

Scalability of Reinforcement Learning Methods for Dispatching in Semiconductor Frontend Fabs: A Comparison of Open-Source Models with Real Industry Datasets

Patrick Stöckermann^{1,2*}, Henning Südfeld^{1,3},
Alessandro Immordino^{1,4}, Thomas Altenmüller¹,
Marc Wegmann³, Martin Gebser², Konstantin Schekotihin²,
Georg Seidel⁵, Chew Wye Chan⁶, Fei Fei Zhang⁶

¹Infiniteon Technologies AG, Am Campeon 1-15, Neubiberg, 85579,
Germany.

²Department of Artificial Intelligence and Cybersecurity, University of
Klagenfurt, Universitätsstraße 65-67, Klagenfurt am Wörthersee, 9020,
Austria.

³Institute for Machine Tools and Industrial Management, Technical
University Munich, Boltzmannstr. 15, Garching b. München, 85748,
Germany.

⁴Department of Information Engineering, University of Padua, Via
Gradenigo 6/B, Padova, 35131, Italy.

⁵Infiniteon Technologies Austria, Siemensstraße 2, Villach, 9500, Austria.

⁶D-SIMLAB Technologies Pte Ltd, 8 Jurong Town Hall Road, 23-05,
Singapore, 609434, Singapore.

*Corresponding author(s). E-mail(s):
patrick.stoeckermann@infineon.com;

Abstract

Benchmark datasets are crucial for evaluating approaches to scheduling or dispatching in the semiconductor industry during the development and deployment phases. However, commonly used benchmark datasets like the Minifab or SMT2020 lack the complex details and constraints found in real-world scenarios. To mitigate this shortcoming, we compare open-source simulation models with a

real industry dataset to evaluate how optimization methods scale with different levels of complexity. Specifically, we focus on Reinforcement Learning methods, performing optimization based on policy-gradient and Evolution Strategies. Our research provides insights into the effectiveness of these optimization methods and their applicability to realistic semiconductor frontend fab simulations. We show that our proposed Evolution Strategies-based method scales much better than a comparable policy-gradient-based approach. Moreover, we identify the selection and combination of relevant bottleneck tools to control by the agent as crucial for an efficient optimization. For the generalization across different loading scenarios and stochastic tool failure patterns, we achieve advantages when utilizing a diverse training dataset. While the overall approach is computationally expensive, it manages to scale well with the number of CPU cores used for training. For the real industry dataset, we achieve an improvement of up to 4 % regarding tardiness and up to 1 % regarding throughput. For the less complex open-source models Minifab and SMT2020, we observe double-digit percentage improvement in tardiness and single digit percentage improvement in throughput by use of Evolution Strategies.

Keywords: Reinforcement Learning, Scheduling, Dispatching, Semiconductor Manufacturing, Complexity, Scalability, Evolution Strategies, SMT2020, Minifab

1 Introduction

Scheduling and dispatching problems in semiconductor frontend manufacturing present significant challenges due to their complexity and optimization potential. However, most approaches lack comparability as they are evaluated on vastly different datasets. Even for the same dataset and approach, researchers can obtain varying results due to the different simulator engines used and the stochastic nature of these models, which can depend on random seeds. Therefore, we aim to compare multiple datasets and test two different Reinforcement Learning (RL) approaches on all of the datasets to evaluate scalability and optimization potential. As public benchmark scenarios often neglect important details of real manufacturing facilities, we additionally include a large-scale industry dataset in our tests.

With increasing complexity or problem size, RL training is typically distributed by computing trajectories in many parallel environments. Besides classic policy-gradient RL approaches like Proximal Policy Optimization (PPO), Evolution Strategies (ES) have been found to be a scalable alternative [49]. Scalability of RL systems also includes implementation effort [29], where RLlib [30] offers an efficient solution to realize distributed training techniques for RL systems, including automatic hyperparameter search.

In this paper, we compare different testbeds from the literature with real large-scale semiconductor manufacturing facilities. We then evaluate the potential and scalability of different Deep Reinforcement Learning (DRL) approaches for dispatching lots in these scenarios.

Our contributions can be summarized as follows:

- We present the **first application of RL to multiple public benchmark testbeds** as well as a **large-scale industrial-grade scenario** for lot dispatching in semiconductor manufacturing.
- We compare the **scalability** of RL algorithms **across different testbeds** and for **various tools** within individual scenarios.
- We analyze the **generalization properties** of the trained RL agents across different scenarios.
- We experimentally investigate the **computational cost** associated with the benchmark scenarios.

2 Fundamentals and Related Work

In order to better understand the different classes of scheduling problems in research and industry, we give an overview regarding different classifications for those problems and then explain the Semiconductor Factory Scheduling Problem. Furthermore, we explain the main concepts of RL and introduce the two algorithms most relevant in this publication.

2.1 Scheduling in Semiconductor Manufacturing

Scheduling problems are characterized by multiple constraints of different types (e.g., process technology or logistics), which affect the assignment of jobs to machines [23]. The most common scheduling problems will be introduced in the following.

Single Machine Scheduling Problems (SMSPs) are relevant for systems with an individual bottleneck that significantly impacts the overall system performance. Often the job sequence is optimized with static dispatching rules such as First-In First-Out (FIFO) or Earliest Due Date (EDD) [46].

The **Parallel Machine Scheduling Problem (PMSP)** deals with a number of parallel machines. This occurs in production environments where several stages or workcenters are arranged, each with multiple machines in parallel. The machines at a workcenter may be identical and a job can be processed on any of them [46].

Flow Shop Scheduling Problems (FSSPs) are typical in setups where multiple operations need to be performed on different machines. Each job has the same sequence of machines to go through. The **Flexible Flow Shop Scheduling Problem (FFSSP)** is an extension of this problem with multiple parallel machines at each stage [46].

The **Job Shop Scheduling Problem (JSSP)** additionally has jobs with different orders regarding the sequence of operations. In the most simple version, each of n jobs has to be processed once on each of m machines [46].

In semiconductor manufacturing, the **Flexible Job Shop Scheduling Problem (FJSSP)** is typical and features workcenters with multiple machines in parallel [46].

In case some jobs can be processed together at the same time on the same machine, it is a **batching problem**. Grouping jobs together to build batches becomes part of the scheduling problem in this case [6]. The **Complex Job Shop Scheduling Problem (CJSSP)** incorporates, among other things, batching and reentrant flows.

Table 1: Comparison of scheduling problems

	SMSP	PMSP	FSSP	FFSSP	JSSP	FJSSP	CJSSP
parallel machines		x		x	x	x	x
sequential operations			x	x	x	x	x
different sequences					x	x	x
batching						x	x
reentrant flow							x

In a reentrant flow shop, the same machine can occur multiple times in the same route. This significantly increases complexity [24].

The properties of the described scheduling problems are compared in Table 1.

Semiconductor Frontend Manufacturing (SCFM) includes hundreds of process steps, on hundreds of machines with highly diverse and stochastic processing times. It involves the coordination of n diverse jobs and m sophisticated machines in a non-preemptive queuing environment. Operations can be processed on multiple candidate machines constrained by setup requirements. Based on these traits, the **Semiconductor Factory Scheduling Problem (SFSP)** is often considered a complex job shop problem, with some models considering additional details. It is often decomposed into SMSP, PMSP, or FJSSP in order to simplify implementation and reduce the computation time. Typically, only bottleneck machines, often the lithography area, are scheduled with traditional approaches using Constraint Programming (CP) or Mixed Integer Programming (MIP) [37, 41].

The $(\alpha||\beta||\gamma)$ notation introduced by [17] is even more elaborate and defines α as the type of machine environment defined by one of the scheduling problem definitions introduced above. Information about the jobs and sequencing constraints is given by β . The objective function is given by γ . More details can be found in [23].

We can further differentiate between deterministic and stochastic scheduling. In deterministic scheduling, processing times, setup times, transport times, due dates and other parameters are assumed to be known in advance and not influenced by uncertainty. However, there are many sources of uncertainty in a real semiconductor manufacturing environment. One of the biggest sources of uncertainty are machine breakdowns. In stochastic scheduling, probabilistic values are not assumed to be known in advance. They are replaced by probability distributions modeling the variability of reality. [42, 47]

Even the classic JSSP is NP-hard [2], and all more advanced variants as well [18]. In order to understand the scalability of deterministic methods, [10] analyzed the performance of different CP solvers on scenarios with up to a million operations to be scheduled on up to one thousand machines. However, they did not consider the constraints that are required in an actual industry use case such as the dedications of processes to a subset of machines within a tool group, batch processes with complex recipe-related constraints and a much more diverse load-mix. Even if the algorithm would be capable of computing a solution within a few hours for a realistic dataset, the solution is already too old after some minutes, as the situation in a real facility constantly changes due to varying processing and transport times as well as failed processes and machine breakdowns. Often, manufacturers use inflexible and suboptimal

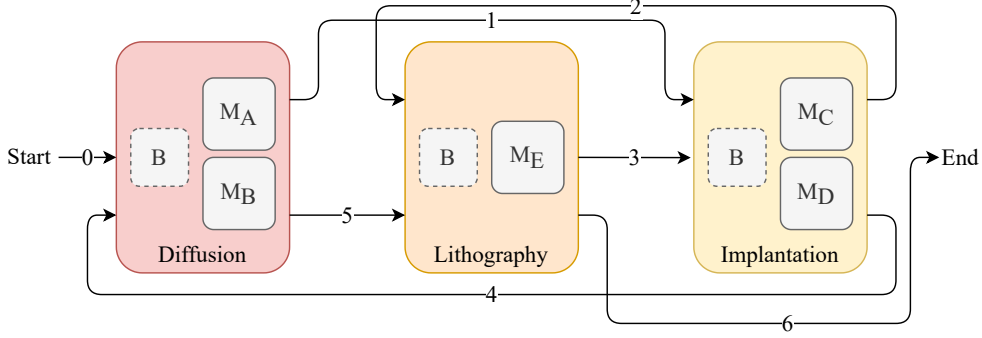


Fig. 1: The Minifab model layout, adapted from [13].

handcrafted heuristics for dispatching to circumvent the problems of mathematical scheduling.

2.2 Semiconductor Frontened Manufacturing Testbeds

In order to evaluate deterministic scheduling approaches without any stochastic modelling [10, 60], researchers often use JSSP benchmark suites like Taillard’s [59] and solvers such as the open-source software OR-Tools [44] or the commercial solution IBM CPLEX [27].

However, we are focusing on stochastic models that are much closer to the properties found in real semiconductor manufacturing facilities. One of the first and widely used testbeds for SCFM scheduling problems is the Minifab model which was developed by researchers from Intel and Arizona State University in an attempt to bridge the gap between research and industry. It incorporates routes with 6 steps on 5 machines. More specifically, it entails two implantation, one lithography and two diffusion tools. The latter are additionally capable of batching three lots at a time together. The fab incorporates three different products with different release rates. Generally, the model aims to provide a variant of the CJSSP with stochastic influences such as machine breakdowns and preventive maintenance. The Minifab model layout [14, 55] is shown in Figure 1.

The Semiconductor Manufacturing Testbed 2020 (SMT2020) is an extension of the Measurement and Improvement of Manufacturing Capacity (MIMAC) datasets [16], which was developed by SEMATECH in 1995. SMT2020 incorporates a larger factory scale with more tools and processing steps. Furthermore, preventive maintenance and unscheduled downtime are modelled. The authors of SMT2020 also note that the older MIMAC datasets miss clear implementation guidelines [25], and therefore it is hard to reproduce results.

The SMT2020 fab models include scenarios for custom orders (Low-Volume/High-Mix) and mass production (High-Volume/Low-Mix). More than 500 operations are modelled for the individual and diverse product routes, performed by between 1071 and 1265 machines depending on the scenario. The transport times are accounted for and operation- as well as sequence-dependent setups are modelled respecting tool specific

Table 2: Comparison of reference models adapted from [4].

Model	MIMAC	Minifab	Harris	SEMATECH 300mm	SMT2020
No. machines	up to 260	5	12	275	1043
No. machine groups	up to 85	3	11	103	105
No. products	up to 21	2	3	1	10
No. process steps	up to 280	6	up to 22	364	up to 632

requirements of the semiconductor manufacturing process. Moreover, the SMT2020 fab models include batching policies for some operations and probabilistic skipping of quality assessments. Critical jobs, called “Hot Lots” and “Super-Hot Lots”, are prioritized to meet urgent deadlines. SFSP adapts to real-time issues, such as defect detection leading to job rerouting [25, 61].

Additional models worth mentioning are the recently published Complex Job Shop Simulation reference model (CoJoSim) [4], which offers a reference implementation based on the MIMAC datasets as well as the smaller Sematech 300mm model [7] and the scaled down model of a Harris Corporation fab [22].

The mentioned models are compared to each other in Table 2. In SMT2020, the scale of the addressed problem goes well beyond the smaller test datasets. However, the SMT2020 fab models have some shortcomings compared to real manufacturing data. We therefore introduce a real industry scenario of similar size but with a more diverse load mix and complex tool dedications. Similar to SMT2020, our dataset has more than 1000 machines, but contains more than ten times the number of products considered in SMT2020. Another important detail that is missing in SMT2020 is that a specific operation may not be executable on all tools within a group and can have flexible processing times on different tools. Especially the dedication of processes to a subset of machines within a tool group in combination with a more diverse load mix leads to a sharp increase in constraints. This also affects batch processes with complex recipe-related constraints limiting the type of products which can be batched together. In fact, the homogeneity between tools of the same group assumed by SMT2020 drastically reduces the complexity. For AI-based methods, extracting statistical relationships and patterns in the data becomes much more difficult with a more diverse dataset. Furthermore, we consider multiple heterogeneous loading scenarios with varying load-mix and volume for the industry case. This significantly increases the difficulty of generalization, as the dispatching policy must not overfit on the training scenarios. Furthermore, the increased complexity leads to longer execution time for the simulation, increasing the computational cost of training.

Lastly, our real industry scenario has equipment group specific combinations of dispatching heuristics, fine tuned by domain experts. These rules are the most difficult to beat among the different models. [56]

2.3 Reinforcement Learning

RL enables the autonomous learning of optimized decision-making policies to act in complex environments. This is possible through continuous learning by interacting with an environment, i.e., a simulation. Usually, the Markov Decision Process (MDP) is used to formalize RL environments [58]. RL algorithms typically optimize a cumulative

Algorithm 1 PPO

```
1: Initialize  $\theta_{\text{old}}, \phi_{\text{old}}$ 
2: for iteration  $t = 1, 2, \dots$  do
3:   Synchronize networks parameters among all the  $N$  agents
4:    $\mathcal{D} = \emptyset$ 
5:   for agent  $t = 1, \dots, N$  do
6:     Collect a trajectory  $\tau$  of length  $T$  using  $\pi_{\theta_{\text{old}}}$ 
7:     Post-process  $\tau$  to get the advantage estimation for each sample
8:      $\mathcal{D} = \mathcal{D} \cup \tau$ 
9:   end for
10:  Optimize  $\mathcal{L}$  w.r.t.  $\theta, \phi$  by performing  $K$  epochs with mini-batch size  $M$  over  $\mathcal{D}$ 
11:   $\theta_{\text{old}} \leftarrow \theta$ 
12:   $\phi_{\text{old}} \leftarrow \phi$ 
13: end for
```

reward R over a number of steps T instead of optimizing for individual direct returns r_t for each time step t :

$$R = \sum_{t=0}^{T-1} \gamma^t \cdot r_{t+1}$$

The individual rewards are weighted by introducing a discount factor γ , which determines the importance of immediate and future rewards. There are numerous RL algorithms with different advantages and weaknesses. Some of the most popular derivative-based RL algorithms, such as policy-based PPO [40, 52] or value-based Deep Q-Network (DQN) [39] approaches, have also been used for dispatching and scheduling in semiconductor manufacturing [1, 60]. Furthermore, taking ES [49] as an alternative to derivative-based RL algorithms also attracted interest in scheduling applications [56, 61].

PPO [52] is based on the Trust Region Policy Optimization (TRPO) [51] algorithm and significantly simplifies implementation. The stochastic policy is defined as π_{θ} , where $\theta \in \mathbb{R}^n$ is a vector of parameters. The policy $\pi_{\theta}(a_t | s_t)$ defines the probability to take an action a_t in the state s_t at time step t . The likelihood ratio $\rho_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)}$, with θ_{old} denoting the policy parameters, is used during sampling. This allows to define the PPO objective, which is called clipped surrogate objective, as

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

with the hyperparameter $\epsilon \geq 0$ and the advantage estimation \hat{A}_t for time step t . The approach ensures that PPO does not perform too radical updates to the parameters of the policy. In its Actor-Critic variant, the parameter vector $\phi \in \mathbb{R}^m$ defines the Neural Network (NN) of the state-value function estimator, also called critic, and θ represents the parameters of the policy network, also called actor. A simplified version of our PPO implementation, which is based on Ray RLLib [30], is shown in Algorithm 1 with N parallel agents and data buffer \mathcal{D} .

Algorithm 2 Summarized method for CMA-ES presented by [19]

- 1: **Input** : Step size σ_0 , covariance matrix $C_0 = I$, mean vector μ_0 , number N_{best} of most promising solutions considered for optimization, number n of parallel agents, maximum iterations T
 - 2: **for** $t = 1, \dots, T$ **do**
 - 3: Sample $w_1, \dots, w_n \sim N(\mu_t, \sigma^2 C_t)$
 - 4: Compute return values $F_i = F(w_i)$ for $i = 1, \dots, n$
 - 5: $\sigma_{t+1} \leftarrow \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} (w_i - \mu_t)$
 - 6: $\mu_{t+1} \leftarrow \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} w_i$
 - 7: **end for**
-

ES can be seen as a black-box algorithm and thus solves the credit assignment problem [58] typical for most RL approaches. The agent only receives one reward for each episode, which is well-suited for environments where a good value function estimate is hard to derive and the effects of individual actions apply with substantial delay. The algorithms add Gaussian noise, with the mean μ and fixed covariance $\sigma^2 I$ in the most simple variant, to the parameters instead of using back-propagation. This leads to good exploration, easy implementation, but poor scalability with the number of parameters θ .

We make use of the Covariance Matrix Adaptation (CMA) approach [19] utilized by [56], which promises faster convergence by using the covariance of the parameters θ for sampling. Algorithm 2 outlines the CMA-ES approach, where α_μ controls the rate at which the mean μ is updated. N_{best} is a hyperparameter that determines the subset of the most promising candidates w_i based on their score given by the fitness function $F(w_i)$. At each iteration t , multiple candidate solutions created by sampling are tested in parallel. The returns are then used to approximate the fitness function and guide the sampling for the next iteration.

2.4 Reinforcement Learning in Semiconductor Frontend Manufacturing

We conduct a comprehensive structured literature search in order to find relevant publications in the field of RL-based dispatching or scheduling approaches in semiconductor manufacturing based on [67]. We execute the search using the abstract and citation database Scopus [15] with the following search string:

```
( TITLE ( "Dispatch*" OR "Schedul*" ) OR ABS ( "Dispatch*" OR "Schedul*" ) )  
  ↳ AND ( TITLE ( "Semiconductor Frontend Manufacturing" OR "Semiconductor  
  ↳ Production" OR "Semiconductor Manufacturing" OR "Wafer fab*" ) OR ABS  
  ↳ ( "Semiconductor Frontend Manufacturing" OR "Semiconductor Production"  
  ↳ OR "Semiconductor Manufacturing" OR "Wafer fab*" ) ) AND ( TITLE ( "  
  ↳ Reinforcement Learning" OR "RL" ) OR ABS ( "Reinforcement Learning" OR  
  ↳ "RL" ) )
```

Publications from 2023 and before (1 year ago) must have at least 1 citation and 5 before 2020 (5 years ago) to be included due to significance. Next, we filter out

Table 3: Comparison of RL-based scheduling and dispatching approaches in the literature sorted by citations (descending). The maximum size is reported with the number m of machines and the number of equipment groups (EGs), if applicable.

Year	Author	Model(s)	Approach	Max Size
2018	Waschneck et al. [66]	Custom	DQN [39]	$m = 6$, 4 EGs
2020	Park et al. [43]	Custom	DQN [39]	$m = 175$
2018	Stricker et al. [57]	Custom	DQN [39]	$m = 8$, 3 EGs
2018	Waschneck et al. [65]	Custom	DQN [39]	$m = 6$, 4 EGs
2020	Altenmüller et al. [1]	Custom	DQN [39]	$m = 10$, 5 EGs
2021	Chien et al. [8]	Custom	DQN [39]	$m = 9$
2022	Lin et al. [32]	Custom, Wu [69]	GWO [38]	$m = 100$
2023	Lin et al. [33]	YFJS-series [5]	LCS [70]	$m = 26$
2023	Sakr et al. [48]	Custom	DQN [39]	$m = ?$, 23 EGs
2022	Liu et al. [34]	MiniFab [55]	A3C [40]	$m = 5$, 3 EGs
2020	Shiue et al. [54]	SEMATECH ¹ [7]	Q-Learning [58]	$m = ?$, 12 EGs
2019	Lee et al. [28]	Custom	SARSA [58]	$m = 160$
2023	Tassel et al. [61]	SMT2020 [25]	ES [49]	$m = 1265$, 106 EGs
2024	Lu [35]	Custom	DDQN [20]	$m = 50$
2018	Wangl et al. [63]	Custom	Custom	$m = 220$
2023	Ma et al. [36]	MiniFab [55]	DDQN [20]	$m = 5$, 3 EGs
2023	Stöckermann et al. [56]	Custom	CMA-ES [19, 49]	$m > 1000$
2023	Liao et al. [31]	Custom ² , TA ² [59]	PPO [52]	$m = 20$
2024 ³	Yedidsion [71]	Custom ²	PPO [52]	$m = 4$, 3 EGs
2024 ³	Zhangg [73]	SFTSP ² [68]	Q-Learning [58]	$m = 36$
2024 ³	Dong [12]	TA ² [59], DMU ² [11]	GTN [72]+ABC [21]	$m = 20$
2024 ³	Wang [64]	Custom	DDQN [20]	$m = 18$
2024 ³	Shao [53]	Custom	A2C [40]	$m = 10$, 4 EGs

¹Simplified version

²No stochasticity modeled

³Recently published, no citations yet

everything that is not related to lot dispatching in a system with more than one modeled equipment group. These filtering steps reduce the results to 23 in Table 3.

It becomes obvious that it is very hard to compare the approaches as they were tested on many different and often custom models. A problem that all of the presented models have in common is the comparability of the implementation, even for the same model [4]. Some researchers implement their simulator with Python packages like SimPy [50], build a new simulator from scratch [26, 61], or rely on a commercial simulator like Autosched AP [45], Siemens Tecnomatix Plant Simulation [3, 4], or D-Simlab D-Simcon Forecaster [9, 56]. Even with the same implementation, individual runs can vary a lot due to the stochastic events that heavily depend on the random seed. Furthermore, the hardware and computational resources vary between research labs. We therefore aim to conduct comparable experiments across the Minifab model, SMT2020 and an industrial scenario, all using the D-Simlab D-Simcon Forecaster [9].

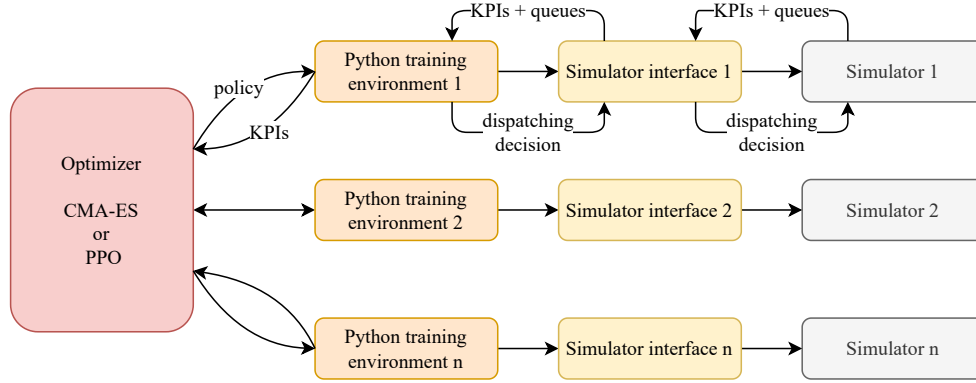


Fig. 2: Architecture of experiments using the CMA-ES or PPO optimizer, respectively.

3 Approach and Design of Experiments

In this section, we introduce our overall approach as well as the design of experiments. This includes the utilized architecture, the simulator, the NN, and the most important parts of the RL system, namely the policy, observation, action, and reward concepts.

3.1 Simulator Architecture

In order to conduct our experiments, we create a highly flexible training architecture with interchangeable simulation models and optimizers (see Figure 2). The training is always controlled by the optimizer, which can be switched between the PPO and CMA-ES algorithms described above. The optimizer then creates parallel workers running on separate CPUs.

Each worker has a Python environment defining the current policy and thus the interaction with the simulator. The environment collects the observations of the state at every decision step to compute a dispatching decision, using the policy and providing the information needed for reward calculation. The individual environments are interacting with separate simulator instances through an interface. Each simulation can be individually initialized with different simulation models and different random seeds, defining stochastic events like tool failure.

We use D-SIMLAB’s highly realistic, state-of-the-art D-SIMCON simulator [9], which can handle our industry scenario with over a thousand pieces of equipment as well as the open-source models Minifab and SMT2020. The simulator is calling the environment every time a dispatching decision has to be made and provides information about the simulator state, such as overall KPIs and queues of waiting lots.

In order to calculate the tardiness of lots, we need due dates for each lot, which are the planned dates for completion. We define the tardiness as the time between the due dates and the real completion time. If a lot is behind schedule, the tardiness is the difference between these two dates, or zero in case the lot is ahead of schedule. The tardiness of uncompleted lots is calculated using their step due date, calculated from the final due date by subtracting the planned cycle time of remaining steps. The

authors of SMT2020 already provide due dates in their dataset, and the same is the case for our industry dataset. For MiniFab, this detail is missing. Here we generate the due dates by multiplying the raw processing time with a planned Flow Factor (FF) to obtain the planned Cycle Time (CT) and thus also the due date. The planned FF is sampled from a uniform distribution between 2.1 and 2.5. We furthermore modify the processing times for each tool in the MiniFab model to vary between products. This condition makes the model more realistic.

In the case of ES, the policy of the individual environments is only updated once every episode, which is an entire simulation run simulating multiple weeks in the fab. In the case of PPO, the optimizer is collecting frequent samples of state, action, and reward tuples to provide many yet small intermediate updates during an episode. PPO therefore has much more synchronization and respective communication overhead.

3.2 Observation and Policy

We extend on the experiments of [56] by utilizing the CMA-ES optimizer [49] for the optimization of Deep NNs in RL. Additionally, we adapt the approach to use it with PPO.

Each simulator registers a predefined list of tools that are controlled by the agent at the beginning of the training. The agent is called every time a dispatching decision has to be performed at one of those tools and receives representations of the lots that are waiting in front of the current tool. The entire queue of lots at the work center, to which the tool is assigned, is filtered to identify lots that can be processed by that specific tool. The queue of lots is represented using the following attributes, with the last two only used in case batch tools are controlled:

- Time to fab due date
- Time to step due date
- Waiting time at current step
- Number of alternative tools
- Boolean flag indicating whether a faster tool exists
- Expected processing time
- Setup time
- Expected remaining cycle time
- Number of wafers in lot
- Boolean flag indicating whether it is a batch tool
- Percentage of available wafers to form full batch

All attributes are subject to z -score normalization using continuously updated statistics collected from all simulations during the experiment. The policy is then used to compute a score for each lot. ES picks the lot with the highest score and PPO samples from a distribution, which is giving the probability of each lot based on the computed score. In case multiple lots can be dispatched at the same time, which is the case for batching tools, the algorithm selects lots that can be batched together with the lot with the highest score. Those additional lots are also selected based on their score. This is only done for ES, though, as PPO is based on individual discrete decisions and expects one index as the result of each forward pass. Batching tools are therefore not considered for the PPO experiments given that the algorithm cannot handle such an implementation without major modifications.

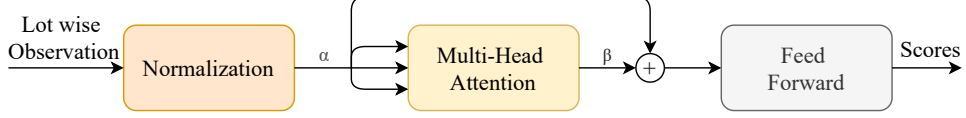


Fig. 3: The architecture of the NN for ES.

In order to be queue size-independent and capture the relationships between the lots in the queue, the observation is fed to the network lot by lot and combined as the dot product of projections. This is possible by the use of the Scaled Dot-Product Attention [62]. It requires three projections for the queries, keys, and values and is computed by applying the softmax function on the scaled transposed dot product of the query Q with all keys K . The transposed dot product is scaled by the dimension d_k of the key projection, and the softmax result is multiplied with the value V :

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

After the information of each lot is infused into the representation of the other lots, the modified representations are fed individually through a feed forward fully connected NN in order to compute the score for each lot (see Figure 3). The architecture is purposefully small in order to keep the number of trainable parameters low, as ES does not scale well with the number of parameters.

For PPO, we are additionally estimating the value function for the critic in order to compute the advantage for the PPO loss function (see Figure 4). This is important as our policy updates as well as the updates to the reward are much more frequent than for the evolutionary approach. This highlights the strength and weakness of PPO in one: if we are able to accurately estimate the value of the intermediate states during the episode, this should lead to more stable and faster convergence compared to updating the policy only once every episode. However, in case this estimate is noisy or incorrect, we cannot converge to a stable and valuable policy.

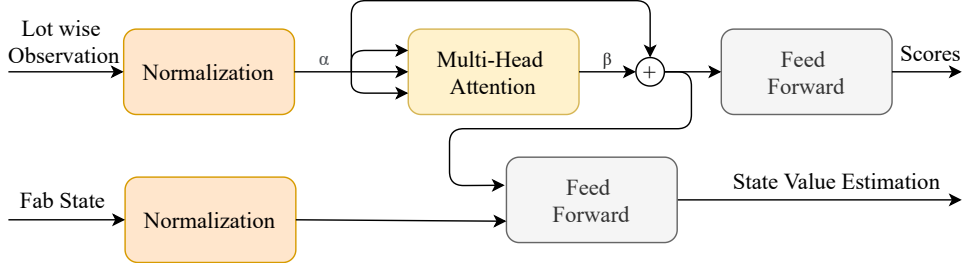


Fig. 4: The architecture of the NN for PPO.

The additional inputs representing the fab state are summarized as follows:

- Work center WIP
- Completed wafers
- Total tardiness
- Total tardy lot count
- Average tardiness
- Standard deviation tardiness
- Average cycle time
- Average fab WIP

All values are compared to the respective values of the reference runs at the same time and given as the difference to the agent.

3.3 ES Cost Function and PPO Reward

We only assign one cost value c_{ES} per episode for the ES experiments. It is normalized by dividing the resulting KPIs of the training episode by the KPIs of the reference run using the default dispatching heuristics. As we mainly want to optimize the tardiness of lots, we base the cost on the tardiness td_{out} of the completed wafers. In order to prevent the agent from sacrificing the other KPIs, we introduce conditional terms to the cost calculation that are only used if we get worse performance on the KPIs of the inner tardiness td_{in} and the throughput tp . The throughput is the number of completed wafers for the entire episode. We set $\alpha_1 = \alpha_2 = 10$ in order to prevent the agent from accepting small worsening of td_{in} and tp to improve td_{out} :

$$c_{\text{ES}}^* = \frac{td_{\text{out}}}{td_{\text{out},\text{ref}}}$$

$$c_{\text{ES}} = \begin{cases} c_{\text{ES}}^* \cdot \alpha_1 \frac{td_{\text{in}}}{td_{\text{in},\text{ref}}} & td_{\text{in}} > td_{\text{in},\text{ref}} \\ c_{\text{ES}}^* \cdot \alpha_2 \frac{tp_{\text{ref}}}{tp} & tp < tp_{\text{ref}} \end{cases}$$

For our industry scenario and its complex interdependencies, we utilize an adjusted cost function c'_{ES} , as c_{ES} does not lead to satisfying results for that model. This seems to be due to the fact that it is way harder to maintain or even improve the throughput for the real-world scenario:

$$c'_{\text{ES}} = \frac{td_{\text{out}} + td_{\text{in}}}{td_{\text{out},\text{ref}} + td_{\text{in},\text{ref}}} \cdot \left(\frac{tp_{\text{ref}}}{tp} \right)^2$$

For PPO, we assign a reward at each step based on the rolling mean of the same metrics as for the ES approach. The rolling mean is recalculated every hour over the past 24 hours in order to reduce noise. Hence, the reward $r_{\text{PPO},t}$ for time steps t is identical if the steps occur within the same hour:

$$r_{\text{PPO},t} = \frac{tp_t}{(tp_t \cdot td_{\text{out},t} + \text{WIP}_t \cdot td_{\text{in},t}) \cdot (\text{WIP}_t + tp_t)}$$

with tp_t being the number of completed wafers during the past 24 hours and WIP_t the number of wafers remaining in the system at the end of the 24-hour interval. The reward is based on the division of the hourly throughput by the tardiness. The

tardiness is the weighted sum of the tardiness of the wip and the tardiness of the wafers completed in the past 24 hours.

As the PPO approach receives regular updates of the reward, not only at the end of an episode, and should understand the relationship between individual actions and the reward, we design it continuously without case distinctions such as for c_{ES} . The reward definition $r_{PPO,t}$ has been empirically found to be the best performing one in our experiments, additional rewards are compared in the appendix under Section A.

4 Results

In this section, we investigate the performance of simulation models with different static dispatching rules. Then, we select the best dispatching rule as a baseline and reference for comparing the RL training techniques. Moreover, we analyze the impact of the utilized computational resources as well as the generalization capabilities.

4.1 Comparison of the Models' Default Dispatching Rules

For the Minifab model, we consider the following dispatching heuristics:

- First-In First-Out (FIFO)
- Critical Ratio (CR) (dividing remaining time to due date by the expected time to complete a lot)
- Shortest Remaining Processing Time (SRPT) (for the remaining steps)
- Shortest Processing Time (SPT) (for the next step)
- Earliest Due Date (EDD)

Table 4 shows (normalized) results regarding the number of completed wafers and the tardiness of lots. As the SRPT heuristic dominates in terms of both performance metrics, we select it as baseline for the Minifab model. Here and for all following plots and tables, the tardiness td is summed over all completed lots (CL), relative to their final due dates (DD), as well as uncompleted lots (WIP) relative to their step due dates (SDD):

$$td = td_{in} + td_{out} = \sum_{l \in WIP} SDD_l - t + \sum_{l \in CL} DD_l - tc_l$$

considering the current time t , in this case the end time of the simulation, and the time of completion tc for each finished lot l . Note that we want to increase the number of completed wafers, but decrease the tardiness.

The authors of the SMT2020 fab models propose a hierarchical combination of dispatching heuristics [25], which is also typical in the industry. Lots that are currently under time constraints, i.e., they will have to repeat the previous operation if the next step is not performed in time, are always processed first. Prioritized lots are dispatched next: Super-Hot Lots at the highest priority, followed by Hot Lots. Third, jobs that align with the current machine setup are favored as this prevents unnecessary setup time from reducing the throughput. Finally, a tie-breaking heuristic is applied, where we consider the same five dispatching heuristics as also used for the Minifab model.

	Completed wafers	Tardiness
FIFO	98.1	100.0
CR	99.5	68.2
SRPT	100.0	61.5
SPT	98.8	87.1
EDD	99.2	73.5

Table 4: Performance of different heuristics for the Minifab model, normalized using the highest value for each metric.

	Completed wafers	Tardiness
FIFO	100.0	3.2
CR	98.3	2.4
SRPT	81.8	87.8
SPT	79.1	100.0
EDD	91.3	25.3

Table 5: Performance of different heuristics for the SMT2020 model, normalized using the highest value for each metric.

The results for SMT2020 in Table 5 yield that the CR heuristic performs best regarding the tardiness of lots, which is our main metric to optimize. Hence, although FIFO has a slight advantage in the number of completed wafers, we select CR as the baseline dispatching heuristic for the SMT2020 fab models.

4.2 Experiments with PPO Algorithm

Our first series of RL experiments investigates the behavior during training of a dispatching strategy for the lithography area. All other tools are controlled by the default dispatching heuristics. It has to be noted that, while the number of overall tools and the number of lithography tools are somewhat comparable for SMT2020 and our industry scenario, it is just one tool in the case of the Minifab. However, as the model only has a total number of five tools, the impact of only controlling the lithography tool is expected to be very significant.

The hyperparameters and the reward have to be adapted for the individual models. This is because the cause-effect relationship is delayed for the larger models as more decisions are made per day. Since the frequency of the KPI-based reward feedback per simulated day is constant but the number of actions changes, more actions are performed in-between reward signals. Secondly, the larger models have a more stable WIP level and the reward signal is less noisy. In order to account for this, the parameters are adapted as follows. The rollout fragment length, also called horizon, is adapted to be roughly the equivalent of 1-2 days of dispatching. This parameter defines the number of actions per worker and sample over which the advantage is calculated.

Another important hyperparameter is the truncation of episodes. If it is true (true eps in Figure 5), we perform multiple policy updates per episode, while not taking all samples collected from the entire episode into account. With our settings and setup, using complete episodes is only possible for the MiniFab, as the size of the collected samples becomes too big to handle for the larger models. Complete episodes are thus only considered for the Minifab model in Figure 5, plotting the improvement in percent relative to the baseline dispatching heuristic for the tardiness and throughput metrics over the training progress in episodes. Please note that the truncation of episodes does not change the simulation horizon but only the number of samples considered for a policy update of the PPO. While we show only the results for the best performing reward in Figure 5, additional rewards are compared in the appendix under Section A.

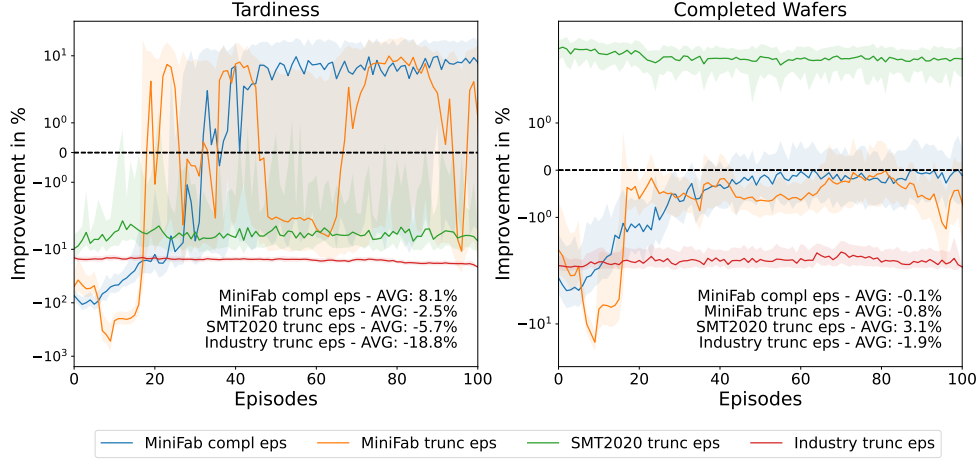


Fig. 5: Results for the PPO experiments controlling the lithography area for three different simulation models with truncated and completed episodes for policy updates.

In one episode, we collect about 50 rollouts per worker. For the Minifab model, each rollout consists of 16 steps, where each step contains an observation for each lot in the queue. The queue length varies between one and a few hundred lots. For SMT2020, each rollout contains over 3000 steps, so that the observations become substantially more exhaustive. This means that the size of the collected samples is 100-1000 times higher, requiring much more memory and making the policy updates computationally expensive. However, using complete episodes leads to much more stable training progress for the MiniFab and could also be useful for the other models. It becomes clear from Figure 5 that we manage to achieve significant improvements for the Minifab model only. Reasons for this could be that the effects of individual actions are too delayed, the KPIs are too noisy, and the value function estimate is too inaccurate for the larger models. These issues are illustrated in Figure 6. In case a long rollout fragment length is used to tackle the noisy reward signal and long-lasting effects of actions, the sample correlation among the steps collected by the same worker in the same rollout increases. This makes it harder for the algorithm to identify advantageous actions within the rollout. In turn, more parallel rollout workers would be needed to collect more samples, which leads to a drastic increase in required computing power and memory.

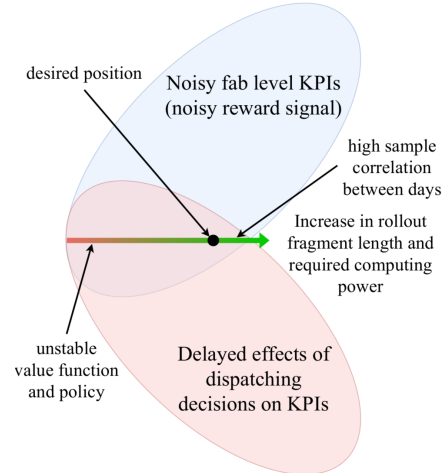


Fig. 6: Influences on PPO behavior.

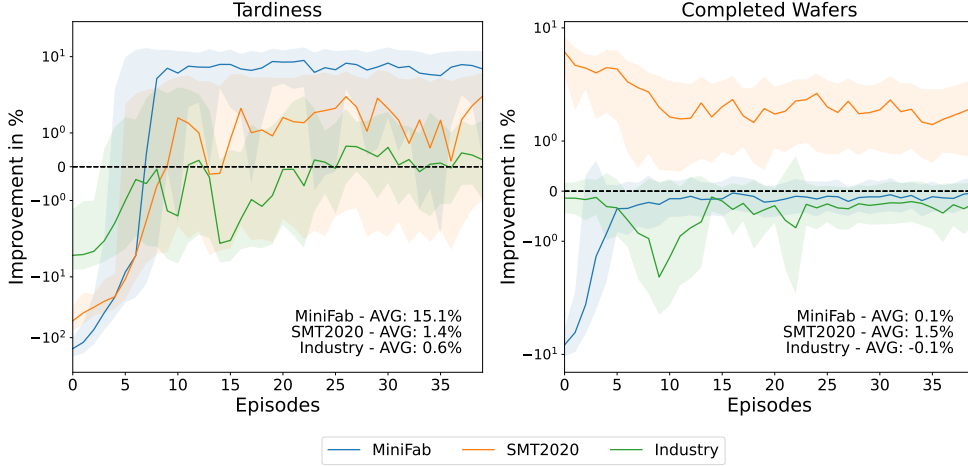


Fig. 7: Results for the CMA-ES experiments controlling the lithography area for three different simulation models.

While the realistic models have cycle times of many weeks and complex long-lasting effects, as well as an unstable situation due to ever-changing loading, the Minifab features constant loading and a short cycle time of only a few days. This explains why the Minifab achieves improvements under the settings of the experiments, while the other models fail. We suspect that, in order to make PPO work for the larger models, the rollout fragment length as well as the number of workers have to be drastically increased, or we would need to find a smoother dense reward.

4.3 Experiments with ES Algorithm

Figure 7 follows the same notation as Figure 5, but instead of the PPO optimizer, results for the CMA-ES training progress are shown. It can be seen that this strategy can handle the industry scenario and SMT2020 much better. Overall, this approach leads to better training results than PPO but converges slower.

Figure 8 displays training results for the Minifab model with different combinations of controlled tools. It can be seen that the best results are achieved for the control of the entire system of three different tool types. It is further noticeable that optimization regarding the control of the diffusion tools and their batching process is only effective in combination with the other tools. This can be explained by the fact that the effective building of batches is dependent on the WIP supply from the previous steps. Similar behavior can be observed for SMT2020 and the industry scenario (see Figure B3 and Figure B4 in the appendix). That is, the higher the percentage of RL-controlled tools, the higher is the potential improvement. The results are even better when training runs for 200 instead of 40 episodes only (see Figure B5 in the appendix), which comes as a matter of the computational cost.

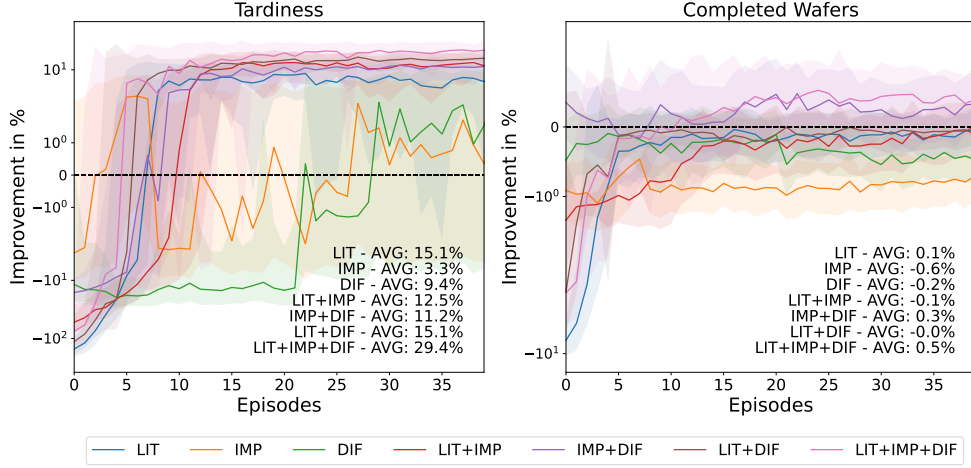


Fig. 8: Results for the CMA-ES experiments controlling different combinations of tools for the Minifab model.

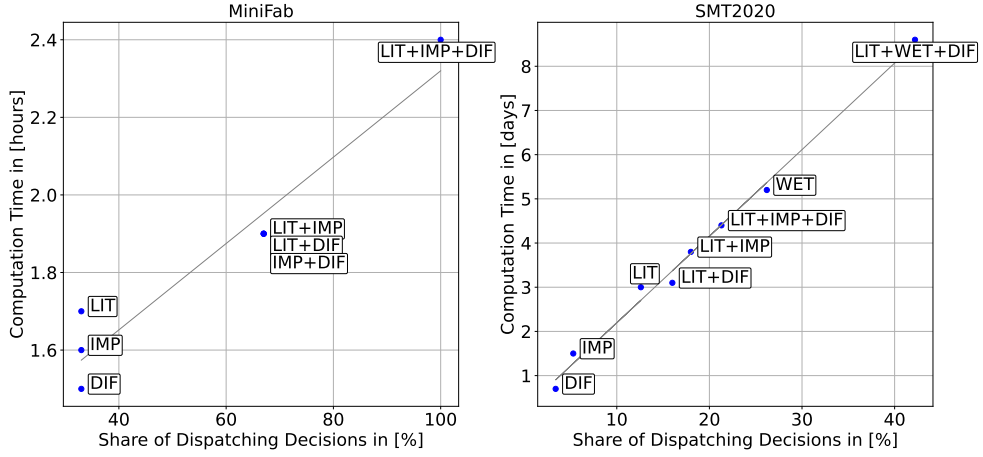


Fig. 9: Computation times for the MiniFab and SMT2020 models for 40 training episodes.

4.4 Computation Time

Different sizes of the simulation models lead to vastly different execution and training times, which are plotted in Figure 9. Besides the size of the simulation model, the number of RL-controlled tools is relevant. Each of those tools has to communicate through the interface with the simulator for making dispatching decisions, which requires the extraction of an observation and the computation of a dispatching decision using the NN. In fact, the plots show that scaling up the size leads to a growth of the execution time from hours to days. It has to be noted that the allocated memory scales linearly

with the number of used CPU cores, as we assign one independent simulator instance to each core. Since the communication overhead increases with more parallel processes, this can lead to a longer execution time for individual episodes. The computational cost can be a factor in deciding on the utilized resources. However, for productive use in semiconductor frontend WIP flow management, the optimization potential is enormous. Fabs cost millions of dollars per week, and increasing the utilization of these resources by even one percent amounts to a significant financial advantage.

4.5 Generalization

In order to investigate the generalization capabilities of the different approaches, we first define the two main sources of uncertainty. The first one includes the stochastic effects of the daily operations, such as unplanned machine breakdowns. This is a huge factor in semiconductor manufacturing as machines fail frequently. It is handled by modeling the failure behavior of the tools with distributions and drawing random values from that distribution based on the random seed for each simulation run.

Figure 10 shows the generalization properties for the MiniFab model, which was trained with CMA-ES on one random seed for the lithography tool only, and once for all tools. It shows that the agent is able to generalize with some loss of improvement if it controls enough tools. Training with more random seeds does not help in this case, as our experiments exhibited a smaller gap between training and testing but much worse training results when training on more random seeds. This could be due to the noise that is injected by the use of more random seeds.

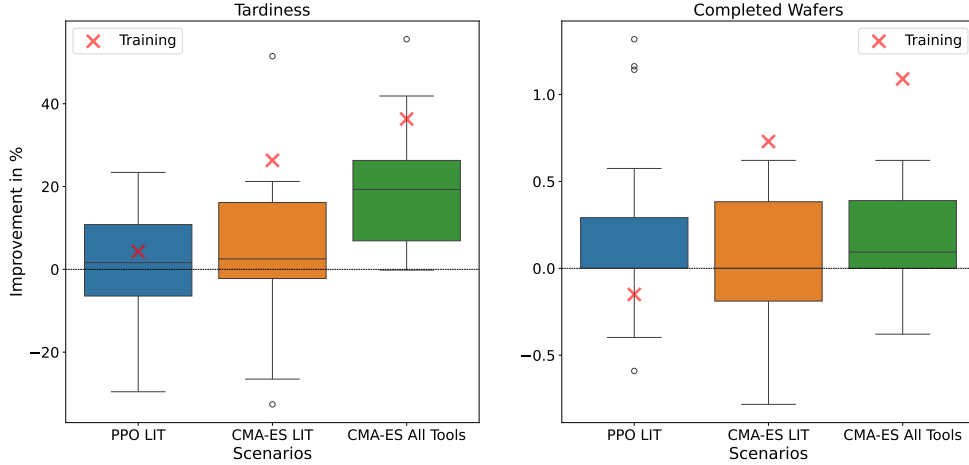


Fig. 10: Results for the testing of the CMA-ES experiments after 200 episodes, controlling lithography tools or all tools for the Minifab, as well as the PPO experiment controlling lithography tools. The red crosses indicate the result of the best parameter set of the last episode during the training for CMA-ES. For PPO, the red crosses indicates the mean of the training results for the last episode as it is only executing one parameter set in parallel but on different random seeds for the simulation.

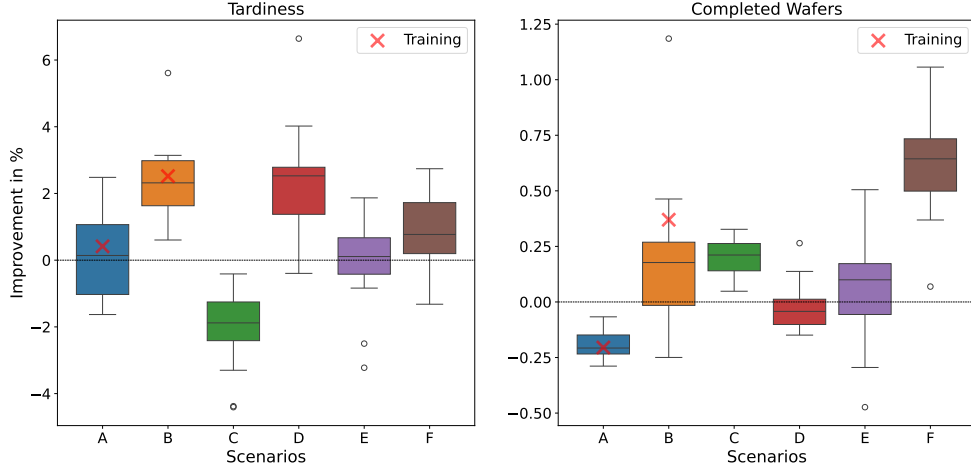


Fig. 11: Results for the testing of the CMA-ES experiments controlling lithography and WET tools for the industry model. The red crosses indicate the result of the best parameter set of the last episode during the training. The scenarios without red crosses were not included in the training.

For PPO, we already train on many different random seeds as we do not have to use our cores to test multiple perturbations of the NN parameters in parallel. For this reason, the strategy generalizes quite well on the test runs. If we train on only one random seed, the PPO delivers much worse performance. This could be because the noise of multiple seeds helps to not overfit on the reward of individual time intervals. However, the training and testing result is worse than for the CMA-ES approach. While more random seeds cannot be handled by the ES algorithm, PPO has no problems with the induced noise. This could be due to the fact that the PPO algorithm is not a black-box algorithm and utilizes the information collected from each decision step, not only from the KPIs achieved over the entire episode.

The second source of uncertainty is the ever-changing load mix. The volume and mix of products are changing over time and can only be estimated for the near future. This issue is only present in the SMT2020 and industry models, while the MiniFab model uses a cyclic pattern for the lot release.

In order to test how strategies found by the agent are generalizing for different load mixes, we test the CMA-ES strategy on multiple different loading scenarios. However, one could also decide to retrain a specific strategy for each new loading scenario in regular planning cycles. This would possibly lead to less robust but ideally more optimized strategies for the individual cases.

The test results for training the CMA-ES strategy with two scenarios and two random seeds each are shown in Figure 11, where the agent is controlling the lithography and WET areas. These results again indicate that the agent is generalizing with some loss of improvement. It can also be noted that the agent over-optimizes one training scenario more than the other. Furthermore, at least one scenario is always a bit worse

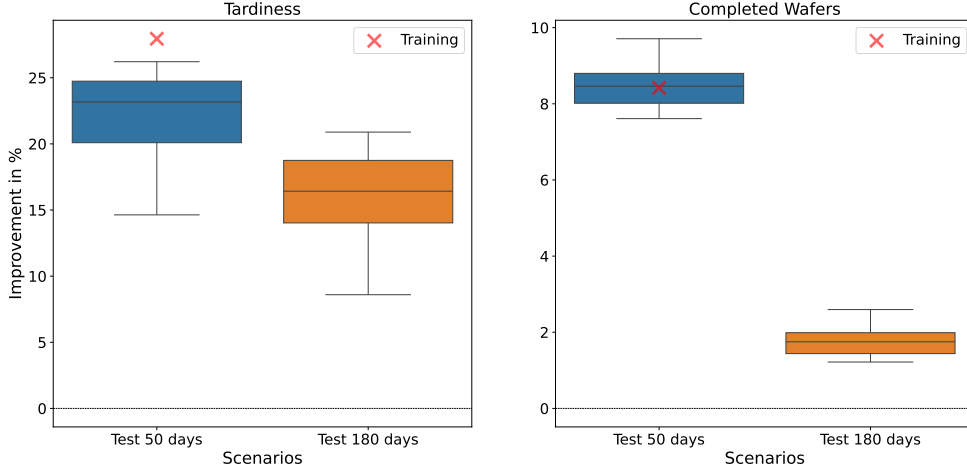


Fig. 12: Results for the testing of the CMA-ES experiments controlling all tools for the SMT2020 model. The red crosses indicate the result of the best parameter set of the last episode during the training, where the agent is trained on the 50-day scenario only.

than the reference for each metric. We suspect that these phenomena can be mitigated by training on more random seeds and scenarios in parallel.

SMT2020 does not provide different loading scenarios for the same production system under the same conditions. The models differ in the number of tools and the types of considered lots. Therefore we only test the trained model on the same scenario but on a longer time horizon (6 months instead of 50 days). This is useful as the loading changes over time.

In Figure 12, we observe that the CMA-ES strategy yields improvements with respect to the reference for both metrics, yet with some loss for the longer unseen scenario. This gap could be closed by training directly on the longer scenario, which would be computationally more expensive as each episode takes roughly four times longer to simulate. It has to be noted that, similar to the results for the MiniFab in Figure 10, SMT2020 also does not show good generalization properties if only one tool type is controlled by the agent.

5 Conclusion

While the amount of publications on RL in the context of semiconductor manufacturing is increasing every year, they substantially lack comparability. Different authors work on different model scenarios as well as different simulators, and utilize a variety of algorithms on diverse hardware. Thus it is hard to assess the scalability of an RL algorithm for dispatching found in the literature. In order to mitigate this shortcoming, we tested the scalability of an ES approach and a classical policy-gradient approach to optimize dispatching. We compared the performance of these algorithms across two public benchmark datasets of significantly different sizes and on an industry dataset.

The most important findings are the limited scalability of the PPO approach for this problem setting, which only shows an improvement for the Minifab model. ES scales well with an increased number of CPU cores. Furthermore, it is significantly harder to optimize real-world scenarios and the potential improvement is smaller. When switching between models, the algorithm has to be tuned regarding the hyper-parameters and the reward function. Lastly, the higher the percentage of tools under the control of the RL system, the higher is also the potential improvement. For the two open-source benchmark models Minifab and SMT2020, we achieve double-digit percentage improvement in tardiness and single digit percentage improvement in throughput. For the real industrial scale scenarios, we achieve an improvement of up to 4 % regarding tardiness and up to 1 % regarding throughput.

6 Limitations and Future Research

The use of policy updates with state-action-reward samples from complete episodes is shown to be more stable and yields better results than truncated samples in our study. However, it was computationally not possible with our setup to save all samples from an entire episode for the industrial scale scenario due to memory overflow. This is due to millions of samples which each have a big observation space. However, with more dedicated hardware resources, the PPO potentially would deliver more promising result for the industrial scale scenario. With more computing power and memory, we could also extend the experiments for the industrial scale model to utilize the RL-agent for all dispatching decisions, not only for bottleneck equipment groups. As different tools might require different dispatching strategies and the tool types are too many to efficiently encode in the observation space, the approach might have to be adapted to a multi-agent system with one policy per tool type.

As we have shown that CMA-ES is an alternative to policy gradient methods in SFSP and scales better with the complexity of the simulation model, the findings of this work can help to scale the training more effectively and thus reduce the overall computational cost.

Potential future areas of research include Explainable AI (XAI) to communicate the results to stakeholders like the responsible managers of the production facilities. Furthermore, there is a need for well-planned testing and validation steps to ensure the safety and generalization in the real production system outside of the simulation environment.

Statements & Declarations

Data availability statement. Due to commercial reasons, the real industry dataset is not available. However, the original dataset of the SMT2020 model [25] and descriptions for the MiniFab model [14, 55] are already available.

Funding. This work was partially funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) project 13IK033 (SmartMan) as well as the Austrian Research Promotion Agency (FFG) projects 894072 (SwarmIn) and FO999910235 (SAELING).

Competing Interests. The authors declare that they have no conflict of interest.

Author Contributions. P.S. developed the initial concept and wrote the main manuscript. P.S. and A.I. developed and implemented the method. This includes among others the neural network architecture, training procedure and environment, and the adaptation of the CMA-ES method to the use case. H.S. carried out the experiments, analyzed the results, validated and revised the implementations of the benchmark datasets. T.A. supervised the work from Infineon side and provided guidance regarding the manufacturing domain and methodology. M.W. contributed to the design of experiment for the comparison of the benchmark models and methodology for the literature search. M.G. and K.S. supervised the work from the academic side and provided theoretical guidance. G.S. provided guidance regarding the simulation and domain knowledge. C.W.C. implemented the benchmark datasets for the used simulator engine. C.W.C. and F.F.Z. provided support with the simulator and the simulator interface. All authors discussed the results and contributed to the final manuscript.

Appendix A Comparison of Training Results for PPO Rewards

We show the results for the best performing PPO reward $r_{\text{PPO},t} = r_{\text{D},t}$ in Section 4.2. In the following, we introduce additional rewards which we initially compared to find the best performing one. The rewards are based on different combinations of the tardiness and throughput of the overall fab, with tp_t being the number of completed wafers during the past 24 hours and WIP_t the number of wafers remaining in the system at time t , the end of the 24 hours interval. The tardiness is described by the tardiness $td_{\text{out},t}$ of the completed wafers within the past 24 h and the tardiness $td_{\text{in},t}$ of the wafers remaining in the system at time t . The results of the comparison are shown in Figure A1. The rewards are defined as follows:

$$r_{\text{A},t} = tp_t$$

$$r_{\text{B},t} = td_{\text{in},t}$$

$$r_{\text{C},t} = \frac{tp_t \cdot (\text{WIP}_t + tp_t)}{tp_t \cdot td_{\text{out},t} + \text{WIP}_t \cdot td_{\text{in},t}}$$

$$r_{\text{D},t} = \frac{tp_t}{(tp_t \cdot td_{\text{out},t} + \text{WIP}_t \cdot td_{\text{in},t}) \cdot (\text{WIP}_t + tp_t)}$$

In order to validate the approach using the rolling mean, we generate the results shown in Figure A2 as a comparison, for which we calculate the rewards not hourly over the past 24 hours but only over the past hour. It can be seen that the training is much more unstable as the reward is more noisy. However, for reward $r_{\text{B},t}$, it converges even faster. We suspect that this is due to the fact that td_{in} is a stable metric because it is calculated over the entire WIP, not only over the completed wafers. Especially for the rewards which are a combinations of multiple KPIs, the noisy feedback is a problem if the denominator approaches 0 in individual time intervals.

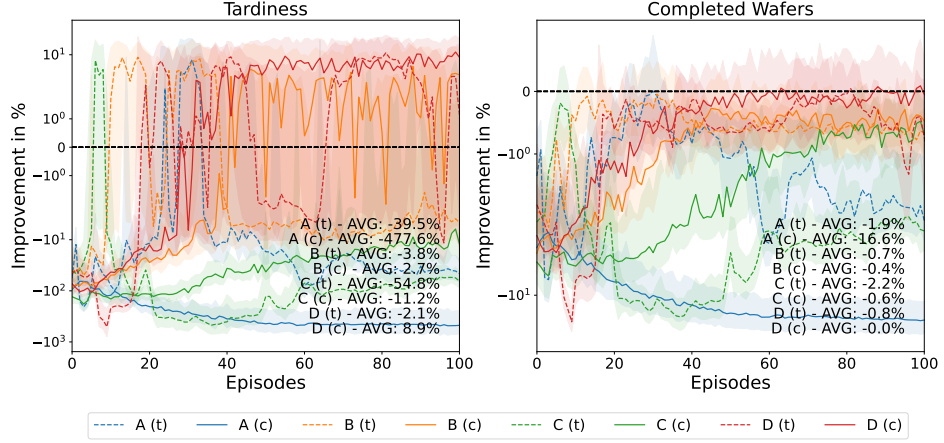


Fig. A1: Results for the PPO experiments with different rewards controlling the lithography area of the MiniFab model. The rewards are calculated hourly as a rolling mean of the previous 24 h. We differentiate between completed (c) and truncated (t) episodes used for the PPO policy updates. The truncation does not affect the simulated horizon but only the number of samples which is stored and used for the back-propagation.

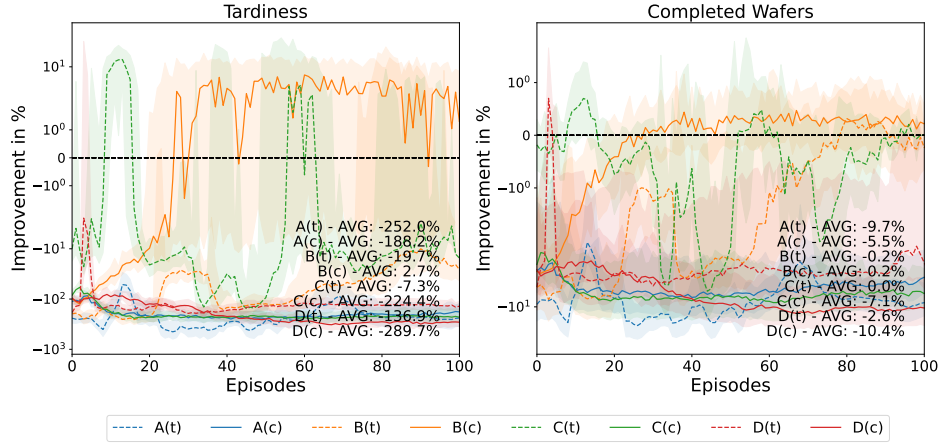


Fig. A2: Results for the PPO experiments with different rewards controlling the lithography area of the MiniFab model. The rewards are calculated hourly considering only the previous 60 min. We differentiate between completed (c) and truncated (t) episodes used for the PPO policy updates. The truncation does not affect the simulated horizon but only the number of samples which is stored and used for the back-propagation.

Appendix B Additional CMA-ES Training Results

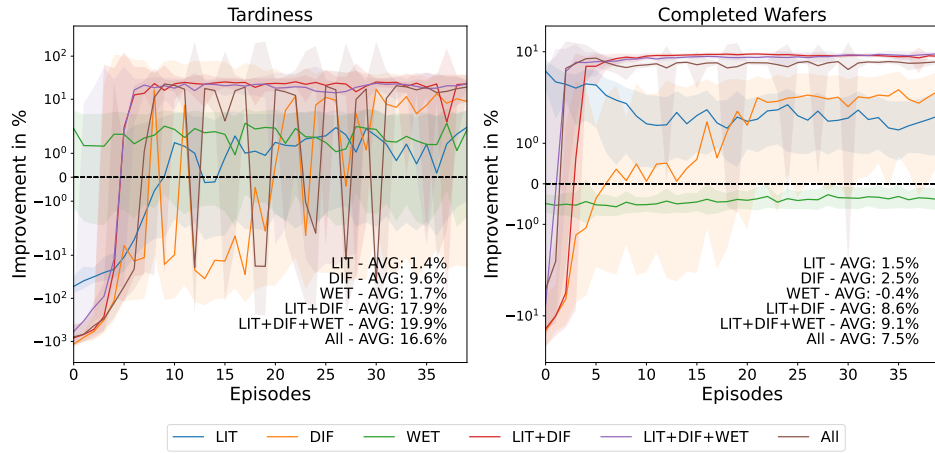


Fig. B3: Results for the CMA-ES experiments controlling different combinations of tools for the SMT2020 model.

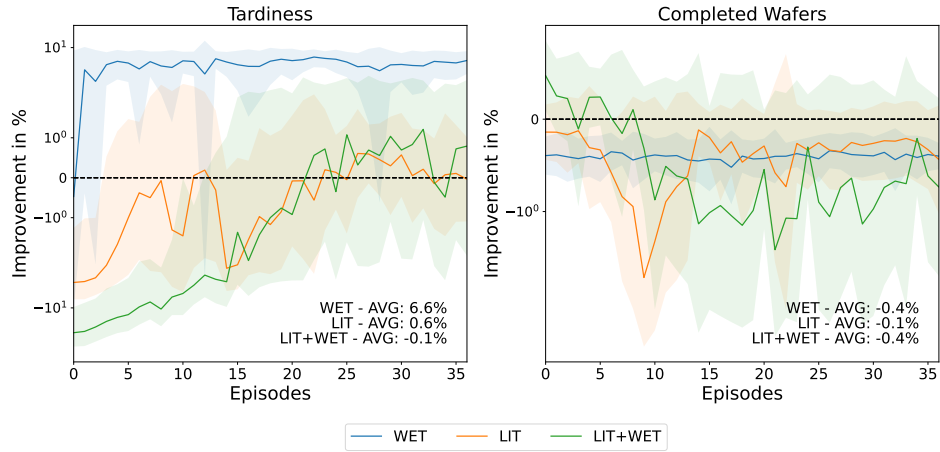


Fig. B4: Results for the CMA-ES experiments controlling different combinations of tools for the industry model.

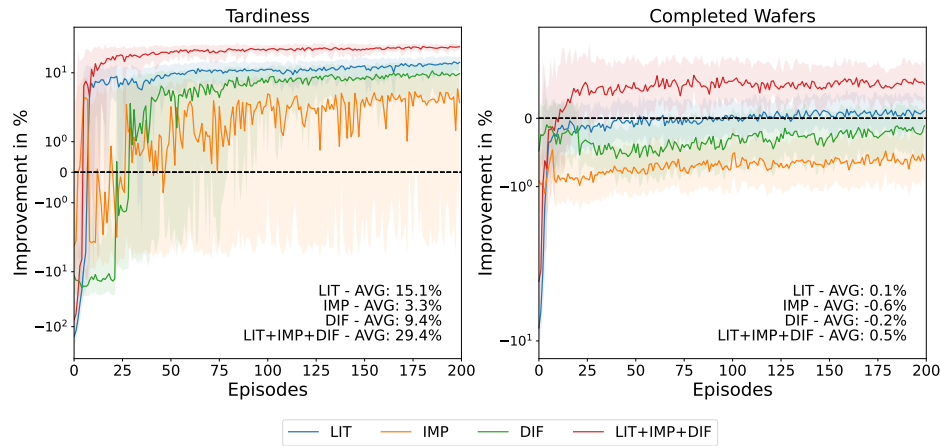


Fig. B5: Results for the CMA-ES experiments after 200 episodes, controlling different combinations of tools for the Minifab model.

References

- [1] Altenmüller T, Stüker T, Waschneck B, et al (2020) Reinforcement learning for an intelligent and autonomous production control of complex job-shops under time constraints. *Prod Eng Res Devel* 14. <https://doi.org/10.1007/s11740-020-00967-8>
- [2] Applegate DL, Cook WJ (1991) A computational study of the job-shop scheduling problem. *INFORMS J Comput* 3(2):149–156. <https://doi.org/10.1287/IJOC.3.2.149>
- [3] Bangsow S (2016) *Tecnomatix Plant Simulation*. Springer, Cham, Switzerland, <https://doi.org/10.1007/978-3-030-41544-0>
- [4] Bauer D, Umgelter D, Schlereth A, et al (2023) Complex job shop simulation “cojosim”—a reference model for simulating semiconductor manufacturing. *Applied Sciences* 13(6):3615. <https://doi.org/10.3390/app13063615>
- [5] Birgin EG, Feofiloff P, Fernandes CG, et al (2014) A MILP model for an extended version of the flexible job shop problem. *Optim Lett* 8(4):1417–1431. <https://doi.org/10.1007/S11590-013-0669-7>
- [6] Brucker P (2007) *Scheduling algorithms*, 5th edn. Springer, Berlin and New York, <https://doi.org/10.1007/10.1007/978-3-540-69516-5>
- [7] Campbell E, Ammenheuser J (2000) 300mm factory layout and material handling modeling: Phase ii report. SEMATECH Technical Transfer report

- [8] Chien C, Lan Y (2021) Agent-based approach integrating deep reinforcement learning and hybrid genetic algorithm for dynamic scheduling for industry 3.5 smart production. *Comput Ind Eng* 162:107782. <https://doi.org/10.1016/J.CIE.2021.107782>
- [9] D-SIMLAB Technologies (2023) Forecaster. URL <http://www.d-simlab.com/category/d-simcon/products-d-simcon/forecaster-and-scenario-manager/>
- [10] Da Col G, Teppan EC (2022) Industrial-size job shop scheduling with constraint programming. *Operations Research Perspectives* 9:100249. <https://doi.org/10.1016/j.orp.2022.100249>
- [11] Demirkol E, Mehta S, Uzsoy R (1998) Benchmarks for shop scheduling problems. *Eur J Oper Res* 109(1):137–141. [https://doi.org/10.1016/S0377-2217\(97\)00019-2](https://doi.org/10.1016/S0377-2217(97)00019-2)
- [12] Dong Z, Ren T, Qi F, et al (2024) A reinforcement learning-based approach for solving multi-agent job shop scheduling problem. *Int J Prod Res* pp 1–26. <https://doi.org/10.1080/00207543.2024.2423807>
- [13] El Adl MK, Rodriguez AA, Tsakalis KS (1996) Hierarchical modeling and control of re-entrant semiconductor manufacturing facilities. In: *Proceedings of 35th IEEE Conference on Decision and Control*, vol 2. IEEE, pp 1736–1742, <https://doi.org/10.1109/CDC.1996.572810>
- [14] El-Khouly IA, El-Kilany KS, El-Sayed AE (2009) Modelling and simulation of re-entrant flow shop scheduling: An application in semiconductor manufacturing. In: *2009 International Conference on Computers & Industrial Engineering*. IEEE, pp 211–216, <https://doi.org/10.1109/ICCIE.2009.5223754>
- [15] Elsevier (2025) Scopus. URL <https://www.scopus.com>, accessed: 2025-01-09
- [16] Fowler JW, Robinson J (1995) Measurement and improvement of manufacturing capacity (mimac) final report
- [17] Graham R, Lawler E, Lenstra J, et al (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer P, Johnson E, Korte B (eds) *Discrete Optimization II*, *Annals of Discrete Mathematics*, vol 5. Elsevier, Amsterdam, New York, Oxford, p 287–326, [https://doi.org/10.1016/S0167-5060\(08\)70356-X](https://doi.org/10.1016/S0167-5060(08)70356-X)
- [18] Gupta AK, Sivakumar AI (2004) Job shop scheduling techniques in semiconductor manufacturing. *The International Journal of Advanced Manufacturing Technology* 27(11–12):1163–1169. <https://doi.org/10.1007/s00170-004-2296-z>
- [19] Hansen N, Ostermeier A (2001) Completely derandomized self-adaptation in evolution strategies. *Evol Comput* 9(2):159–195. <https://doi.org/10.1162/106365601750190398>

- [20] van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: AAAI. AAAI Press, pp 2094–2100, <https://doi.org/10.1609/AAAI.V30I1.10295>
- [21] Karaboğa D (2005) An idea based on honey bee swarm for numerical optimization. <https://doi.org/10.1609/AAAI.V30I1.10295>
- [22] Kayton D, Teyner T, Schwartz C, et al (1997) Focusing maintenance improvement efforts in a wafer fabrication facility operating under the theory of constraints. *Production and Inventory Management Journal* 38(4):51–57
- [23] Klemmt A (2012) Ablaufplanung in der Halbleiter- und Elektronikproduktion. Vieweg+Teubner Verlag, Wiesbaden, Germany, <https://doi.org/10.1007/978-3-8348-1994-9>
- [24] Knopp S, Dauzère-Pérès S, Yugma C (2017) A batch-oblivious approach for complex job-shop scheduling problems. *Eur J Oper Res* 263(1):50–61. <https://doi.org/10.1016/J.EJOR.2017.04.050>
- [25] Kopp D, Hassoun M, Kalir A, et al (2020) Smt2020—a semiconductor manufacturing testbed. *IEEE Transactions on Semiconductor Manufacturing* 33(4):522–531. <https://doi.org/10.1109/TSM.2020.3001933>
- [26] Kovacs B, Tassel P, Ali R, et al (2022) A customizable simulator for artificial intelligence research to schedule semiconductor fabs. In: ASMC. IEEE, pp 1–6, <https://doi.org/10.1109/ASMC54647.2022.9792520>
- [27] Laborie P, Rogerie J, Shaw P, et al (2018) Ibm ilog cp optimizer for scheduling: 20+ years of scheduling with constraints at ibm/ilog. *Constraints* 23. <https://doi.org/10.1007/S10601-018-9281-X>
- [28] Lee W, Kim B, Ko K, et al (2019) Simulation based multi-objective fab scheduling by using reinforcement learning. In: WSC. IEEE, pp 2236–2247, <https://doi.org/10.1109/WSC40007.2019.9004886>
- [29] Liang E (2021) Scalable reinforcement learning systems and their applications. PhD thesis, University of California, Berkeley, USA
- [30] Liang E, Liaw R, Nishihara R, et al (2018) Rllib: Abstractions for distributed reinforcement learning. In: ICML, Proceedings of Machine Learning Research, vol 80. PMLR, pp 3059–3068
- [31] Liao Z, Chen J, Zhang Z (2023) Solving job-shop scheduling problem via deep reinforcement learning with attention model. In: IEA/AIE (2), Lecture Notes in Computer Science, vol 13926. Springer, pp 201–212, <https://doi.org/10.1007/978-3-031-36822-6.18>

- [32] Lin C, Cao Z, Zhou M (2022) Learning-based grey wolf optimizer for stochastic flexible job shop scheduling. *IEEE Trans Autom Sci Eng* 19(4):3659–3671. <https://doi.org/10.1109/TASE.2021.3129439>
- [33] Lin C, Cao Z, Zhou M (2023) Learning-based cuckoo search algorithm to schedule a flexible job shop with sequencing flexibility. *IEEE Trans Cybern* 53(10):6663–6675. <https://doi.org/10.1109/TCYB.2022.3210228>
- [34] Liu J, Qiao F, Zou M, et al (2022) Dynamic scheduling for semiconductor manufacturing systems with uncertainties using convolutional neural networks and reinforcement learning. *Complex & Intelligent Systems* 8(6):4641–4662. <https://doi.org/10.1007/s40747-022-00844-0>
- [35] Lu S, Wang Y, Kong M, et al (2024) A double deep q-network framework for a flexible job shop scheduling problem with dynamic job arrivals and urgent job insertions. *Eng Appl Artif Intell* 133:108487. <https://doi.org/10.1016/J.ENGAPPAI.2024.108487>
- [36] Ma Y, Cai J, Li S, et al (2023) Double deep q-network-based self-adaptive scheduling approach for smart shop floor. *Neural Comput Appl* 35(30):22281–22296. <https://doi.org/10.1007/S00521-023-08877-3>
- [37] Mason SJ, Fowler JW, Carlyle WM (2022) A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. *Journal of Scheduling* <https://doi.org/10.1002/jos.102>
- [38] Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. *Adv Eng Softw* 69:46–61. <https://doi.org/10.1016/J.ADVENGSOFT.2013.12.007>
- [39] Mnih V, Kavukcuoglu K, Silver D, et al (2013) Playing atari with deep reinforcement learning. *CoRR* abs/1312.5602
- [40] Mnih V, Badia AP, Mirza M, et al (2016) Asynchronous methods for deep reinforcement learning. In: *ICML, JMLR Workshop and Conference Proceedings*, vol 48. JMLR.org, pp 1928–1937
- [41] Mönch L, Fowler JW, Dauzère-Pérès S, et al (2011) A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. *J Sched* 14(6):583–599. <https://doi.org/10.1007/s10951-010-0222-9>
- [42] Mönch L, Fowler JW, Mason SJ (2013) *Production Planning and Control for Semiconductor Wafer Fabrication Facilities - Modeling, Analysis, and Systems, Operations research / computer science interfaces series*, vol 52. Springer, Cham, Switzerland, <https://doi.org/10.1007/978-1-4614-4472-5>
- [43] Park I, Huh J, Kim J, et al (2020) A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities. *IEEE Trans Autom Sci Eng*

- 17(3):1420–1431. <https://doi.org/10.1109/TASE.2019.2956762>
- [44] Perron L, Didier F (2023) Cp-sat. URL <https://developers.google.com/optimization/cp/cp-solver>
 - [45] Phillips T (1998) AUTOSCHED AP by autosimulations. In: WSC. WSC, pp 219–222, <https://doi.org/10.1109/WSC.1998.744926>
 - [46] Pinedo M (2005) Planning and scheduling in manufacturing and services. Springer series in operations research, Springer, New York, NY, USA, <https://doi.org/10.1007/978-1-4419-0910-7>
 - [47] Pinedo ML (2012) Scheduling: Theory, Algorithms, and Systems. Springer US, New York, NY, USA, <https://doi.org/10.1007/978-3-031-05921-6>
 - [48] Sakr AH, AboElHassan A, Yacout S, et al (2023) Simulation and deep reinforcement learning for adaptive dispatching in semiconductor manufacturing systems. J Intell Manuf 34(3):1311–1324. <https://doi.org/10.1007/S10845-021-01851-7>
 - [49] Salimans T, Ho J, Chen X, et al (2017) Evolution strategies as a scalable alternative to reinforcement learning. CoRR abs/1703.03864
 - [50] Scherfke S, Lünsdorf O (2013) Simpy. URL <https://simpy.readthedocs.io/en/latest/>
 - [51] Schulman J, Levine S, Abbeel P, et al (2015) Trust region policy optimization. In: ICML, JMLR Workshop and Conference Proceedings, vol 37. JMLR.org, pp 1889–1897
 - [52] Schulman J, Wolski F, Dhariwal P, et al (2017) Proximal policy optimization algorithms. CoRR abs/1707.06347
 - [53] Shao H, Ge J, Wang G (2024) Attention assigning: Stacked dual network-based reinforcement learning for solving assignment problems. In: 2024 5th International Conference on Electronic Communication and Artificial Intelligence (ICECAI), pp 713–716, <https://doi.org/10.1109/ICECAI62591.2024.10675000>
 - [54] Shiue Y, Lee K, Su C (2020) A reinforcement learning approach to dynamic scheduling in a product-mix flexibility environment. IEEE Access 8:106542–106553. <https://doi.org/10.1109/ACCESS.2020.3000781>
 - [55] Spier J, Kempf K (1995) Simulation of emergent behavior in manufacturing systems. In: ASMC. IEEE, pp 90–94, <https://doi.org/10.1109/ASMC.1995.484347>
 - [56] Stöckermann P, Immordino A, Altenmüller T, et al (2023) Dispatching in real frontend fabs with industrial grade discrete-event simulations by deep reinforcement learning with evolution strategies. In: WSC. IEEE, pp 3047–3058,

<https://doi.org/10.1109/WSC60868.2023.10408625>

- [57] Stricker N, Kuhnle A, Sturm R, et al (2018) Reinforcement learning for adaptive order dispatching in the semiconductor industry. *CIRP Annals* 67(1):511 – 514
- [58] Sutton RS, Barto AG (1998) Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning Series, Massachusetts Institute of Technology Press, Cambridge, Massachusetts
- [59] Taillard E (1993) Benchmarks for basic scheduling problems. *Eur J Oper Res* 64(2):278–285. [https://doi.org/10.1016/0377-2217\(93\)90182-M](https://doi.org/10.1016/0377-2217(93)90182-M)
- [60] Tassel P, Gebser M, Schekotihin K (2021) A reinforcement learning environment for job-shop scheduling. *CoRR* abs/2104.03760
- [61] Tassel P, Kovács B, Gebser M, et al (2023) Semiconductor fab scheduling with self-supervised and reinforcement learning. In: WSC. IEEE, pp 1924–1935, <https://doi.org/10.1109/WSC60868.2023.10407747>
- [62] Vaswani A, Shazeer N, Parmar N, et al (2017) Attention is all you need. In: NIPS, pp 5998–6008
- [63] Wang J, He J, Zhang J (2018) A reinforcement learning method to optimize the priority of product for scheduling the large-scale complex manufacturing systems. In: Proceedings of International Conference on Computers and Industrial Engineering, CIE
- [64] Wang M, Zhang J, Zhang P, et al (2025) Cooperative multi-agent reinforcement learning for multi-area integrated scheduling in wafer fabs. *Int J Prod Res* 63(8):2871–2888. <https://doi.org/10.1080/00207543.2024.2411615>
- [65] Waschneck B, Reichstaller A, Belzner L, et al (2018) Deep reinforcement learning for semiconductor production scheduling. In: ASMC. IEEE, pp 301–306, <https://doi.org/10.1109/ASMC.2018.8373191>
- [66] Waschneck B, Reichstaller A, Belzner L, et al (2018) Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP* 72:1264–1269. <https://doi.org/10.1016/j.procir.2018.03.212>
- [67] Webster J, Watson RT (2002) Analyzing the past to prepare for the future: Writing a literature review. *MIS Q* 26(2)
- [68] Wu J, Chien C (2008) Modeling semiconductor testing job scheduling and dynamic testing machine configuration. *Expert Syst Appl* 35(1-2):485–496. <https://doi.org/10.1016/j.eswa.2007.07.026>
- [69] Wu J, Hao X, Chien C, et al (2012) A novel bi-vector encoding genetic algorithm for the simultaneous multiple resources scheduling problem. *J Intell Manuf*

23(6):2255–2270. <https://doi.org/10.1007/S10845-011-0570-0>

- [70] Yang X, Deb S (2009) Cuckoo search via lévy flights. <https://doi.org/10.1109/NABIC.2009.5393690>
- [71] Yedidsion H, Dawadi P, Norman D, et al (2022) Deep reinforcement learning for queue-time management in semiconductor manufacturing. In: WSC. IEEE, pp 3275–3284, <https://doi.org/10.1109/WSC57314.2022.10015463>
- [72] Yun S, Jeong M, Kim R, et al (2019) Graph transformer networks. In: NeurIPS, pp 11960–11970
- [73] Zhang L, Lin Y, Xu C, et al (2024) A new EDA algorithm combined with q-learning for semiconductor final testing scheduling problem. Comput Ind Eng 193:110259. <https://doi.org/10.1016/J.CIE.2024.110259>