Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

YIELD IMPROVEMENT USING DEEP REINFORCEMENT LEARNING FOR DISPATCH RULE TUNING

David Norman[1], Prafulla Dawadi[1], and Harel Yedidsion[1]

[1]AI/ML Team, Automation Products Group,
Applied Materials, 3050 Bowers Avenue, Santa Clara, CA 85054, USA

ABSTRACT

In this paper we discuss improving yield in semiconductor manufacturing using reinforcement learning (RL) to tune a dispatching rule parameter to increase the number of lots that process on high-yield equipment. We consider a dispatching rule with a parameter that controls whether or not the rule allows a lot to process on a lower-yield equipment or waits to allow the lot to possibly process later on a high-yield equipment. In a factory such a parameter would be set periodically, e.g., once a week, but RL allows the parameter to be updated frequently, leading to better factory performance. We also consider a novel measure of on-time delivery where the goal is to have 95% on-time delivery in a set of time intervals, e.g., shifts. We show how a trained RL agent using a graph neural network outperforms the baseline by maintaining on-time delivery while processing significantly more lots on high-yield equipment.

1 INTRODUCTION

Dispatching and scheduling in the semiconductor industry are widely studied, both because they provide academically interesting problems and because they are directly applicable in industry. Here we focus on dispatching in the implant area of a semiconductor factory. Implant is characterized by long setup times that make it difficult to estimate the future processing capacity given a set of lots to process. We consider two KPIs for an implant area: on-time delivery and improving yield.

For the manager of an equipment area, on-time delivery is typically the most important KPI (Stöckermann et al. 2023). The area must process the lots quickly enough that they will be able to be shipped on time and that downstream areas will be fed properly.

Yield has long been important at the equipment level, but only recently has it become important at the dispatching level. A factory has data that, for certain process steps, certain equipment will have higher yield than other equipment. These differences between equipment are small: a few percentage points or less, so historically they were not widely considered when making dispatching decisions. However, at leading-edge nodes with their longer and more complex process flows, the value of a single wafer increases, yield becomes more important (Peters 2022). For this reason, it becomes important when making dispatching decisions to try to process lots on equipment that have high yields.

It is important to note that these two KPIs are in conflict. The best way to improve on-time delivery is to process lots as quickly as possible. However, to improve yield it is best to wait to process a lot until a high-yield equipment becomes available. Managing the tradeoff between these two KPIs using deep reinforcement learning and graph neural networks is the subject of this paper.

Graph neural networks (GNNs) are key to our approach since they allow the neural network to handle varying inputs, including varying numbers of lots, varying setups, and varying equipment configurations. Traditional approaches using multi-layer perceptrons (MLP) have fixed-length vectors as input which require workarounds like padding variable-length inputs a fixed length vector, which then forces the MLP to learn the difference between the real input and the padding. The GNN also has the advantage that its

input, the graph, explicitly models the relationships between the lots, equipment, and setups in the factory, see section 5.2, thus making it easier for the agent to learn.

1.1 Reinforcement Learning

Reinforcement learning is one of the branches of machine learning along with supervised learning and unsupervised learning. In reinforcement learning an agent is given observations about an environment. The agent then performs an action which changes the environment, after which the agent is given a reward and new observation. This cycle repeats. The agent is trained to choose actions which maximize the reward it receives.

In deep reinforcement learning the agent is represented by a neural network which maps the observation onto the action to be performed. Deep reinforcement learning has been used successfully in domains such as computer games (Wurman et al. 2022) complex games (Silver et al. 2017), robotics (Hanna and Stone 2017; Park et al. 2020), and control (Cui et al. 2021).

1.2 Dispatching Parameters and Reinforcement Learning

The academic literature typically focuses on simple dispatching rules like shortest processing time, earliest due date, or critical ratio, however the rules that are used in semiconductor factories are usually more complex, since they need to manage multiple competing KPIs. As factory conditions change, e.g., as factory loading increases or decreases, the different KPIs managed by the rule can change, so the rules will typically have parameters that manage the relative importance of the various KPIs. These parameters are periodically set based on current conditions, usually manually based on experience, although sometimes also using a simulation coupled with optimization.

In this paper we explore using reinforcement learning to control such dispatching parameters. In this case the RL agent would take as input the current factory conditions and produce as output the dispatching parameters to be used for those conditions. Such a system not only allows the parameters to be set without having a human expert available, but also allows the parameters to be changed more frequently, allowing the rule to adapt more quickly to factory changes.

2 RELATED WORK

Research using graph-neural networks for dispatching and scheduling is becoming more common, see for example (Hameed and Schwung 2023; Hu et al. 2020; Huang et al. 2023; Liu et al. 2022; Song et al. 2023; Soykan et al. 2023). There is, however, not a consensus about the structure of the graph that is input to the GNN or how the GNN is used in the overall reinforcement learning network. In financial portfolio optimization, (Sun 2024) uses a very similar architecture to ours where the GNN is used as a feature extractor which then feeds a neural network agent.

Most research using RL for scheduling and dispatching optimizes makespan or average cycle time. We consider on-time delivery, a binary value for each lot indicating if the lot processed on time, which is similar to tardiness, a real number for each lot that is positive if the lot is late and zero if it is on time. Various papers consider tardiness, for example (Min and Kim 2022) optimize total weighted tardiness, (Waschneck et al. 2018) minimize total tardiness, and (Tassel et al. 2023) minimize average tardiness. We are not aware of any work that directly considers optimizing on-time delivery percentage or the more complex on-time delivery KPI we discuss below.

Our problem of using RL to tune a parameter of a dispatching rule is novel. Most uses of RL for dispatching and scheduling aim to replace the dispatching rule by having the RL agent directly select the lot that should process next. There are a few other approaches, e.g., (Liu et al. 2020) and (Zhang et al. 2023), that use the RL agent to select among a prespecified set of dispatching rules.

3 SIMULATION MODELS

We use simulation models for the AutoSched™ software to train and evaluate the RL agent. These models are designed to be roughly similar to a single implant equipment group in a medium-sized factory. The models only include the single equipment group, so all routes have a single step.

We model each implant step as having two different types of setups: an energy and a recipe group. Changing an equipment's energy setup is expensive and takes hours, while changing the recipe group setup is much shorter.

Each model is randomly generated using the following parameters:

- Each model has ten implant equipment.
- Each step has a processing time between 10 and 20 minutes.
- Each step has a cycle time target equal to 13 times the processing time. This is used to calculate a due date for each lot.
- 90% of the steps have two randomly-chosen stations with high yield. The remaining steps have one randomly-chosen station with high yield.
- 30% of steps require high energy and the remaining steps require low energy.
- A setup change from high energy to medium energy takes 2 hours.
- A setup change from medium energy to high energy takes 3 hours.
- Each model has between 12 and 16 recipe group setups.
- Each recipe group setup change time takes between 25 and 35 minutes.
- Each model has an average lot arrival rate λ which varies as described in the experiments below.
  - Individual lot arrivals are randomly perturbed so that there is variation in lot arrivals while maintaining the average lot arrival rate.
  - The lot arrival rates are low enough that the on-time delivery goals can be met.
- Initial WIP tuned to match the lot arrival rate.
- Randomly created preventative maintenance operations (PMs) with random duration four, six, or eight hours such that the equipment group is in the PM state a specified percentage π of the time as described in the experiments below.

Our training models include different step processing times and different numbers of recipe groups, which means that the trained agent will be robust to factory changes like new products, recipe changes, etc.

3.1 Dispatching Rule

We control the behavior of the simulation using a dispatching rule that selects the next lot an equipment should process or returns that the station should remain idle. This rule includes a parameter T that is used to control how the rule manages the tradeoff between on-time delivery and processing on high-yield equipment. This is the parameter that will be controlled by the RL agent. We implement the rule using the APF Formatter™ and Fusion™ software which allows us to easily build and integrate a complex rule with the simulation.

The rule includes the following functionality:

- No two equipment may have the same energy and recipe group setups.
- An energy setup change is only done if there are a sufficient number of lots waiting for the new energy setup.
- An energy setup change is not done if there are many lots waiting for the current energy setup.
- The number of stations with high energy setup must remain within one of the target of 30% of the total number of stations.

- The parameter *T* such that, if a lot can be selected to an equipment and that equipment is not high-yield for that lot's step, then the lot will not be selected if the remaining time until the lot's due date is greater than *T*. Note that this can cause equipment to be left idle.
  - This is the parameter that will be tuned by the RL agent.
  - This definition of the parameter implies that large values of *T* mean that the equipment group will process lots as quickly as possible, while small values of *T* mean that equipment group will wait as long as possible to process lots.
- Order the lots based on critical ratio.
- As a tiebreaker, the rule prefers processing lots with earlier due dates and lots whose ID is lexicographically earlier. This makes the rule deterministic.

4 PERFORMACE KPIS

We use two KPIs to evaluate the performance of the baselines and RL agent: on-time delivery over a specified time interval and the percentage of lots that process on high-yield equipment.

We calculate on-time delivery similarly to how an area manager would be measured in a fab. Over a specified time interval, we compute the percentage of lots that finish on time. This percentage must be greater than or equal to 95% for the on-time delivery goal to be met for that interval. We report the percentage of time intervals where the 95% on-time delivery is met.

In a fab, the time interval would typically be one shift or one day. In our case, in order to make the neural network training easier to train we use the interval given by 32 time steps.

The second KPI is yield as measured by the percentage of lots that start processing on one of their step's high-yield equipment divided by the total number of lots that start processing.

There is a tradeoff between these two KPIs: achieving on-time delivery means that the equipment group should process lots as quickly as possible, but increasing yield means that the equipment group should wait to process lots to see if they can process on a high-yield station. Normally competing objectives like this would mean that we would need to perform a delicate weight balancing process to get the sorts of solutions we want. However, the mathematical structure of our on-time delivery KPI makes this unnecessary. The key point is that, for any given time interval, the on-time delivery is a binary value: either the equipment group achieves the goal or it fails. Furthermore, we assume the factories we are modeling are well-run, so that on-time delivery is achievable. This means that the weights given to the two KPIs do not matter as long as the on-time delivery weight is sufficiently large. Conceptually, the RL agent needs to discover the region where on-time delivery is achieved, then, within that region, figure out how to maximize yield.

5 SOLUTION METHOD

5.1 RL Training and Policy Updates

For RL agent training we use the Proximal Policy Optimization (PPO) algorithm (Schulman et al. 2017), an actor-critic deep RL algorithm that uses on-policy learning to train a stochastic policy using gradient descent. We use the implementation from Stable Baselines3 (https://github.com/DLR-RM/stable-baselines3).

Stable Baselines3 requires that the environment be implemented as a Gymnasium (https://github.com/Farama-Foundation/Gymnasium) environment. We do this using a python wrapper for the AutoSched™ software. This wrapper allows individual values to be passed from the wrapper to a Fusion™ dispatching rule in the model which we use to implement the actions. It also allows the wrapper to run Fusion™ reports which we use to get the simulation information used to construct the observation.

5.2 State Observation

The state observation consists of two components. The first is a vector containing:

- The number of WIP lots,
- The number of lots that will arrive in the next two hours,
- Information about the on-time delivery state of the area, i.e., the number of lots ahead or behind the on-time delivery goal of 95%.
- The number of time steps until the on-time delivery reward will be reported.

The second component of the observation is a graph describing the state of the lots and equipment in the area. The graph is heterogeneous, i.e., it contains different types of nodes and edges. The graph has a node for each lot, either in current WIP or that will arrive in the next two hours, a node for each equipment, and a node for each (energy setup, recipe group setup) that is not currently on an equipment. The lot nodes have attributes giving the processing time for the lot and number of hours until the lot is due. The station nodes have attributes giving the number of hours until the station will be available to process another lot and the number of hours until the next PM.

The graph includes the following types of bidirectional edges, including their types as in Figure 1:

- SameSetup: An edge between each lot and equipment where the equipment has the correct setup for the lot.
  - These edges have a binary attribute indicating if the equipment is high-yield for the lot's step.
- HighYield: An edge between each lot and each equipment where the equipment is high-yield for the lot's step and where there are no other edges between the lot and equipment.
- LotSetup: An edge between each lot and setup node where the lot's step requires the setup represented by the node. These edges are only created if no equipment has the setup required by the lot's step.
- SameEnergySetup: An edge between each setup node and each equipment where the setup has the same energy setup as the current energy setup of the equipment.

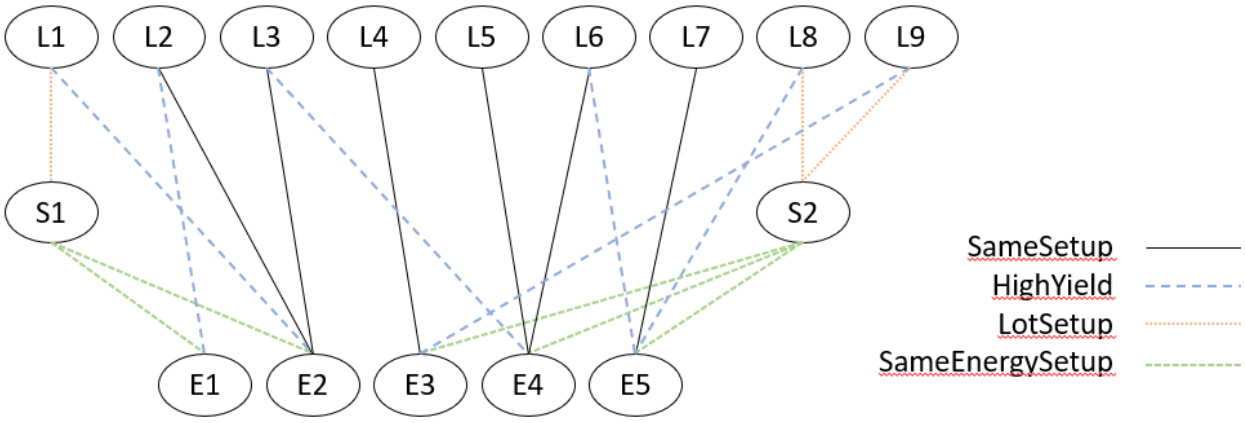

Figure 1: Example observation graph.

Figure 1 has an example observation graph. Lots L2, L3, L4, L5, L6, and L7 have an equipment with the required setup, so they have edges to the appropriate equipment. Lots L1, L8, and L9 do not have an equipment with their step's required setup, so they have edges to setup nodes representing their required setups. Lots L1, L2, L3, L6, L8, and L9 have high-yield equipment that does not have the correct setup so they have an edge to those equipment. S1 has edges to the two equipment with the same energy setup as

S1, meaning that a short recipe group setup change time is required to switch to S1 and thus be able to process L1.

Our motivation for the graph structure we have chosen is as follows: Most of the lots that will process in the near future will process on equipment with their current setup, so we include SameSetup edges to encode this information. The more interesting case is equipment that will change setup. The GNN will need to predict the new setup, which is most likely to be a setup which is not currently on any other equipment. Predicting the new setup requires information about the lots requiring that setup, e.g., the rule prefers converting an equipment to a setup where the lots that will use the setup have high yield on that equipment. For that reason, we introduce Setup nodes to connect all lots that require a setup. The GNN can use the Setup node to summarize information about all the lots with that setup and then pass it to the Equipment nodes which will predict the next setup. Since the new setup is likely to involve a setup that just changes the recipe group and not the energy and also has many lots where the equipment is high-yield, we also include the SameEnergySetup and HighYield edges.

5.3 Action Space

The action space is a one-dimensional continuous space giving the value for the dispatching parameter T described in section 3.1. We normalize the action space so that the output of the neural network is in the interval $[-1,1]$.

5.4 Reward Function

The reward function is given by:

$$R_i = \mathrm{W}_{\mathrm{yield}} \frac{P_i}{N_i} + W_{otd} \begin{cases} 0 \; if \; (i+1 \; mod \; 32) \; \neq 0, \\ 0 \; if \; (i+1 \; mod \; 32) = 0 \; and \; OTD \; goal \; met, \\ -1 \; if \; (i+1 \; mod \; 32) = 0 \; and \; OTD \; goal \; not \; met, \end{cases}$$

where R_i is the reward for time step i, W_{yield} is the weight for the yield portion of the reward, P_i is the number of lots that start processing on high-yield equipment during the step, N_i is the total number of lots that start processing during the step, and W_{otd} is the weight for the on-time delivery portion of the reward. The constant 32 is the number of steps in the time intervals over which on-time delivery is calculated as discussed in section 4. The on-time delivery reward is zero except at the end of a time interval where the on-time delivery goal was not met, in which case it is -1.

It is worth noting that there is a delay between the actions that affect the on-time delivery and when the reward is reported to the agent. Such delayed rewards can cause agent training to be more difficult.

As discussed in section 4, the weights W_{yield} and W_{otd} should not affect the final training result, however they do have some effect on training speed. Experimentally we find that $2000 \, W_{yield} = W_{otd}$. We scale the reward returned to PPO to the interval $[-1,1]$ so the exact values are arbitrary.

The reward function exactly matches the KPIs that we are expecting the agent to improve. In particular, we do not need to create any additional rewards to improve agent training, a process often referred to as reward engineering.

5.5 Neural Network

As mentioned above, our observation consists of a graph representing the state of the lots and equipment and an additional vector giving global information about the simulation and KPIs. In order to convert the graph information into a vector that can be fed to PPOs multi-layer perceptron (MLP) network, we use the concept of graph embedding (Goyal and Ferrara 2018) whose objective is to create a function that takes a graph as input and outputs an *n*-dimensional vector, i.e., it embeds the graph in *n*-dimensional space for some *n*. This can also be viewed as creating *n* features that represent the graph.

We use graph neural networks based on message passing layers (You 2020) to perform this embedding. The output of the message passing layers is a vector for each node in the graph, i.e., a node embedding. To convert the node embedding to a graph embedding we aggregate all the nodes of each type, then concatenate the resulting three vectors to give a single graph embedding vector. To be precise, if $G = (n^L, n^E, n^S, E)$ is the graph with lot nodes, equipment nodes, setup nodes, and edges, respectively, then applying the GNN to a graph gives:

$$GNN(G) = GNN\left(n_i^L, n_j^E, n_k^S, E\right) = \left(e_i^L, e_j^E, e_k^S\right)$$

Where the right-hand side are *n*-dimensional embeddings of the lot, equipment, and setup nodes, respectively. The graph embedding e^G can be written as

$$e^G = \langle \textstyle\sum_i e_i^L \,|\, \sum_j e_j^E \,|\, \sum_k e_k^S \rangle,$$

where $\langle \cdot \mid \cdot \rangle$ denotes vector concatenation.

We then concatenate the vector e^G with the vector portion of the observation to get the vector which is the input to the usual MLPs in the PPO agent. This architecture allows us to train the graph embedding and the PPO networks simultaneously. See Figure 2 for a complete diagram.

The GNN also includes two-layer MLP networks to pre-process the lot and station features as described in (You et al. 2020). Experimentally these pre-processing MLPs improve performance significantly. We implement the network using torch-geometric (Fey and Lenssen 2019).

PPO is an actor-critic method, so it has a neural network for the policy and a network that predicts the value of a given state. Experimentally, it is better for our problem to have separate GNNs for the policy and value networks, i.e., it is better to have separate feature extraction for the policy and value networks. This means that we have a separate instance of Figure 2 for the policy and the value networks.

5.6 Training

During training we start the simulation with a three-hour warmup period followed by 256 time steps of 30 minutes. The RL agent updates the dispatching parameter T at each time step. We use a stepped learning rate decay with an initial learning rate of 1e-3 and a final rate of 1e-5 with three steps between. Other PPO parameters are provided in appendix A. The agent trained on models without PMs was trained for approximately 800,000 time steps and the agent trained on models with PMs was trained for approximately 1,770,000 timesteps.

6 EVALUATION

6.1 Baseline Agent

We want to compare the RL agents to a baseline that is similar to a factory where the dispatching parameter is manually set periodically, e.g., once a week, based on factory conditions. For example, the factory employees that manage an implant equipment group might meet once a week and consider the current product mix, factory loading, upcoming PMs, and other factory data and determine dispatching parameters for the next week.

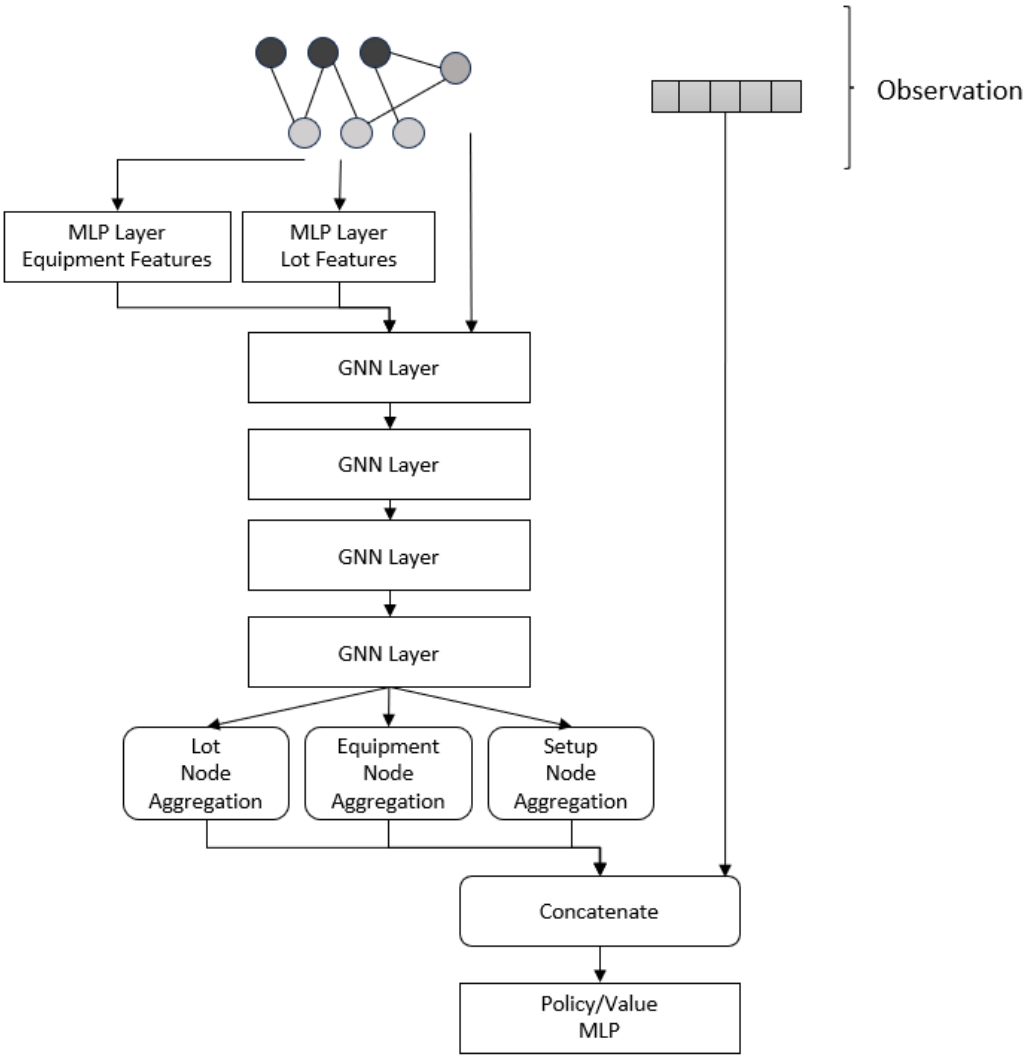

Figure 2: Policy or value neural network.

To create a baseline similar to this factory behavior, we generate baseline simulation models for sets of (λ, π), where λ is the lot arrival rate and π is the percentage of time the equipment group is in the PM state. The models are generated as described in section 3. Each group of simulation models for fixed (λ, π) corresponds to a fixed factory state. For each of these groups we find the value of the dispatching parameter T that results in 100% of the time intervals meeting the on-time delivery goal and that maximizes the average percentage of lots that process on their step's high-yield equipment across all simulation models in the group. This optimization is a straightforward optimization of a one-dimensional, monotonic function with some noise. The optimal value for T corresponds to the factory employees picking the dispatching parameter based on the fab conditions corresponding the models in the group. We repeat the optimization for each group of models with the same (λ, π). The result of this process is a map from (λ, π) to an optimal fixed value of the dispatching parameter. We use this fixed mapping as our baseline agent.

6.2 RL Agent

The RL agent is evaluated by running the evaluation models with the RL agent updating the dispatching parameter every 30 minutes of simulated time. We run the models for 256 steps of 30 minutes after a three-hour warmup.

7 RESULTS

7.1 Without PMs

We first consider the case without PMs. We experimentally determined the equipment group is close to fully loaded when the lot arrival rate is about 23 lots/hour. Based on this, we generated 500 training models with the lot arrival rate λ selected from a uniform distribution on the interval $[21, 23]$ and PM percentage π equal to zero. For evaluation models we use $\lambda \in \{21, 22, 23\}$ with 50 models for each value, giving a total of 150 evaluation models. The baseline is calculated as described in section 6.1 and the agent is trained as in section 5.6.

The evaluation results are in Table 1. Note that the RL agent outperforms the baseline on percentage of lots that process on high-yield equipment and is essentially equivalent in on-time delivery. A paired t-test with one sample for each model shows that the outperformance is significant at the 99.9% level. Note that the baseline does not achieve 100% on-time delivery because the evaluation models are different than the models used to find the baseline value for the dispatching parameter.

Table 1: Comparison of RL agent with the baseline using models without PMs.

	Baseline Agent	RL Agent
Pct of Intervals Meeting OTD	99.5	99.1
Pct of Lots Processing on High-Yield Equipment	38.9	41.5
Average Action	3.40	3.28

The table also includes information about the average action, i.e., the average dispatching parameter value *T*, taken by the agent and baseline across all models and time steps. The values are quite different, indicating that the agent has not simply converged to a policy similar to the baseline. We see this further in Figure 3 which has example distributions of the actions of the RL agent for two randomly-selected simulation models.

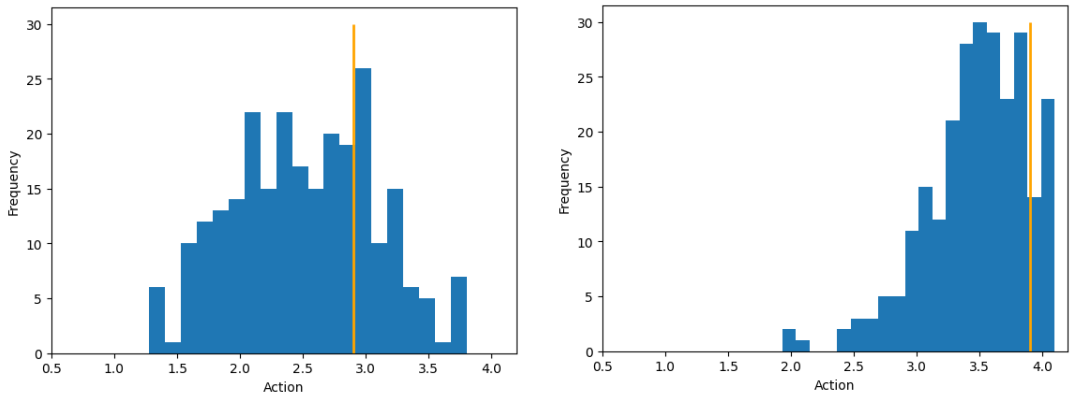

Figure 3: Action distributions for the RL agent for two example simulation models. The vertical line is the constant action for the baseline agent for the model.

7.2 With PMs

We hypothesize that the RL agent outperforms the baseline because the agent can react quickly to changing factory conditions whereas the baseline remains static. To test this, we consider the case where there are PMs that disrupt the normal operation of the equipment group. If the RL agent is outperforming because it reacts more quickly, we would expect its outperformance to increase as the number of disruptions increases, i.e., as the number of PMs increase.

Before generating the training models, we determined experimentally that for PM percentage π equal to 8, 10, and 12, the equipment group is fully loaded if the lot arrival rate λ is equal to 13.6, 11.3, 8.2, respectively. Based on this, we generated 500 training models with the PM percentage π drawn from a uniform distribution on $[8, 12]$ and λ chosen by interpolating the experimentally determined values above. The evaluation models have values $\pi \in \{8, 10, 12\}$ and corresponding $\lambda \in \{13.6, 11.3, 8.2\}$. We generate 50 training models for each (π, λ), giving 150 evaluation models. The baseline is calculated as described in section 6.1 and the agent is trained as in section 5.6.

Table 2 gives the results of the baseline and RL agent on the evaluation set of models. Notice that as the percentage of time spent in the PM state increases from 0.08 to 0.10 to 0.12 the agent's percentage of lots processing on high-yield equipment outperforms the baseline by 18%, 47%, and 52%, which is consistent with our hypothesis.

To test statistical significance, we used a paired t-test with one sample for each of the 150 models. The results indicate that the agent's superiority is statistically significant with a confidence level of 99.9%.

Table 2: Comparison of RL agent against the baseline using models with PMs.

PM%	Lot Arrival Rate (lots/hour)	Pct of Intervals meeting OTD		Pct of Lots Processing on High-Yield Eqp	
		Baseline	RL Agent	Baseline	RL Agent
8	13.6	100	99.8	33.3	39.3
10	11.3	100	100	29.9	44.0
12	8.2	100	99.2	34.7	52.6

8 CONCLUSIONS

In this paper, we have demonstrated RL agents that outperform a baseline consisting of a constant action. We also offer evidence that the RL agents outperform the baseline because they are able to change the dispatching parameter *T* in response to changing factory conditions. The training and evaluation models include varying processing times and varying numbers implant recipe groups, showing that the RL agent preforms well even when the factory product mix or equipment recipes change. We use the percentage of lots that process on higy-yield equipment and a novel on-time delivery KPI to train and evaluate the agents.

We also introduce a measure of on-time delivery based on the performance of the equipment group on specified time intervals and show that, even though the RL agent is only given a reward at the end of each interval, i.e., the reward is significantly delayed, the agent is still able to learn to meet the on-time delivery goal.

We also discuss how to integrate a GNN into the PPO RL algorithm. The GNN allows the RL agent to directly handle different numbers of lots and implant setups. This avoids workarounds like padding that are needed if a MLP network is used directly.

A PPO PARAMETERS

Table 3: PPO Parameters and Values.

Parameter Name	Value

batch_size	2048
n_steps	2048
Gamma	0.99
gae_lambda	0.99
pi net_arch	64,64,8
vf net_arch	128,128,8
activation_fn	ReLU

REFERENCES

Cui, J., W. Macke, H. Yedidsion, A. Goyal, D. Urieli, and P. Stone. 2021. "Scalable Multiagent Driving Policies for Reducing Traffic Congestion". In *Proceedings of the 20th International Conference on Autonomous Agents and Multi Agent Systems*, Virtual Event United Kingdom, May 3 - 7, 2021, 386-394.

Fey, M. and J. Lenssen. 2019. "Fast Graph Representation Learning with PyTorch Geometric*." ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Goyal, P. and E. Ferrara. 2018. "Graph Embedding Techniques, Applications, and Performance: A Survey." *Knowledge-Based Systems* 151 (July): 78–94.

Hameed, M. S. A., and A. Schwung. 2023. "Graph Neural Networks-Based Scheduler for Production Planning Problems Using Reinforcement Learning." *Journal of Manufacturing Systems* 69 (August): 91–102.

Hanna, J.P., and P. Stone. 2017. "Grounded Action Transformation for Robot Learning in Simulation". *In Thirty-first AAAI Conference on Artificial Intelligence*. San Francisco, California, 4 – 9 February 2017.

Hu, L., Z. Liu, W. Hu, Y. Wang, J. Tan, and F. Wu. 2020. "Petri-Net-Based Dynamic Scheduling of Flexible Manufacturing System via Deep Reinforcement Learning with Graph Convolutional Network." *Journal of Manufacturing Systems* 55 (April): 1–14.

Huang, J.-P., L. Gao, X.-Y. Li, and C.-J. Zhang. 2023. "A Novel Priority Dispatch Rule Generation Method Based on Graph Neural Network and Reinforcement Learning for Distributed Job-Shop Scheduling." *Journal of Manufacturing Systems* 69 (August): 119–34.

Liu, C.-L., C.-C. Chang, and C.-J. Tseng. 2020. "Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems." *IEEE Access* 8: 71752–62.

Liu, J., F. Qiao, M. Zou, J. Zinn, Y. Ma, and B. Vogel-Heuser. 2022. "Dynamic Scheduling for Semiconductor Manufacturing Systems with Uncertainties Using Convolutional Neural Networks and Reinforcement Learning." *Complex & Intelligent Systems* 8 (6): 4641–62.

Min, B., and C. O. Kim. 2022. "State-Dependent Parameter Tuning of the Apparent Tardiness Cost Dispatching Rule Using Deep Reinforcement Learning." *IEEE Access* 10: 20187–98.

Park, J.S., B. Tsang, H. Yedidsion, G. Warnell, D. Kyoung, and P. Stone. 2020. "Learning to Improve Multi-Robot Hallway Navigation". In *Proceedings of the Conference on Robot Learning*, *CORL2020*, edited by J. Kober, F. Ramos and C. J. Tomlin. 1883-1895. Cambridge, MA: PMLR.

Peters, L. 2022 "Systematic Yield Issues Now Top Priority at Advanced Nodes." *Semiconductor Engineering* https://semiengineering.com/systematic-yield-issues-now-top-priority-at-advanced-nodes/, accessed 4th April 2024.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, and Y. Chen. 2017. "Mastering the Game of Go without Human Knowledge". *Nature*, 550(7676): 354-359.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. 2017. "Proximal Policy Optimization Algorithms." arXiv preprint arXiv: 1707.06347.

Song, W., X. Chen, Q. Li, and Z. Cao. 2023. "Flexible Job-Shop Scheduling via Graph Neural Network and Deep Reinforcement Learning." *IEEE Transactions on Industrial Informatics* 19 (2): 1600–1610.

Soykan, B., A. Bany Abdelnabi, and G. Rabadi. 2023. "Adaptive Scheduling In Dynamic Environments: A Data-Driven Approach With Digital Twins And Deep Reinforcement Learning." In *2023 IEEE 3rd International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 1–6. Orlando, FL, USA: IEEE.

Stöckermann, P., A. Immordino, T. Altenmüller, G. Seidel, M. Gebser, P. Tassel, C. W. Chan, and F. Zhang. 2023. "Dispatching in Real Frontend Fabs With Industrial Grade Discrete-Event Simulations by Deep Reinforcement Learning with Evolution Strategies." In *2023 Winter Simulation Conference (WSC)*, 3047–58 https://doi.org/10.1109/WSC60868.2023.10408625.

Tassel, P., B. Kovács, M. Gebser, K. Schekotihin, P. Stöckermann, and G. Seidel. 2023. "Semiconductor Fab Scheduling With Self-Supervised And Reinforcement Learning." In *2023 Winter Simulation Conference (WSC)*, 1924–35 https://doi.org/10.1109/WSC60868.2023.10407747.

Waschneck, B., A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek. 2018. "Optimization of Global Production Scheduling with Deep Reinforcement Learning." *Procedia CIRP* 72:1264–69.

Wurman, P.R., S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T.J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, and L. Gilpin. 2022. "Outracing Champion Gran Turismo Drivers with Deep Reinforcement Learning". *Nature*, 602(7896): 223-228.

You, J., R. Ying, and J. Leskovec. 2020. "Design Space for Graph Neural Networks." *In Proceedings of the 34th International Conference on Neural Information Processing Systems* Vancouver, BC, Canada. 1427-1440.

Zhang, T., K. E. Kabak, C. Heavey, and O. Rose. 2023. "A Reinforcement Learning Approach for Improved Photolithography Schedules." In *2023 Winter Simulation Conference (WSC)*, 2136–47 https://doi.org/10.1109/WSC60868.2023.10408616.

AUTHOR BIOGRAPHIES

DAVID NORMAN a distinguished member of technical staff at Applied Materials. He earned his Ph.D. in mathematics at the University of Minnesota. His interests include applying mathematical optimization, machine learning, and reinforcment learning to scheduling and planning problems in manufacturing and supply-chain managmement. His e-mail address is david_norman@amat.com.

PRAFULLA DAWADI is a data scientist at AI/ML team at Applied Materials. He earned his Ph.D. in computer science from the School of Electrical Engineering and Computer Science at Washington State University, Pullman,WA. His research interests includes data science, machine learning systems and reinforcement learning. His e-mail address is prafulla_dawadi@amat.com.

HAREL YEDIDSION is a research scientist at the AI/ML team at Applied Materials. He earned his Ph.D. form the department of Industrial Engineering and Management at Ben Gurion University in Israel, and was a postdoc fellow at the department of Computer Science at the University of Texas at Austin. His research interests include multi-agent systems, robotics, and reinforcement learning. His e-mail address is harel_yedidsion@amat.com.