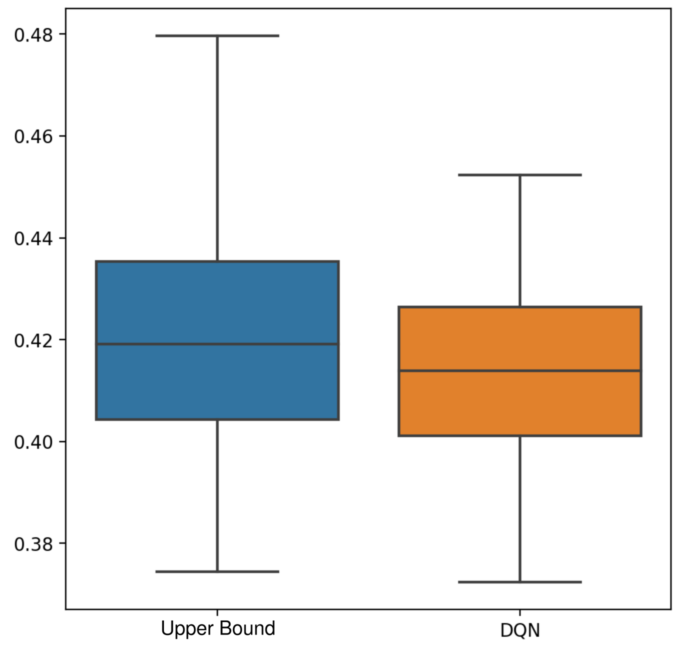


FIGURE 3: MEAN UTILIZATION RATES FOR BOTH UPPER BOUND RULE-BASED ASSIGNMENT AND DQN.

Given that, the simulation model has been run for 100 episodes, and the mean utilization rate of these runs is shown in Figure 3. The upper bound rule-based assignment algorithm scored a mean utilization rate of 0.419 across all the episodes compared to 0.414 for DQN which proves the efficiency of the learned policy with standard deviations of 0.0235 and 0.0182, respectively.

5.1 Reward Function 1

For the first reward function design, formulated in Equation 2, the function mainly aims at enabling the agent to learn a policy that makes correct assignments regardless of other factors, such as makespan or utilization. We assume this task is the most crucial, and mistakes by the agent cannot be tolerated. Assigning cells with specifications that do not match the requirements of the incoming server might lead to permanent damage to crucial and expensive components of the server. Figure 4 shows the reward function value, accuracy, and utilization progress during the training phase. As only one component is considered in this setting (compatibility), the behavior of the reward function and accuracy during the course of training is identical, starting at a low value at the beginning of training and then reaching near the maximum value after 800 episodes. However, we notice a drop in the utilization factor at later stages of training. This can be attributed to the nature of the reward function that does not account for factors such as load distribution that impact the utilization metric.

$$R_i = (b_{v_i} \times R_{v_i}) + (b_{c_i} \times R_{c_i}) \quad (2)$$

TABLE 1: DESCRIPTION OF STATE INPUTS TO THE AGENT. $\mathbb{Z}$ REPRESENTS INTEGERS, $\mathbb{Z}_m$ REPRESENTS m CATEGORIES, AND $\mathbb{R}$ REPRESENTS REAL NUMBERS.

State type	State	Definition	Unit and Distribution
Server attributes	Voltage requirement	Voltage level required by the server	$\in \mathbb{Z}_2$ (Unitless) ~ U{"HV": 0.5, "LV": 0.5}
	Cooling requirement	Air or water cooling	$\in \mathbb{Z}_2$ (Unitless) ~ U{"Air": 0.5, "Water": 0.5}
	Service time	Testing time required by the server	$\in \mathbb{R}^+$ (Hours) ~ $clip(N(30,5), 10, 90)$
Test cell attributes	On-off (occupancy status)	Test cell states	$\in \mathbb{Z}_2$ (Unitless)
	Availability time	Time at which the cell will become available	$\in \mathbb{R}^+$ (Hours)
	Number of activated cells (in each row)	Count for the number of activated cells	$\in \mathbb{Z}^+$ (Unitless)
	Number of activated cooling units (in each row)	Count for the number of activated cooling units at arrival	$\in \mathbb{Z}^+$ (Unitless)
	Voltage specs	{0: "HV", 1: "LV", 2: "Both"}	$\in \mathbb{Z}_3$ (Unitless)
	Cooling specs	{0: "Air", 1: "Water/Air"}	$\in \mathbb{Z}_2$ (Unitless)

where R_i is the total reward at time step i, b_{v_i} can be -1 when requirements are matched or 1 otherwise at time time step i, R_{v_i} is the constant value for the reward received upon correct assignment. We set its value to +100. Similarly, we have b_{c_i} and R_{c_i} for cooling matching. During the experiments, we assign 400 orders in each episode. Hence, the maximum reward value is 80,000.

During the evaluation phase, where we assessed the agent over 10 episodes with resets between each episode, it demonstrated 100% accuracy but only 0.121 utilization. This highlights the agent's success in learning the assigned task (compatibility matching) by selectively focusing on a few cells that meet all the necessary criteria. The utilization of cells is depicted in Figure 5, revealing that only a small subset of cells is actively used, while the rest remain untouched by the agent. Notably, the agent incurs no penalty as long as these used cells meet the requirements, even if servers are queued up awaiting testing on the same cell. This observation prompted us to experiment with waiting time penalization in the design of the second reward function.

5.2 Reward Function 2

The improved reward function, as illustrated in Equation 3, imposes a penalty on the agent proportional to the time a server spends in the queue before undergoing testing. Our experimentation involved testing three distinct weighting factors, denoted as ζ: 1, 5, and 10. The weighting factor represents the constant by which the number of hours a server spends in the queue is multiplied. Figure 6 shows the reward, accuracy (i.e., the count of compatible assignments over the total number of assignments), and utilization progress during the training course.

$$R_i = (b_{v_i} \times R_{v_i}) + (b_{c_i} \times R_{c_i}) - (\zeta \times t_i) \tag{3}$$

where ζ is a weighting factor and t_i is time spent by server i in the queue before the test initiation.

In contrast to the previous scenario where only one component was considered, the inclusion of two components introduces a notable distinction between the behavior of the reward function value and accuracy. The accuracy metric in this context is defined to quantify the rate of compatibility as explained in Section 5.3. With the agent striving to strike a balance between these components during training, it tends to prioritize factors with higher weights. Notably, as in Figure 6 for $\zeta = 1$, the agent exhibits a behavior where it places greater emphasis on compatibility matching while also allocating some attention to the time factor. In the initial stages of the training phase, the hindrance imposed by the reward function prevents the agent from fully focusing on compatibility, which is reflected in the decrease of the reward value when the utilization decreases. This impediment compels the agent to more effectively distribute weights across test cells, ultimately reducing time spent in the queue and enhancing overall utilization.

Conversely, the experiments reveal that employing $\zeta = 10$ does not enhance the agent's performance when compared to $\zeta = 5$ in terms of utilization. The former achieves a utilization value of 0.411, while the latter achieves a value of 0.425 during evaluation. Both values significantly surpass the scenario with no time penalty, which yields a utilization of 0.121. Opting for $\zeta = 1$ results in a slightly lower utilization of 0.392. This prompts consideration that there may exist an optimal value for ζ that maximizes utilization. However, we intend to further explore this topic in future work. Notably, the overall utilization rate of 0.425 achieved when setting ζ to 5 matches the upper-bound achieved by the FCFS baseline which has been recorded at 0.424. This, in turn, reveals the efficiency of the learned policy by the RL agent.

In the beginning, we hypothesized that a greater time penalty would result in improved load distribution, consequently leading to higher average utilization. Figure 7 illustrates the utilization rates for the three ζ values, which reinforce this hypothesis but up to a specific point, beyond which the performance of the agent starts to degrade in terms of both utilization and accuracy. Notably, for the third scenario, where $\zeta = 10$, we notice that the agent might require more training than 2000 episodes to achieve perfect matching between servers and assigned cells. The large time penalty associated with the reward function prevents the

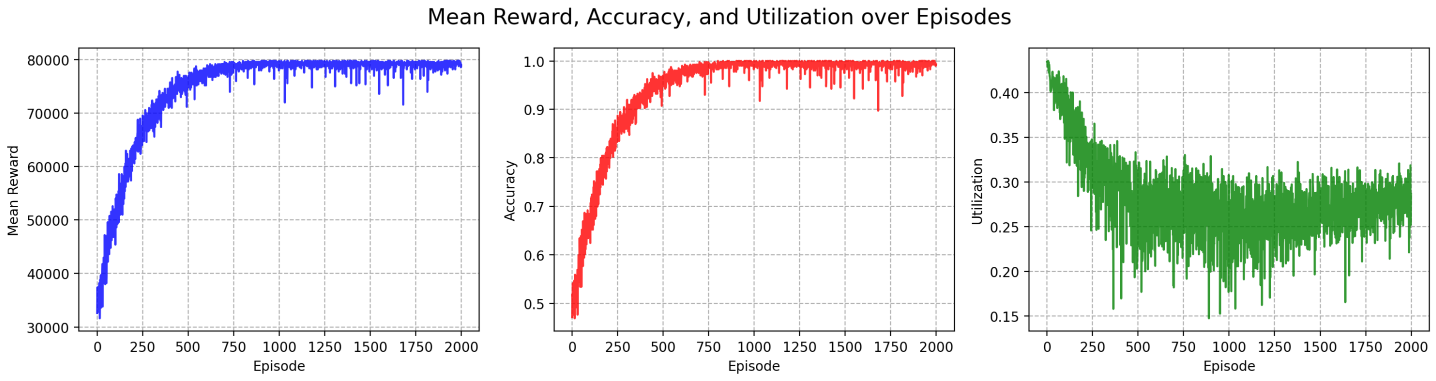


FIGURE 4: REWARD-ACCURACY-UTILIZATION PROGRESS DURING TRAINING WITHOUT TIME PENALTY (REWARD FUNCTION 1).

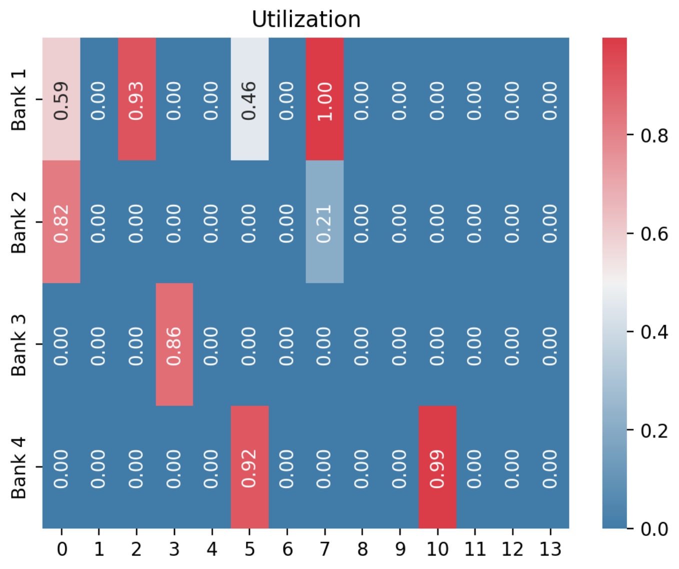


FIGURE 5: UTILIZATION OF TEST CELLS DURING EVALUATION. NO TIME PENALTY IS IMPOSED DURING TRAINING (REWARD FUNCTION 1).

agent from achieving positive reward values. The incoming server needs to wait in a queue for only 20 hours to make the agent get a negative reward even in the case of a compatible assignment.

Figure 7 depicts the utilization rates for the three ζ values, supporting this hypothesis up to a certain threshold. Beyond this point, the agent's performance diminishes in terms of both utilization and accuracy. Notably, in the third scenario where $\zeta = 10$, it becomes apparent that the agent may require more than 2000 episodes of training to achieve perfect matching between servers and assigned cells. This is in contrast to the first and second cases, where less than 1500 episodes proved sufficient to attain 100% accuracy. The substantial time penalty within the reward function hinders the agent from attaining positive reward values, and even a 20-hour wait in a queue can lead to a negative reward, even in cases of compatible assignments.

Conversely, a significant enhancement in load distribution is observed when incorporating a time penalty into the reward function. In Figure 7, the utilized cells are depicted in a more reddish hue compared to the bluish cells. This stark contrast highlights the notable difference in load distribution compared to Figure 5, where only 9 cells out of 56 are utilized.

To better understand the spikes in Figure 6, we start by depicting the exploration decay performed during the training process. Figure 8 presents the ϵ decay during the course of training. This decay takes place according to Equation 4. From Figure 8, we notice that under the chosen conditions, the decay stops at episode 919 and the exploration factor stabilizes at 0.01. Hence, we study the phase of training that takes place after the episode 919.

$$\epsilon_{new} = \max(\epsilon_{min}, \epsilon \times decay\ factor) \tag{4}$$

where ϵ_{new} is the exploration rate for the subsequent episode, ϵ is the current exploration rate, and ϵ_{min} is the predefined minimum exploration rate. In our experiments, we set the decay factor to 0.995 and ϵ_{min} to 0.01.

We have retrieved one of the training episodes where the compatibility rate (accuracy) fell below 0.5 to study the order assignments. Figure 9 shows these assignments. We notice that the assignment spanned only nine testing cells. The remaining testing cells where not utilized at all. Moreover, three out of these nine cells to almost 300 out of the 400 orders, meanwhile, the remaining six cells handle merely about 25% of the orders. This,

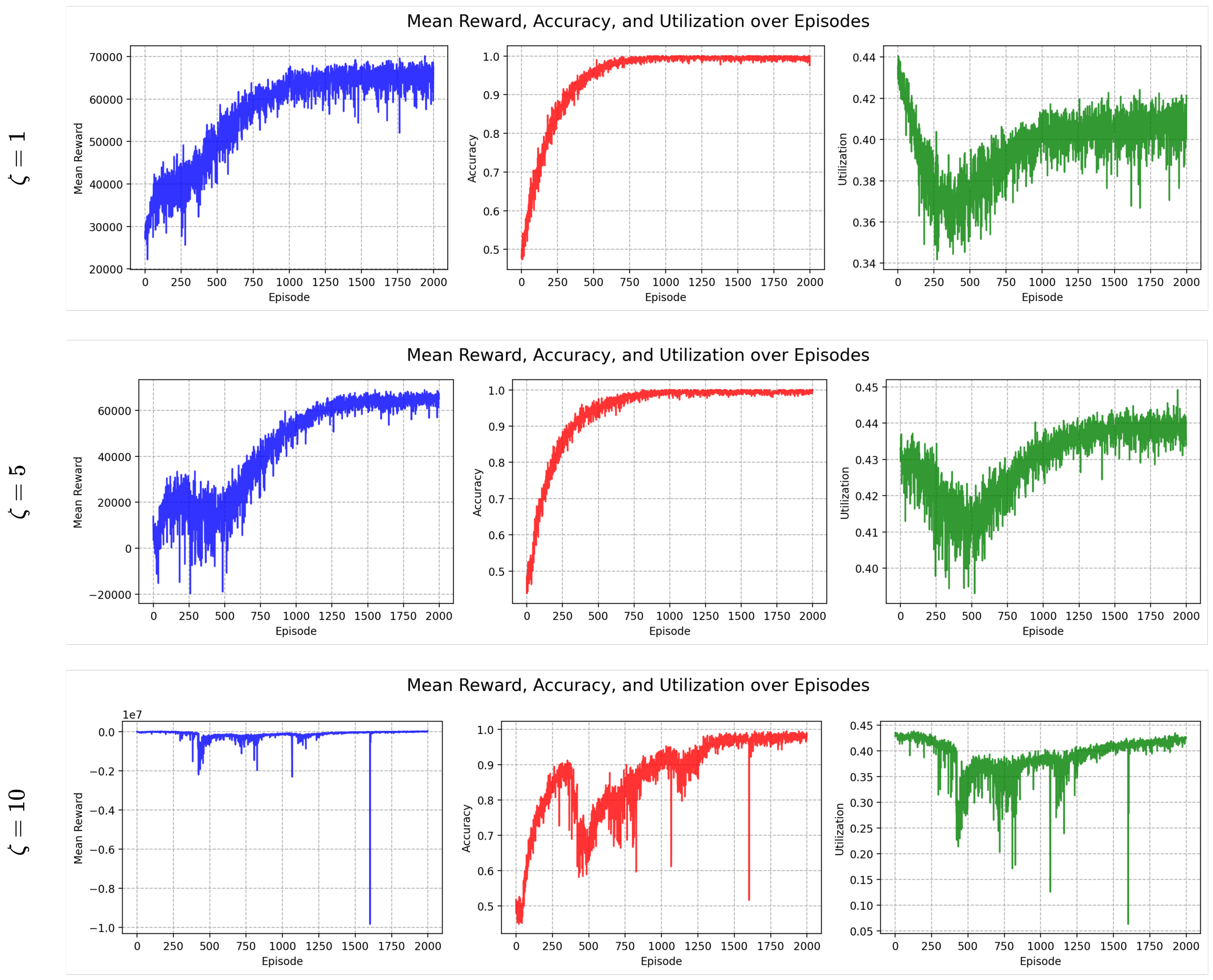


FIGURE 6: REWARD-ACCURACY-UTILIZATION PROGRESS DURING TRAINING WITH TIME PENALTY (REWARD FUNCTION 2).

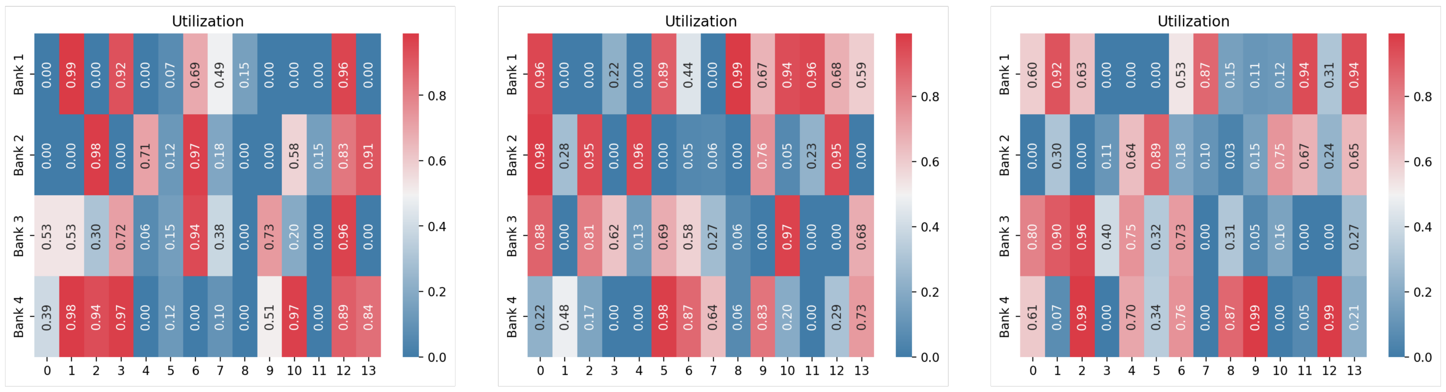


FIGURE 7: CELL UTILIZATION DURING EVALUATION WITH TIME PENALTY (REWARD FUNCTION 2). ζ = 1 (LEFT), ζ = 5 (MIDDLE), AND ζ = 10 (RIGHT).

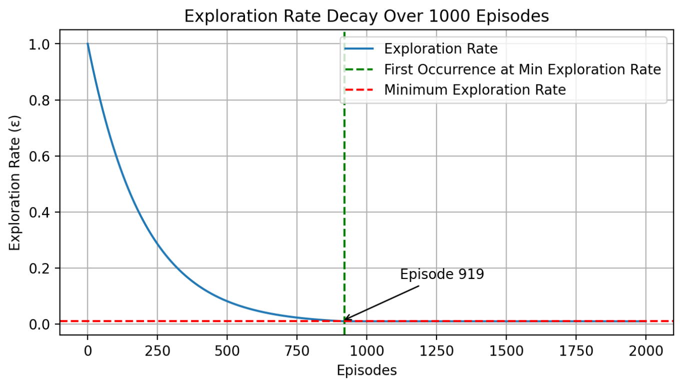


FIGURE 8: EXPLORATION RATE (ϵ) DECAY DURING TRAINING.

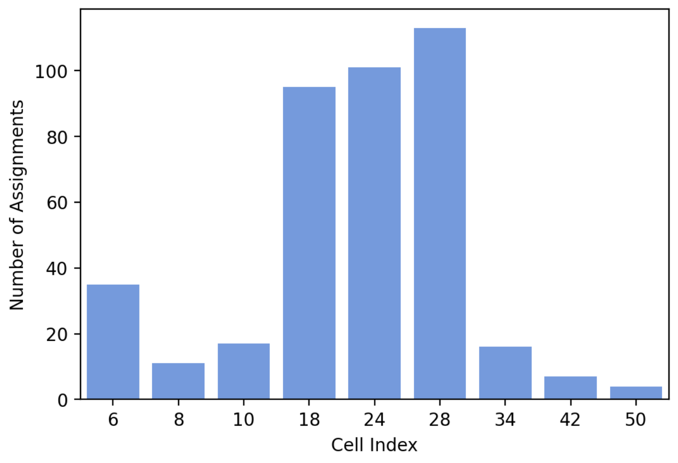


FIGURE 9: CELL ASSIGNMENTS DURING AN EPISODE WHERE ACCURACY FELL BELOW 50%.

in turn, explains the low utilization rate spikes in Figure 7. Table 2 shows the specifications of the assigned cells.

5.3 Evaluation Metrics

In this work, we monitored two evaluation metrics in addition to the reward function: accuracy and the utilization rate. Accuracy in this context refers to the rate of compatible assignments, where the model successfully assigns incoming servers to testing cells that meet specific compatibility requirements. These requirements are based on two features: voltage and cooling needs. An incorrect assignment in either feature results in a zero accuracy score for that action. Conversely, fulfilling both requirements yields a 100% accuracy score for the action. The overall accuracy is computed throughout the training/evaluation episode. The utilization rate, on the other hand, indicates the proportion of time during the total episode that a cell is actively testing a server. This rate is averaged across all testing cells to produce the second metric in our evaluation.

5.4 Upper Bound Solution

To evaluate the proposed RL, we compare its performance against an upper bound rule-based scheduling approach for the current formulation. The current formulation of the problem treats the inputs as unchangeable conditions. In other words, upon the arrival of a server, it gets assigned based on its characteristics and the environment state considering previously assigned servers as fixed inputs that cannot be preceded. Hence, the minimum time that can be spent in the system given this formulation can be secured by assigning the server to the compatible cell that will become soonest available. This is the basis of our upper bound solution to which we compare the performance of the proposed RL method.

TABLE 2: REWARD FUNCTION 1 ASSIGNED CELL SPECIFICATIONS.

Cell Index	Voltage/Cooling Specs
6	High/Air
8	High/Water
10	Hybrid/Water
18	Hybrid/Water
24	High/Water
28	High/Water
34	Hybrid/Water
42	High/Water
50	Low/Air

6. CONCLUSION

In this study, a *DRL* approach was used to assign test cells to incoming servers within a simulated model of a computer-server fulfillment environment. The DQN algorithm was employed to train the *DRL* agent, and two reward functions were examined during experimentation. The first reward was solely based on compatibility, while the second introduced a waiting time penalty. The latter exhibited superior performance, achieving perfect assignment (100% compatibility) and an increased utilization of 0.425 compared to 0.121 in the former case. Notably, the weighting factor applied to the waiting time demonstrated a significant impact on the agent's performance. A weighting factor of 5 outperformed both 1 and 10, prompting further exploration of the optimal point in future investigations. Additionally, our future work aims to enhance the realism of the server environment simulation to maximize the impact of the findings for real-world applications.

7. LIMITATIONS

Although this work has employed an advanced *DRL* approach that offers several benefits over traditional techniques, there are still some limitations. These limitations stem from inherent challenges in RL training, including the agent's specialization in solving the task that it was trained on. RL agents, in general, are trained on specific environments and hence any changes to the environment configuration, such as the number of testing cells or rows, would require building a new simulation model for the environment and training the agent on that environment. Moreover, the second reward function with a high weighting factor of 10 for the time delay yields compatibility mismatching. This, in turn, needs to be handled either by finding a more optimized weighting factor, allowing for longer training, or applying masking to filter out invalid cells. The latter has higher potential for resolving the problem and creates more space for the model to focus on other objectives.

On the other hand, to simplify the task, certain simplifications were made while building the simulation model. These include assuming a single server type and, consequently, a single processing time distribution. In the real-worl scenario, however, there are various types of servers, each with its own processing time distribution. Enhancements to the simulation model to address these variations are planned for future work.

8. FUTURE WORK

The proposed DRL framework has the potential to enhance test efficiency in server manufacturing systems. By performing dynamic scheduling of servers to testing cells, the framework ensures increased utilization of these cells. This study represents the initial phase of our research that focuses on enabling production scheduling that minimizes energy consumption within the server manufacturing system while improving the test cell occupance. Future work will incorporate these objectives in a manner that ensures a reduction in energy consumption without violating any constraints, such as compatibility and deadlines. Consequently, the proposed framework will offer a solution applicable to real-world scenarios in server manufacturing and similar industries while enabling dynamic, near-optimal, and near-real-time scheduling under uncertainty. This current work forms the first step toward achieving that goal.

ACKNOWLEDGEMENT

We gratefully acknowledge the financial support of the National Science Foundation (NSF) under grant CMMI #2038325. Any opinions, findings, or conclusions found in this paper are those of the authors and do not necessarily reflect the views of the sponsor.

REFERENCES

[1] Aqlan, Faisal, Lam, Sarah S and Ramakrishnan, Sreekanth. "An integrated simulation–optimization study for consolidating production lines in a configure-to-order production environment." *International Journal of Production Economics* Vol. 148 (2014): pp. 51–61.

[2] Waubert de Puiseau, Constantin, Meyes, Richard and Meisen, Tobias. "On reliability of reinforcement learning based production scheduling systems: a comparative survey." *Journal of Intelligent Manufacturing* Vol. 33 No. 4 (2022): pp. 911–927.

[3] Marte, Rafael et al. *Handbook of heuristics*. Springer, (2018).

[4] Lenstra, Jan Karel, Kan, AHG Rinnooy and Brucker, Peter. "Complexity of machine scheduling problems." *Annals of discrete mathematics*. Vol. 1. Elsevier (1977): pp. 343–362.

[5] Pinedo, Michael and Hadavi, Khosrow. "Scheduling: theory, algorithms and systems development." *Operations Research Proceedings 1991: Papers of the 20th Annual Meeting/Vorträge der 20. Jahrestagung*: pp. 35–42. 1992. Springer.

[6] Gershwin, Stanley B. "The future of manufacturing systems engineering." *International Journal of Production Research* Vol. 56 No. 1-2 (2018): pp. 224–237.

[7] Lang, Sebastian, Schenk, Michael and Reggelin, Tobias. "Towards Learning-and Knowledge-Based Methods of Artificial Intelligence for Short-Term Operative Planning Tasks in Production and Logistics: Research Idea and Framework." *IFAC-PapersOnLine* Vol. 52 No. 13 (2019): pp. 2716–2721.

[8] Rossit, Daniel Alejandro, Tohmé, Fernando and Frutos, Mariano. "A data-driven scheduling approach to smart manufacturing." *Journal of Industrial Information Integration* Vol. 15 (2019): pp. 69–79.

[9] Priore, Paolo, Gomez, Alberto, Pino, Raúl and Rosillo, Rafael. "Dynamic scheduling of manufacturing systems using machine learning: An updated review." *Ai Edam* Vol. 28 No. 1 (2014): pp. 83–97.

[10] Liu, Juan, Qiao, Fei and Ma, Yumin. "Real time production scheduling based on Asynchronous Advanced Actor Critic and composite dispatching rule." *2020 Chinese Automation Congress (CAC)*: pp. 7380–7383. 2020. IEEE.

[11] Badia, Adrià Puigdomènech, Piot, Bilal, Kapturowski, Steven, Sprechmann, Pablo, Vitvitskyi, Alex, Guo, Zhaohan Daniel and Blundell, Charles. "Agent57: Outperforming the atari human benchmark." *International conference on machine learning*: pp. 507–517. 2020. PMLR.

[12] Vinyals, Oriol, Babuschkin, Igor, Czarnecki, Wojciech M, Mathieu, Michaël, Dudzik, Andrew, Chung, Junyoung, Choi, David H, Powell, Richard, Ewalds, Timo, Georgiev, Petko et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning." *Nature* Vol. 575 No. 7782 (2019): pp. 350–354.

[13] Zhang, Wei and Dietterich, Thomas G. "A reinforcement learning approach to job-shop scheduling." *International Joint Conference on Artificial Intelligence*, Vol. 95: pp. 1114–1120. 1995.

[14] Zweben, Monte, Davis, Eugene, Daun, Brian and Deale, Michael J. "Scheduling and rescheduling with iterative repair." *IEEE Transactions on Systems, Man, and Cybernetics* Vol. 23 No. 6 (1993): pp. 1588–1596.

[15] Riedmiller, Simone and Riedmiller, Martin. "A neural reinforcement learning approach to learn local dispatching policies in production scheduling." *International Joint Conference on Artificial Intelligence*, Vol. 2: pp. 764–771. 1999.

[16] Gabel, Thomas and Riedmiller, Martin. "Adaptive reactive job-shop scheduling with reinforcement learning agents." *International Journal of Information Technology and Intelligent Computing* Vol. 24 No. 4 (2008): pp. 14–18.

[17] Lin, Chun-Cheng, Deng, Der-Jiunn, Chih, Yen-Ling and Chiu, Hsin-Ting. "Smart manufacturing scheduling with edge computing using multiclass deep Q network." *IEEE Transactions on Industrial Informatics* Vol. 15 No. 7 (2019): pp. 4276–4284.

[18] Liu, Chien-Liang, Chang, Chuan-Chin and Tseng, Chun-Jan. "Actor-critic deep reinforcement learning for solving job shop scheduling problems." *Ieee Access* Vol. 8 (2020): pp. 71752–71762.

[19] Baer, Schirin, Bakakeu, Jupiter, Meyes, Richard and Meisen, Tobias. "Multi-agent reinforcement learning for job shop scheduling in flexible manufacturing systems." *2019 Second International Conference on Artificial Intelligence for Industries (AI4I)*: pp. 22–25. 2019. IEEE.

[20] Sutton, Richard S and Barto, Andrew G. *Reinforcement learning: An introduction*. MIT press (2018).

[21] Zhou, MengChu. *Petri nets in flexible and agile automation*. Vol. 310. Springer Science & Business Media (2012).

[22] Baer, Schirin, Turner, Danielle, Mohanty, Punit, Samsonov, Vladimir, Bakakeu, Romuald and Meisen, Tobias. "Multi agent deep q-network approach for online job shop scheduling in flexible manufacturing." *Proc. ICMSMM*: pp. 78–86. 2020.

[23] Hubbs, Christian D, Li, Can, Sahinidis, Nikolaos V, Grossmann, Ignacio E and Wassick, John M. "A deep reinforcement learning approach for chemical production scheduling." *Computers & Chemical Engineering* Vol. 141 (2020): p. 106982.

[24] Zhu, Huayu, Li, Mengrong, Tang, Yong and Sun, Yanfei. "A deep-reinforcement-learning-based optimization approach for real-time scheduling in cloud manufacturing." *IEEE Access* Vol. 8 (2020): pp. 9987–9997.

[25] Zhou, Longfei, Zhang, Lin, Ren, Lei and Wang, Jian. "Real-time scheduling of cloud manufacturing services based on dynamic data-driven simulation." *IEEE Transactions on Industrial Informatics* Vol. 15 No. 9 (2019): pp. 5042–5051.

[26] Deb, Kalyanmoy, Pratap, Amrit, Agarwal, Sameer and Meyarivan, TAMT. "A fast and elitist multiobjective genetic algorithm: NSGA-II." *IEEE transactions on evolutionary computation* Vol. 6 No. 2 (2002): pp. 182–197.

[27] Jian, CF and Wang, Y. "Batch task scheduling-oriented optimization modelling and simulation in cloud manufacturing." *International Journal of Simulation Modelling* Vol. 13 No. 1 (2014): pp. 93–101.

[28] Semchedine, Fouzi, Bouallouche-Medjkoune, Louiza and Aissani, Djamil. "Task assignment policies in distributed server systems: A survey." *Journal of network and Computer Applications* Vol. 34 No. 4 (2011): pp. 1123–1130.

[29] Semchedine, Fouzi, Bouallouche-Medjkoune, Louiza and Aissani, Djamil. "Improving the performance for task assignment in distributed server systems by partitioning the large tasks." *International Journal of Computer Mathematics* Vol. 92 No. 2 (2015): pp. 250–265.

[30] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg et al. "Human-level control through deep reinforcement learning." *nature* Vol. 518 No. 7540 (2015): pp. 529–533.