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

FABSIM: A MICRO-DISCRETE EVENT SIMULATOR FOR MACHINE LEARNING DYNAMIC JOB SHOP SCHEDULING OPTIMIZERS

Sean Mondesire[1], Michael Brown[1], and Bulent Soykan[2]

[1]School of Modeling, Simulation, and Training, University of Central Florida, Orlando, FL, USA
[2]Institute for Simulation and Training, University of Central Florida, Orlando, FL, USA

ABSTRACT

Dynamic job shop scheduling (DJSS) demands rapid, data-driven decisions to balance conflicting objectives in complex, stochastic manufacturing environments. While Artificial Intelligence, particularly deep reinforcement learning (RL), offers powerful optimization capabilities, its development and evaluation are often limited by the lack of suitable high-speed, flexible simulation testbeds. To address this, we introduce FabSim, a micro-discrete-event simulation (micro-DES) environment purpose-built for developing and evaluating intelligent scheduling strategies for DJSS. FabSim models core scheduling dynamics, including resource contention, flexible routing, batching, and stochastic events, using an efficient event-driven kernel and indexed look-up tables that enable sub-second simulation runtimes. Compliant with the Farama Gymnasium API, it integrates seamlessly with standard machine learning libraries. FabSim facilitates reproducible benchmarking and serves as a practical back-end for digital twin applications requiring near-real-time analysis. This paper details FabSim's design, validates its high-speed performance, and demonstrates its utility for AI-driven scheduling research.

1 INTRODUCTION

Effective machine scheduling is a cornerstone of efficient operations in numerous complex manufacturing and service systems, impacting throughput, resource utilization, cost-effectiveness, and responsiveness (Pinedo 2012; Rinnooy Kan 2012). Dynamic job shop scheduling (DJSS), in particular, demands rapid, data-driven decisions to balance these objectives across ever-changing production lines (Ouelhadj and Petrovic 2009). However, deriving optimal or near-optimal schedules is notoriously difficult. Many scheduling problems are $\mathcal{NP}$-hard, involving combinatorial complexity that grows rapidly with problem size (Garey et al. 1976; Lenstra et al. 1977). Real-world systems often exhibit further complexities such as high job mix, flexible routing including re-entrance, batch processing requirements, sequence-dependent setups, and significant stochasticity from unforeseen events like machine breakdowns or processing time variations. Effectively navigating this confluence of combinatorial complexity, intricate constraints, and dynamic uncertainty makes DJSS an exceptionally demanding optimization challenge critical to operational performance.

These inherent characteristics pose significant challenges for traditional scheduling methodologies. Simple priority-based dispatching rules, while computationally cheap and ubiquitous in practice, operate on limited local information. Their myopic nature often leads to globally suboptimal performance, failing to anticipate bottlenecks or effectively manage complex constraints, particularly in dynamic settings (Pinedo 2012). Conversely, exact optimization methods such as Mixed-Integer Linear Programming (MILP) or Constraint Programming (CP), while capable of finding optimal solutions for smaller, deterministic problems, struggle with the scale and stochasticity of many real-world scheduling scenarios. The computational effort required often becomes prohibitive for operational decision-making. Simulation-based approaches, including coupling discrete-event simulation with search heuristics, can model system dynamics and uncertainty with higher fidelity but may require significant computation time to find high-quality solutions and still necessitate effective control logic to guide the search or operational decisions. Thus, there remains a need for scheduling

approaches that can effectively handle the scale, complexity, and dynamic nature of modern scheduling environments.

Reinforcement Learning (RL) presents a compelling paradigm for tackling these challenges (Sutton and Barto 2018). By learning through direct interaction with the system or a simulation, an RL agent can autonomously develop sophisticated control policies. It observes the system state, selects actions (e.g., assigning a job to a machine), and receives feedback via a reward signal aligned with scheduling objectives (such as minimizing cycle time or maximizing throughput). Progress in AI, particularly deep RL, has outpaced the availability of flexible, high-speed test beds for training and comparison. RL's potential to learn complex, adaptive policies capable of handling stochasticity and optimizing long-term goals makes it a highly promising avenue for complex machine scheduling, but realizing this potential hinges on suitable development platforms.

However, applying RL effectively requires appropriate simulation environments. While general RL frameworks (e.g., Gymnasium (Towers et al. 2024)) and some environments for Operations Research (e.g., OR-Gym (Hubbs et al. 2020), RL4CO (Berto et al. 2023)) or generic production settings (e.g., FabricatioRL (Rinciog and Meyer 2021), PySCFabSim (Kovács et al. 2022)) exist, they often lack the necessary combination of speed, detail, and flexibility for rigorous DJSS research. Many existing tools struggle with configuration rigidity, exhibit slow run times unsuitable for the millions of samples needed in deep RL training, oversimplify critical scheduling constraints (like detailed batching or setups), lack mechanisms for real-time policy interaction, or offer poor integration with standard machine learning pipelines. There is a need for lightweight, fast, standardized, and sufficiently representative simulation environments for DJSS.

To address this gap, this paper introduces *FabSim*: a micro-discrete-event simulation (micro-DES) environment purpose-built for developing and evaluating intelligent scheduling strategies, especially RL agents, in generalized DJSS settings. FabSim serves as the primary contribution, providing a flexible, high-speed platform that encapsulates common scheduling challenges. Key characteristics include: (i) *Efficient Micro-DES Core:* An event-driven engine with indexed look-up tables eliminates idle looping, yielding sub-second runtimes suitable for large-scale experiments and digital twin back-ends. (ii) *DJSS Dynamics Modeling:* Represents jobs, operations, specialized machines (single/batch), stochastic failures, calendars, and setup constraints. (iii) *Standardized RL Interface:* Full compliance with the Farama Gymnasium API allows seamless integration with mainstream RL libraries. (iv) *Configurability and Reproducibility:* Supports dynamic arrivals, priorities, user-defined stochastic models, and ensures reproducible results via random number generator (RNG) seeding and configuration management.

This paper contributes (i) the formal design of FabSim's abstraction and event kernel, (ii) benchmark results showing millisecond-scale step times on large job sets, and (iii) illustrative experiments with heuristic and RL agents. The central research question is: *How can a standardized, reproducible, fast, and sufficiently detailed micro-DES environment be designed to effectively facilitate the development, training, and benchmarking of AI-driven optimizers for DJSS problems?* By detailing FabSim's design and validation, and releasing it to the community, we aim to accelerate AI research for DJSS and lower the barrier to deploying intelligent schedulers in complex, real-world operations.

The remainder of this paper is structured as follows. Section 2 provides further background on DJSS and related work. Section 3 presents the detailed design and implementation of FabSim. Section 4 details experimental validation. Finally, Section 5 discusses the findings, limitations, and concludes the paper.

2 BACKGROUND AND RELATED WORK

This section provides context for FabSim by first formalizing the machine scheduling problem, discussing traditional solution approaches, reviewing the application of reinforcement learning to scheduling, and finally surveying existing simulation environments for RL in this domain, highlighting the gap FabSim aims to fill.

2.1 The Machine Scheduling Problem

Machine scheduling problems involve the allocation of limited resources (machines) over time to perform a set of tasks (operations belonging to jobs), subject to various constraints, with the goal of optimizing one or more performance objectives (Pinedo 2012). A common and challenging variant is the Job Shop Scheduling Problem (JSSP), where a set of N jobs, $J = \{j_1, j_2, ..., j_N\}$, must be processed on a set of M machines, $M = \{m_1, m_2, ..., m_M\}$. Each job j_i consists of a specific sequence of operations $O_i = (o_{i1}, o_{i2}, ..., o_{in_i})$, where each operation o_{ik} requires a specific machine $m(o_{ik}) \in M$ for a certain processing time p_{ik}. Precedence constraints dictate that o_{ik} must be completed before $o_{i,k+1}$ can start. A machine can only process one operation at a time, and operations are typically non-preemptive. The Flexible Job Shop Scheduling Problem (FJSSP) extends this by allowing an operation to be processed by any machine from a set of eligible machines, often with different processing times (Brandimarte 1993). The DJSS further incorporates real-world complexities such as unexpected job arrivals, machine breakdowns, varying processing times, and shifting priorities, making it highly relevant to modern manufacturing and service operations.

Common objectives in machine scheduling include minimizing makespan (C_{max}), mean flow time, and tardiness, or maximizing throughput and resource utilization. Attaining these goals is significantly complicated by factors prevalent in many complex production settings. These include managing a high mix of jobs with diverse and potentially lengthy processing routes on shared resources. Routes may involve revisiting machine groups (re-entrance), creating complex dependencies (Kayhan and Yildiz 2023). Further challenges arise from batch processing requirements involving intricate formation rules (balancing utilization and wait times) (Mönch et al. 2011), sequence-dependent setup times adding combinatorial difficulty (Allahverdi et al. 2008), general resource constraints (machines, buffers, tools, operators) creating bottlenecks, and pervasive stochasticity stemming from uncertain job arrivals, machine breakdowns, and variable processing times, demanding robust and adaptive scheduling solutions (Aytug et al. 2005). The confluence of these elements renders many machine scheduling problems $\mathscr{NP}$-hard and particularly challenging to solve optimally in dynamic, real-world environments.

Various approaches have been developed to tackle machine scheduling problems. *Dispatching rules* are simple heuristics (e.g., Shortest Processing Time - SPT, First-In-First-Out - FIFO, Earliest Due Date - EDD) used extensively in practice for making real-time decisions at individual machines (Pinedo 2012). Their main advantages are low computational cost and ease of implementation. However, they are typically myopic, relying on local information and often leading to globally suboptimal performance, especially in complex systems with interacting constraints or significant stochasticity (Holthaus and Rajendran 1997).

Mathematical optimization techniques, such as MILP or CP, provide frameworks for finding provably optimal solutions (Maravelias 2006). While powerful for modeling constraints, these exact methods often face severe scalability limitations due to the combinatorial nature of scheduling problems. Solving industrial-scale instances to optimality is frequently computationally intractable for operational use, restricting their application mainly to smaller problems or offline planning with simplified models. Capturing stochastic elements within these deterministic frameworks also poses difficulties.

Metaheuristics, including Genetic Algorithms (Lee et al. 1998), Tabu Search (Laguna et al. 1991), and Simulated Annealing (Kim et al. 2002), represent a class of approximate optimization algorithms that search the solution space more intelligently than simple heuristics. They can often find high-quality solutions for complex problems where exact methods fail. However, they typically require significant computational time, can be sensitive to parameter tuning, and may still struggle to adapt quickly to highly dynamic changes or real-time events compared to dispatching rules.

Discrete-event simulation is a powerful tool for modeling the dynamics and stochasticity of manufacturing systems (Kampa et al. 2017). It allows for detailed "what-if" and "what-next" analysis, evaluating the performance of different dispatching rules or system configurations. However, simulation itself does not optimize; it requires an external mechanism—be it a human analyst, a search heuristic, or an intelligent agent—to propose and evaluate different control strategies to find improved schedules. This highlights the need for integrating intelligent decision-making within simulation models.

2.2 Reinforcement Learning for Scheduling

RL has emerged traction as a method to develop adaptive scheduling policies by learning from interaction (Kayhan and Yildiz 2023). An RL agent learns a policy (π) mapping system states (s) to actions (a) (e.g., which job to dispatch) to maximize a cumulative reward (R) tied to scheduling objectives. This learning framework is well-suited to dynamic and stochastic environments where optimal rules are not easily derived.

Early applications often used tabular methods (e.g., Q-learning) for smaller problems. The integration with deep learning (Deep RL) has enabled scaling to more complex scenarios (Mnih et al. 2015). Value-based methods such as Deep Q-Networks (DQN) (Luo et al. 2021) learn the expected value of actions, while policy gradient methods such as REINFORCE or Actor-Critic (Liu et al. 2020; Waschneck et al. 2018) directly optimize the policy parameters. Various state representations (vectors, images, graphs using GNNs (Zhang et al. 2020)) and action space formulations (selecting jobs, selecting rules) have been explored. Much research focuses on learning effective dispatching policies for dynamic job shops or flexible manufacturing systems, often demonstrating superior performance compared to static dispatching rules, especially for multi-objective criteria or under uncertainty (Rinciog and Meyer 2021).

2.3 Existing Simulation Environments for RL in Scheduling

The successful application of RL to complex machine scheduling problems heavily relies on the availability of suitable simulation environments for agent training, validation, and benchmarking. While several platforms have been developed to facilitate research in RL for scheduling and related optimization domains, they often exhibit limitations when considering the specific demands of dynamic, feature-rich job shop scheduling, particularly regarding the balance between fidelity, speed, standardization, and flexibility.

Several open frameworks specifically target production or job shop scheduling for RL. **FabricatioRL** (Rinciog and Meyer 2021) offers a modular, reproducible environment based on SimPy and the Gymnasium API, suitable for general production settings including FJSSP. Similarly, **PySCFabSim** (Kovács et al. 2022) adapts the Proximal Policy Optimization (PPO) algorithm for JSSP, demonstrating policy generalization under processing constraints. These platforms provide valuable baselines and focus on RL integration. However, their reliance on the high-level abstractions within traditional DES engines (such as SimPy's Process and Resource classes) can limit execution speed compared to more specialized approaches. While powerful for general modeling, these abstractions introduce overhead not suitable for the millions of simulation steps required in deep RL training (Soykan and Rabadi 2023; Soykan and Rabadi 2024). This distinction motivates a micro-DES approach (Nsiye et al. 2024), which prioritizes computational performance for this specific application domain. Also, while configurable, they might require extensions for highly specific machine types or dynamic event handling beyond basic stochastic inputs, and their support for live policy updates or real-time interaction may be limited compared to environments designed with that capability in mind.

Other relevant libraries include **OR-Gym** (Hubbs et al. 2020) and the more recent, comprehensive **RL4CO** (Berto et al. 2023). These platforms offer standardized Gymnasium environments for a wide array of Operations Research (OR) and Combinatorial Optimization (CO) problems, making them valuable for benchmarking RL algorithms on fundamental tasks like vehicle routing or packing. Their breadth and focus on optimization benchmarks are strengths. However, they typically lack dedicated, high-fidelity simulation models for complex end-to-end machine scheduling processes that incorporate detailed operational logic like batching, sequence-dependent setups, or specific resource interactions found in factory environments.

Also, various **domain-specific simulation environments** have been developed within individual research projects. Examples include tools tailored for specific industries like semiconductor manufacturing (Kuhnle et al. 2022) or those focused on particular problems like the JSSP, such as **PySCFabSim** (Kovács et al. 2022), which successfully adapts PPO and demonstrates generalization. While these custom environments can offer high realism for their target application, they often face limitations that hinder broader adoption and comparative research. These can include a lack of adherence to standardized APIs like Gymnasium

(making integration with RL libraries difficult), limited public code availability or ongoing maintenance, potential reproducibility issues if not carefully designed, or a narrow focus that limits generalizability to other scheduling contexts or variations. The underlying simulation approach (e.g., using SimPy vs. a custom kernel) also significantly impacts performance and suitability for large-scale RL.

Consequently, while progress has been made, there remains a need for accessible, standardized, yet sufficiently flexible and detailed simulation environments specifically designed for complex machine scheduling. Such an environment should allow easy configuration of various JSSP/FMS characteristics (machines, routes, batching, setups, stochasticity), adhere to standards such as Gymnasium for broad compatibility, and facilitate reproducible research, bridging the gap between general RL platforms and the specific needs of machine scheduling research. FabSim is developed to address this specific need.

3 FABSIM: DESIGN AND IMPLEMENTATION

FabSim is designed as a flexible and extensible discrete-event simulation environment for applying RL to machine scheduling problems. Its development was guided by the need for a standardized tool that captures essential scheduling complexities while ensuring ease of use and reproducibility. This section details the core requirements, conceptual model, RL formulation, software architecture, and implementation specifics.

3.1 Core Requirements

Derived from the analysis of machine scheduling challenges and existing environment limitations (Section 2), the fundamental requirements for FabSim are:

- **Modeling Core Scheduling Dynamics:**
  - *Flexible Job Routing:* Support for complex, multi-step process routes for different job types, including the possibility of jobs revisiting machine groups (re-entrance or flexible flow paths).
  - *Diverse Machine Types:* Ability to model both single-job processing machines and batch processing machines with configurable rules (e.g., minimum/maximum batch size, compatibility constraints, time limits).
  - *Sequence-Dependent Setups:* Capability to model machine setup times where the duration depends on the sequence of operations processed.
  - *(Extensible) Resource Constraints:* Foundational support for primary machine constraints, with extensibility for auxiliary resources (e.g., machines, operators).
- **Representation of Stochasticity:** Incorporate key sources of operational uncertainty, including stochastic *machine breakdowns* (failures and repairs) and *processing time variability*.
- **Standardized Reinforcement Learning Interface:** Strict adherence to the *Gymnasium API standard* (Towers et al. 2024), ensuring compatibility with standard RL libraries through well-defined observation/action spaces and `step`/`reset` methods.
- **Configurability for Research Flexibility:** Enable users to easily define and modify system configurations (machines, buffers, process plans, job characteristics), simulation parameters, stochastic levels, and RL reward functions, preferably via external configuration files.
- **Reproducibility for Scientific Rigor:** Implement mechanisms for reproducible stochasticity via robust RNG seeding for both instance generation and simulation dynamics, facilitating reliable experiment replication and comparison.

3.2 Conceptual Model of the Manufacturing System

FabSim employs a discrete-event simulation based on a conceptual model of a general manufacturing system, focusing on job flow and resource allocation dynamics. The model's scope typically includes job processing operations but abstracts details such as material handling specifics unless explicitly configured. Key entities are: (i) **Job:** The unit of work flowing through the system (analogous to lots). Jobs possess

attributes such as ID, type, current operation, location, priority, due date, and route. (ii) **Operation (Process Step):** A task within a job's route, requiring a specific machine capability (machine group) and having associated processing time characteristics, setup needs, and batching rules. (iii) **Machine (Resource):** A processing unit with states (idle, processing, setup, down, maintenance). Machines belong to **Machine Groups** based on shared capabilities. (iv) **Buffer:** Queues preceding machine groups where jobs wait. Key scheduling concepts are represented as follows: Job routes are defined as sequences of operations, allowing flexible paths including revisits to machine groups. Machines are defined with their type (single/batch) and capabilities. Batching rules (size, time constraints, compatibility) and sequence-dependent setup times (from a matrix) are associated with machines or operations and handled by the simulation logic.

3.3 Reinforcement Learning Environment Formulation

The scheduling problem within FabSim is formulated as a Markov decision process (MDP) (S, A, R, T) for RL agent interaction:

State Space (S): The observation provided to the agent typically includes configurable features such as machine status (idle, busy, down, setup configuration), buffer levels (queue lengths, waiting times), and attributes of waiting jobs (current step, remaining steps, priority, due date, time-in-queue). Global metrics such as WIP or time can also be included. The representation aims to be informative yet manageable.

Action Space (A): The agent makes decisions at discrete events (e.g., machine becomes idle). Actions are typically discrete and may include selecting which job to dispatch from a buffer to an idle machine, or deciding whether to start a batch process. Action masking ensures only valid actions (available jobs compatible with the machine) are selectable.

Reward Function (R): Configurable reward signals guide learning towards objectives such as minimizing makespan, minimizing mean cycle time/flow time, maximizing throughput, or minimizing tardiness. Both sparse (end-of-episode) and dense (step-based) rewards can be implemented, often incorporating penalties for waiting time or setups and bonuses for job completions.

State Transition Logic (T): Governed by the underlying SimPy-based discrete-event simulation engine. The simulation clock advances based on events (job arrivals, operation completions, machine breakdowns/repairs, setup completions). Agent actions schedule future events (e.g., operation completion based on sampled processing time). Stochastic events are triggered based on configured distributions using seeded RNGs, ensuring deterministic transitions for a given seed and action sequence.

3.4 Software Architecture and Implementation

FabSim is implemented in *Python*. For its core event-scheduling logic, it leverages the minimalist, highly-optimized event queue from *SimPy* (Matloff 2008). However, it deliberately avoids SimPy's higher-level, slower abstractions for modeling entities. Instead, all system state (e.g., machine status, job locations) is maintained in simple, efficient data structures like NumPy arrays and dictionaries. This micro-DES architecture provides the speed of a custom kernel while retaining the reliability of SimPy's battle-tested event-sequencing logic. The environment uses *Gymnasium* (Towers et al. 2024) for the RL API standard, and *NumPy* for numerical efficiency. It provides the standard Gymnasium API (`__init__`, `reset`, `step`, `observation_space`, `action_space`), ensuring compatibility with common RL libraries (e.g., Stable-Baselines3, RLlib, Tianshou). System configurations (factory layout, machines, buffers, process routes, job details, simulation parameters, stochastic models, reward functions) are defined externally, typically via *YAML* or *JSON* files, promoting flexibility. Reproducibility is ensured through careful *RNG seeding* managed via the `reset` method, the separation of instance generation from simulation runs, and clear logging capabilities.

3.5 Modeling Specific Scheduling Features

The architecture allows modeling key machine scheduling features:

Flexible Routing (including Re-entrance): Job process plans are defined as ordered lists of operations, each specifying the required machine group. The environment routes the job to the next required machine group upon operation completion, naturally handling revisits if a machine group appears multiple times in the sequence.

Batch Processing: Machines designated as batch type consult associated parameters (`min_batch_size`, `max_batch_size`, `max_wait_time`, compatibility rules) stored in the configuration. The simulation logic checks these conditions when a batch machine is idle or a lot arrives, potentially triggering an agent decision (start batch/wait) if configured, or automatically starting batches when minimum size or maximum wait time is reached. Batch completion schedules a single event for all jobs in the batch.

Stochasticity: Machine breakdowns are scheduled based on sampling from time-to-failure distributions (e.g., Exponential based on Mean Time Between Failures (MTBF)) and time-to-repair distributions (e.g., Lognormal based on Mean Time to Repair (MTTR)). Processing times are sampled from specified distributions (e.g., Normal, Uniform) associated with each operation step when the operation begins. All stochastic sampling uses internal, seeded RNGs for reproducibility.

4 EXPERIMENTAL VALIDATION

The validation assessment for FabSim aims to demonstrate its suitability as a testbed for RL-based machine scheduling optimization, focusing on two key aspects: (i) verifying that the environment generates logically consistent schedules that provide a meaningful learning signal for automated agents, and (ii) confirming that its simulation speed meets the demanding requirements for near real-time decision-making often expected in digital twin applications.

4.1 Experimental Setup

All validation experiments utilized a default synthetic job shop configuration: 10 jobs, each comprising 100 operations requiring processing on a set of 20 machines. These machines cover 10 distinct operation types, ensuring specialization and resource contention. Operation processing times were drawn uniformly from discrete intervals of 1 to 4 time steps. This configuration was designed to represent a complex, medium-scale manufacturing environment. The high number of operations per job (100) ensures that agents must learn long-horizon policies, while the significant resource contention (10 concurrent jobs on 20 machines with 10 distinct types) creates a challenging scheduling problem where simple dispatching rules are likely to fail. The dynamic job arrivals further ensure the environment is non-static, testing the agent's ability to adapt to changing conditions, which is characteristic of real-world DJSS problems. To introduce dynamism, job arrivals were uniformly distributed between time step 0 and 100, creating a mix of jobs available at the start and jobs released later during the simulation. To assess the environment's response under RL training, four configurations of the PPO algorithm from Stable-Baselines3 (Raffin et al. 2021) were trained for ten million interaction steps each. These configurations varied primarily in the agent's observation look-ahead (number of future operations considered in the state): (1) one-operation look-ahead, (2) five-operation look-ahead, (3) ten-operation look-ahead, and (4) one-operation look-ahead but with the initial one million actions forced to be uniformly random to emphasize exploration. Standard SB3 hyperparameters were used (two-layer 64-unit MLP policy, learning rate 3×10^{-4}, GAE $\lambda = 0.95$, batch size 64) to focus the assessment on the simulator's behavior rather than extensive algorithm tuning. Five independent trials were run for each configuration, resulting in twenty simultaneous learners interacting with separate instances of the FabSim environment within the same process pool.

4.2 Simulation Validation Assessment and Results

Correctness and Logical Consistency: Schedule validity was checked automatically by the environment during every step of the 200-million-step training corpus. Checks included ensuring assignments respected machine-operation compatibility (only assigning jobs to machines capable of the required operation type),

job release dates (not scheduling jobs before they arrive), and machine availability (not scheduling on a busy or downed machine). Across all experiments, no invalid assignments were permitted by the simulator's internal validation layer, confirming the robustness of the lookup-table indexing and action-masking logic in preventing infeasible agent actions by design. Furthermore, key performance metrics (makespan, cumulative idle time, cumulative wait time) were recalculated from the raw event logs generated by the simulator and found to match the environment-reported metrics to machine precision, validating the integrity of the performance tracking. Qualitative inspection of generated Gantt charts (Figure 1 and Figure 2) revealed logically consistent schedules respecting precedence constraints and machine availability.

Learning Signal Validation: While the primary goal was not policy optimization, the results demonstrated that the environment provides a learnable gradient. The four PPO treatments converged to statistically similar makespans, as shown in Table 1. The one-step look-ahead agent achieved an average makespan of 849.2 (std dev 23.15), while the five-step, ten-step, and exploration-heavy one-step variants stabilized around 787.0 (± 67.68), 878.8 (± 68.07), and 845.0 (± 41.03), respectively. A one-way ANOVA indicated no statistically significant difference at the 0.5% level, aligning with the validation focus. However, the significant variance between runs and the distinct (though not statistically significant) average makespans achieved by agents with different look-aheads illustrate that the environment's state and reward signals effectively differentiate policy qualities, providing the necessary feedback for RL optimization. Further inspection of Gantt charts showed qualitative differences, with longer look-aheads leading to policies that deferred early jobs more often to potentially optimize downstream machine alignment, demonstrating that the environment's state representation is rich enough to support strategic variations learned by the RL agent. Post-mortem analysis confirmed that machines handling the scarcest operation types became bottlenecks, aligning with expected JSSP dynamics and reinforcing FabSim's face validity.

Table 1: Average makespan per treatment from validation runs.

Treatment	Description	Average	Standard Deviation
1	One Look-ahead	849.2	23.15
2	Five Look-ahead	787.0	67.68
3	Ten Look-ahead	878.8	68.07
4	One Look-ahead Uniform Start	845.0	41.03

Speed and Scalability: Simulation speed was profiled on a Windows 11 workstation (AMD Threadripper Pro 5975WX 32-cores, 256 GB RAM, NVIDIA RTX 6000). Average step throughput during the 20-agent training run varied slightly across treatments but consistently remained high: 97.2 ± 3.49 steps/s (one-step), 108.8 ± 1.47 steps/s (five-step), 105.0 ± 3.03 steps/s (ten-step), and 94.8 ± 1.32 steps/s (one-step uniform start). In inference mode (no gradient calculations), mean throughput increased to 102.45 steps/s. Given that a full schedule in this benchmark takes roughly 1000 environment steps, these results indicate that complete episode rollouts finish in well under ten seconds.

To assess scaling, the one-operation configuration was run with varying numbers of simultaneous PPO agents (actors) in the same process pool. As shown in Figure 3 and Table 2, a single actor achieved 3266.2 steps/s. Performance scaled sub-linearly with additional actors: 2474.4 steps/s for two, 2129.2 steps/s for three, and 2126.0 steps/s for four. This degradation beyond two actors likely reflects contention related to Python's Global Interpreter Lock (GIL) within a single process pool on this specific multi-core setup, rather than an inherent limitation of the simulator logic itself. Vectorized Gymnasium wrappers or distributed architectures (e.g., Ray RLlib) can mitigate this on appropriate hardware. Crucially, even the slowest observed rate (approx. 2100 steps/s with four actors) implies that a 1000-step schedule can be simulated in under half a second per actor, comfortably meeting the near real-time requirements (< 1-2 seconds) of many industrial digital twin applications.

The validation study confirms FabSim's correctness in enforcing scheduling constraints and its high computational speed. It provides reliable performance metrics and exposes a sufficiently rich state repre-

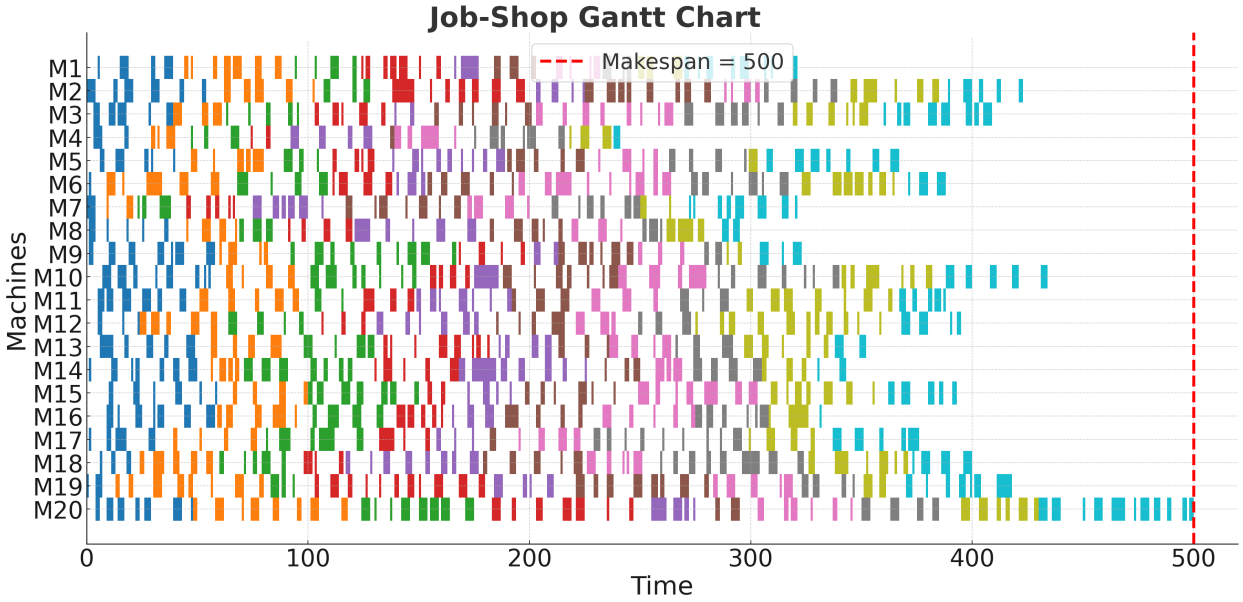

Figure 1: Example Gantt chart generated by FabSim visualizing a job-shop schedule. The y-axis represents machines (M1-M20), and the x-axis represents time. Each colored bar indicates the processing of an operation from a specific job on a particular machine. The dashed red line marks the final makespan for this schedule at 500 time units.

Table 2: Steps per second for simultaneous training models.

Number of Models	Average Steps/Sec	Standard Deviation
1	3266.2	19.50
2	2474.4	32.09
3	2129.2	203.37
4	2126.0	307.80

sentation for RL agents to learn diverse strategies. The environment sustains throughput well within near real-time requirements, even under multi-agent training loads, making it suitable for prototyping intelligent dispatchers and for integration into digital twin or smart manufacturing control loops.

5 DISCUSSION AND CONCLUSION

This paper introduced FabSim, a micro-discrete-event simulation environment purpose-built to address key gaps in DJSS research tools, particularly for applications involving machine learning and digital twins. Existing simulators often lack the combination of speed, flexibility, reproducibility, and integration capabilities needed for modern AI-driven scheduling development. FabSim tackles these deficits through an efficient event-driven kernel, optimized data structures, and adherence to established standards. Our validation experiments confirmed FabSim's core value proposition: it reliably enforces fundamental scheduling constraints (machine compatibility, job releases, resource availability) while generating accurate performance metrics, as demonstrated across 200 million interaction steps with PPO agents without logical errors.

A key limitation of the current study is the absence of direct performance benchmarks against other established environments like FabricatioRL, OR-Gym, or RL4CO. While our results demonstrate high absolute speed, a comparative analysis is essential to rigorously quantify the performance benefits of our micro-DES architecture. Such a benchmark is complex, as environments often differ in their feature sets, abstraction levels, and intended use cases. For instance, platforms like RL4CO are optimized for routing

Figure 2: A second Gantt chart example from FabSim, illustrating a different scheduling outcome, potentially from an alternative policy or simulation run on the same or a similar job-shop instance. The allocation pattern and the resulting makespan (marked at 916 time units by the dashed red line) differ significantly compared to Figure 1, demonstrating the impact of scheduling decisions on overall completion time.

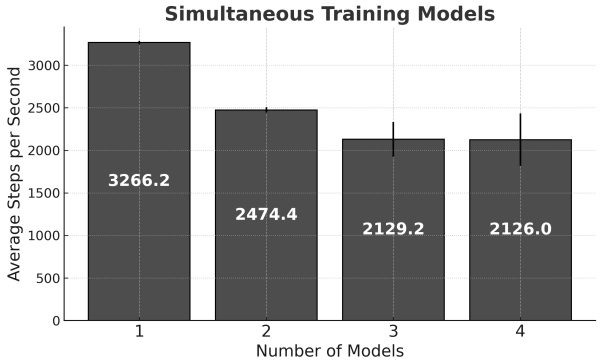

Figure 3: Average steps per second processed by FabSim when training multiple PPO models simultaneously within the same process pool.

problems and may not model detailed shop-floor dynamics like batching or setups. Nevertheless, conducting a fair and detailed comparison on a set of common benchmark problems (e.g., from the FJSSP library) is a critical next step. This future work will provide the community with a clearer understanding of the trade-offs between different platforms and further validate the need for specialized, high-speed simulators for DJSS research.

FabSim delivers the high performance necessary for both large-scale RL training and near real-time deployment. Training throughput rates exceeding 90 steps per second per agent, even under multi-agent loads, and single-actor inference rates surpassing 3000 steps/second were observed on standard hardware. This confirms that complete schedule rollouts for typical benchmark problems can be achieved in well under ten seconds, meeting the latency requirements for integration into digital-twin pipelines where continuous rescheduling and dispatch feedback loops are essential. The environment's compliance with

the Farama Gymnasium standard further enhances its utility, allowing researchers and practitioners to seamlessly integrate FabSim with mainstream RL libraries (like Stable-Baselines3, RLlib) and benchmark diverse approaches—heuristics, metaheuristics, supervised learners, and RL agents—through a unified, reproducible API. The design of its observation and action spaces provides rich contextual information suitable for deep learning models while remaining compact and efficiently manageable.

FabSim provides a robust, validated, and high-speed micro-DES environment specifically tailored for DJSS. Its combination of correctness, performance, standardization, and reproducibility positions it as an ideal platform for advancing the state-of-the-art in machine learning-based scheduling. It serves both as an effective training ground for developing novel AI schedulers and as a potential decision engine for integration into digital twin systems. We acknowledge that a complete digital twin requires robust data pipelines for real-time state synchronization, which is a separate engineering challenge. FabSim's contribution is to provide the high-speed simulation core necessary for the 'what-if' analysis and policy optimization loop within such a system. Ongoing and future work focuses on extending the platform's capabilities. Ongoing and future work focuses on extending the platform's capabilities further by incorporating transport resource modeling, sequence-dependent changeover penalties, and potentially GPU-accelerated kernels, solidifying FabSim as a foundational tool for the next generation of automated scheduling research and deployment in complex, dynamic environments.

REFERENCES

Allahverdi, A., C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov. 2008. "A Survey of Scheduling Problems with Setup Times or Costs". *European Journal of Operational Research* 187(3):985–1032.

Aytug, H., M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy. 2005. "Executing Production Schedules in the Face of Uncertainties: A Review and Some Future Directions". *European Journal of Operational Research* 161(1):86–110.

Berto, F., C. Hua, J. Park, L. Luttmann, Y. Ma, and F. Bu. 2023. "RL4CO: An Extensive Reinforcement Learning for Combinatorial Optimization Benchmark". *arXiv preprint arXiv:2306.17100*.

Brandimarte, P. 1993. "Routing and Scheduling in a Flexible Job Shop by Tabu Search". *Annals of Operations Research* 41(3):157–183.

Garey, M. R., D. S. Johnson, and R. Sethi. 1976. "The Complexity of Flowshop and Jobshop Scheduling". *Mathematics of Operations Research* 1(2):117–129.

Holthaus, O., and C. Rajendran. 1997. "Efficient Dispatching Rules for Scheduling in a Job Shop". *International Journal of Production Economics* 48(1):87–105.

Hubbs, C. D., H. D. Perez, O. Sarwar, N. V. Sahinidis, I. E. Grossmann, and J. M. Wassick. 2020. "OR-Gym: A RL Library for Operations Research Problems". *arXiv preprint arXiv:2008.06319*.

Kampa, A., G. Gołda, and I. Paprocka. 2017. "Discrete Event Simulation Method as a Tool for Improvement of Manufacturing Systems". *Computers* 6(1):10.

Kayhan, B. M., and G. Yildiz. 2023. "Reinforcement Learning Applications to Machine Scheduling Problems: A Comprehensive Literature Review". *Journal of Intelligent Manufacturing* 34(3):905–929.

Kim, D.-W., K.-H. Kim, W. Jang, and F. F. Chen. 2002. "Unrelated Parallel Machine Scheduling with Setup Times Using Simulated Annealing". *Robotics and Computer-Integrated Manufacturing* 18(3-4):223–231.

Kovács, B., P. Tassel, R. Ali, M. El-Kholany, M. Gebser, and G. Seidel. 2022. "A Customizable Simulator for AI Research to Schedule Semiconductor Fabs". In *Proceedings of the 33rd Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 1–6. May 2-5, Saratoga Springs, New York.

Kuhnle, A., M. C. May, L. Schäfer, and G. Lanza. 2022. "Explainable Reinforcement Learning in Production Control of Job Shop Manufacturing System". *International Journal of Production Research* 60(19):5812–5834.

Laguna, M., J. W. Barnes, and F. W. Glover. 1991. "Tabu Search Methods for a Single Machine Scheduling Problem". *Journal of Intelligent Manufacturing* 2:63–73.

Lee, K.-M., T. Yamakawa, and K.-M. Lee. 1998. "A Genetic Algorithm for General Machine Scheduling Problems". In *Proceedings of the Second International Conference on Knowledge-Based Intelligent Electronic Systems (KES'98)*, Volume 2, 60–66. April 21-23, Adelaide, Australia.

Lenstra, J. K., A. H. G. Rinnooy Kan, and P. Brucker. 1977. "Complexity of Machine Scheduling Problems". In *Annals of Discrete Mathematics*, Volume 1, 343–362. Elsevier.

Liu, C.-L., C.-C. Chang, and C.-J. Tseng. 2020. "Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems". *IEEE Access* 8:71752–71762.

Luo, S., L. Zhang, and Y. Fan. 2021. "Real-Time Scheduling for Dynamic Partial-No-Wait Multiobjective Flexible Job Shop by Deep Reinforcement Learning". *IEEE Transactions on Automation Science and Engineering* 19(4):3020–3038.

Maravelias, C. T. 2006. "A Decomposition Framework for the Scheduling of Single- and Multi-Stage Processes". *Computers & Chemical Engineering* 30(3):407–420.

Matloff, N. 2008. "Introduction to Discrete-Event Simulation and the SimPy Language". Technical report, Department of Computer Science, University of California at Davis.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, and M. G. Bellemare. 2015. "Human-Level Control Through Deep Reinforcement Learning". *Nature* 518(7540):529–533.

Mönch, L., J. W. Fowler, S. Dauzère-Pérès, S. J. Mason, and O. Rose. 2011. "A Survey of Problems, Solution Techniques, and Future Challenges in Scheduling Semiconductor Manufacturing Operations". *Journal of Scheduling* 14:583–599.

Nsiye, E., T. Wright, B. Tse, and S. Mondesire. 2024. "A Micro-Discrete Event Simulation Environment for Production Scheduling in Manufacturing Digital Twins". In *MODSIM World*. May 20-22, Norfolk, Virginia.

Ouelhadj, D., and S. Petrovic. 2009. "A Survey of Dynamic Scheduling in Manufacturing Systems". *Journal of Scheduling* 12:417–431.

Pinedo, M. L. 2012. *Scheduling*. New York: Springer.

Raffin, A., A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. 2021. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". *Journal of Machine Learning Research* 22(268):1–8.

Rinciog, A., and A. Meyer. 2021. "Fabricatio-RL: A Reinforcement Learning Simulation Framework for Production Scheduling". In *2021 Winter Simulation Conference (WSC)*, 1–12 https://doi.org/10.1109/WSC52266.2021.9715372.

Rinnooy Kan, A. H. G. 2012. *Machine Scheduling Problems: Classification, Complexity and Computations*. New York: Springer Science & Business Media.

Soykan, B., and G. Rabadi. 2023. "Optimizing Multi Commodity Flow Problem Under Uncertainty: A Deep Reinforcement Learning Approach". In *Proceedings of the 2023 International Conference on Machine Learning and Applications*, 1267–1272. December 15-17, Jacksonville, Florida.

Soykan, B., and G. Rabadi. 2024. "Optimizing Job Shop Scheduling Problem Through Deep Reinforcement Learning and Discrete Event Simulation". In *2024 Winter Simulation Conference (WSC)*, 2607–2618.

Sutton, R. S., and A. G. Barto. 2018. *Reinforcement Learning: An Introduction*. 2nd ed. The MIT Press.

Towers, M., A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, and T. Deleu. 2024. "Gymnasium: A Standard Interface for Reinforcement Learning Environments". *arXiv preprint arXiv:2407.17032*.

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

Zhang, C., W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi. 2020. "Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning". In *Advances in Neural Information Processing Systems*, Volume 33, 1621–1632.

AUTHOR BIOGRAPHIES

SEAN MONDESIRE is an Assistant Professor in the School of Modeling, Simulation, and Training (SMST) at UCF. His research interests include real-time decision-making through machine learning, big data analytics, scalable digital twins, and simulations. His email address is sean.mondesire@ucf.edu.

MICHAEL BROWN is a doctoral student in the Modeling and Simulation program at SMST. He holds a Bachelor's degree in Computer Science and is a member of the High-performance AI Laboratory. His email address is michael.brown2@ucf.edu.

BULENT SOYKAN is an Associate Research Scientist in the Institute for Simulation and Training at UCF. His research interests include real-time decision-making under uncertainty, reinforcement learning. His email address is bulent.soykan@ucf.edu.