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.

DES-GYMNAX: FAST DISCRETE-EVENT SIMULATOR IN JAX

Yun Hua[1], Jun Luo[1], and Xiangfeng Wang[2]

[1]Antai College of Economics and Management, Shanghai Jiao Tong University, Shanghai, CHINA
[2]School of Computer Science and Technology, East China Normal University, Shanghai, CHINA

ABSTRACT

In this work, we introduce **DES-Gymnax**, a novel high-performance discrete-event simulator implemented in JAX. By leveraging the just-in-time compilation, automatic vectorization, and GPU acceleration capabilities of JAX, DES-Gymnax can achieve 10x to 100x times performance improvement over traditional PYTHON-based discrete-event simulators like *Salabim*. The proposed DES-Gymnax can feature a Gym-like API that facilitates seamless integration with reinforcement learning algorithms, addressing a critical gap between simulation engines and AI techniques. DES-Gymnax is validated on three benchmark models, i.e., an M/M/1 queue, a multi-server model, and a tandem queue model. Experimental results demonstrate that DES-Gymnax maintains simulation accuracy while significantly reducing execution time, enabling efficient large-scale sampling crucial for reinforcement learning applications in operations research areas. The open-source code is available in the DES-Gymnax repository (Yun, Jun, and Xiangfeng 2025).

1 INTRODUCTION

Discrete-event simulation (DES) serves as a critical modeling and analysis tool widely applied in operations research tasks closely related to national economies, such as supply chain management (Manuj, Mentzer, and Bowers 2009), warehouse planning (Agalianos, Ponis, Aretoulaki, Plakas, and Efthymiou 2020), and factory scheduling (Wu and Wysk 1989; Greasley 2005). discrete-event simulators constitute the core tools enabling efficient execution of these simulation tasks. Despite the availability of powerful commercial engines (e.g., Anylogic (Borshchev 2014), Arena (Allen 2011) and open-source alternatives (e.g., Unity-SimuLean (Pernas-Álvarez and Crespo-Pereira 2024)), the integrating of artificial intelligence (AI) techniques, particularly reinforcement learning (RL) (Sutton and Barto 1998) and large language models (LLMs) (Sanderson 2023) into DES processes remains a pressing challenge. meet two key requirements: (1) providing standardized interfaces that enable seamless interaction between AI algorithms and the environment to facilitate training, and (2) delivering high simulation performance to support extensive training–especially for AI algorithms like reinforcement learning, which demand substantial sampling and efficient simulation capabilities (Yu 2018; Zhang et al. 2021).

The widespread adoption of Python in AI research, coupled with the standardization of the OpenAI Gym interface (Brockman, Cheung, Pettersson, Schneider, Schulman, Tang, and Zaremba 2016) for reinforcement learning (RL) environments, highlights the strategic value of Python-based discrete-event simulators (DES) for AI integration. Frameworks such as SimPy (Zinoviev 2024) and Salabim (van der Ham 2018), along with initiatives like QGym (Chen et al. 2024), represent significant steps toward bridging DES with AI methodologies. Nevertheless, these platforms often face critical limitations—including long simulation run times and inadequate support for parallel execution—which hinder their effectiveness in RL applications that demand large-scale data generation.

While traditional DES research has explored performance improvements through parallel computing (Fujimoto 1990), distributed systems (Legrand 2001), and GPU acceleration (Chapuis et al. 2015), these techniques are frequently incompatible with Python-based AI workflows. In the AI domain, high simulation efficiency is essential for solving complex tasks. Frameworks like OpenSpiel (Lanctot et al.

2019) have employed hybrid approaches combining C++ simulation cores with Python APIs to improve the efficiency. However, such designs encounter challenges including cross-language performance bottlenecks and limited GPU compatibility. Recent advancements in Python-native high-performance simulation environment for RL training, such as Isaac Gym (Makoviychuk et al. 2021), Brax (Freeman et al. 2021), Pgx (Koyamada et al. 2023), and Gymnax (Lange 2022), leverage the JAX library (Bradbury et al. 2018) to deliver efficient, scalable simulation environments. By utilizing JAX's automatic vectorization, Just-In-Time (JIT) compilation and multi-device support, implementing parallelized sampling of simulation tasks and significantly enhancing sampling efficiency during RL training.

Simulator	Parallel	Support GPU	Gym-like API	Python-Original
Simpy (Zinoviev 2024)	×	×	×	✓
Salabim (van der Ham 2018)	×	×	×	✓
QGym (Chen, Li, Che, Dong, Peng, and Namkoong 2024)	×	×	✓	✓
SimX (Thulasidasan, Kroc, and Eidenbenz 2014)	✓	×	×	×
DES-Gymnax	✓	✓	✓	✓

Table 1: Comparison of various Python-based discrete-event simulators.

This paper introduces **DES-Gymnax**, a novel discrete event simulator implemented using the JAX library. DES-Gymnax is designed to combine scalable simulation capabilities with useful compatibility for AI applications. Key characteristics of DES-Gymnax include: 1) Efficient Simulation Performance: By leveraging JAX's core features—automatic vectorization, Just-In-Time (JIT) compilation, GPU/TPU acceleration, and functional programming support—DES-Gymnax achieves substantial computational efficiency. This enables support for large-scale, compute-intensive tasks such as reinforcement learning (RL) training; 2) AI-Friendly Design: DES-Gymnax incorporates a standardized API adhering to the Gymnasium (formerly Gym) interface conventions. This design choice promotes straightforward integration with contemporary AI algorithms and frameworks, enabling rapid prototyping and deployment of RL agents interacting with simulated environments. To validate the effectiveness of DES-Gymnax, we implemented three representative queuing models: the M/M/1 queue, a multi-server system, and a tandem queue. These benchmarks were used to assess both the computational efficiency and the fidelity of the simulation. Results demonstrate that DES-Gymnax provides significant speed advantages over traditional Python-based simulators like Salabim, while maintaining accuracy in event-driven dynamics. In addition, Table 1 presents a qualitative comparison between DES-Gymnax and other Python-based discrete event simulation frameworks. This analysis shows that DES-Gymnax uniquely combines four critical properties—parallelism, hardware acceleration, AI-friendly design, and pure Python implementation—making it particularly suitable for simulation-driven AI research.

The primary contributions of this paper are as follows: **(1) Development of DES-Gymnax:** A high-performance, parallelized, and AI-friendly discrete-event simulator built on JAX, tailored for operations research and AI applications; **(2) Significant Performance Improvements:** Experimental results demonstrate that JAX-DES outperforms Python-based discrete-event simulators like Salabim, achieving 10 to 100 times speedups in large-scale simulation tasks; **(3) Facilitation of AI and Operations Research Synergy:** By furnishing a standardized Gym-compatible interface, DES-Gymnax lowers the barrier for integrating DES methodologies with advanced AI techniques, thereby fostering innovation, especially in RL-based control and optimization of simulated systems.

2 RELATED WORK

2.1 Discrete-event Simulator

The efficacy of DES implementations critically depends on the underlying efficiency and scalability characteristics. Traditionally, discrete-event simulators have been constrained by their inherently sequential

execution model, where events must be processed in strict temporal order. This sequential nature poses significant computational limitations when simulating large-scale, complex systems, often resulting in prohibitively lengthy execution times for practical applications.

To overcome these performance constraints, researchers have developed Parallel discrete-event Simulation (Fujimoto 1990) (PDES), which involves decomposing monolithic DES models into multiple concurrent sub-models that can execute independently. Critical to this methodology is the implementation of sophisticated time synchronization algorithms that maintain temporal causality and ensure simulation correctness across distributed execution contexts. Contemporary PDES frameworks leverage diverse computational architectures—including multi-core CPUs, distributed systems (Legrand 2001), and GPU (Chapuis, Eidenbenz, Santhi, and Park 2015) to achieve substantial performance improvements. Despite these computational advantages, many state-of-the-art PDES solutions present significant usability barriers, particularly regarding their limited integration with Python—the predominant programming language in AI research. This interoperability gap significantly impedes the seamless incorporation of advanced decision-making algorithms, such as RL algorithm, within simulation environments.

Recent years have witnessed the emergence of Python-based DES simulators featuring graphical interfaces and compatibility with AI frameworks like OpenAI Gym (Brockman, Cheung, Pettersson, Schneider, Schulman, Tang, and Zaremba 2016). These developments have facilitated the application of RL methods to simulation-based problems. However, most Python-compatible simulators remain bound by sequential processing limitations, with only a small subset supporting parallel execution paradigms. Existing solutions such as SimX (Thulasidasan, Kroc, and Eidenbenz 2014) and Simulus (Liu 2020) offer limited parallelism but suffer from poor compatibility with modern AI libraries due to maintenance deficiencies. This technological gap highlights the urgent need for a robust, Python-compatible PDES framework that integrates seamlessly with modern AI methodologies to support sophisticated decision-making in large-scale, computationally intensive simulations.

2.2 JAX in Simulation

JAX (Bradbury et al. 2018) has emerged as a transformative Python library that combines accelerator-oriented array computation with powerful program transformation capabilities, specifically designed for high-performance numerical computing and large-scale machine learning applications. Its impact extends across numerous AI domains, including optimization acceleration (Blondel et al. 2022) and efficient sampling in RL algorithms (Lange 2022; Koyamada et al. 2023). Given that RL, particularly online methods—fundamentally relies on simulation environments for training, JAX has been increasingly integrated into RL simulation pipelines to enhance computational efficiency.

Current JAX implementations span diverse simulation domains, including board games (Koyamada et al. 2023), physics engines, and environments for research in robotics, human perception, materials science, and other simulation-intensive applications (Freeman et al. 2021). Gymnax (Lange 2022) represents a prominent example of this integration, providing JAX-accelerated implementations of various RL environments encompassing classic control problems, bsuite benchmarks, MinAtar environments, and numerous classic and meta-RL tasks. Its gym-compatible API facilitates seamless adoption by researchers familiar with standard RL frameworks. Additionally, Gymnax provides companion baseline implementations through Gymnax-BLines, enabling rigorous algorithm evaluation within JAX-accelerated environments. Beyond this, frameworks such as Foragax (Chaturvedi et al. 2024) further extend JAX's applicability to agent-based modeling, highlighting its versatility across diverse simulation paradigms.

The proposed DES-Gymnax framework can extend this paradigm by inheriting Gymnax's intuitive API while introducing JAX acceleration specifically tailored for discrete-event simulation. This approach not only significantly enhances simulation performance but also provides a framework that aligns with the methodological requirements and expectations of the AI research community, thereby bridging the longstanding gap between high-performance simulation and advanced AI techniques.

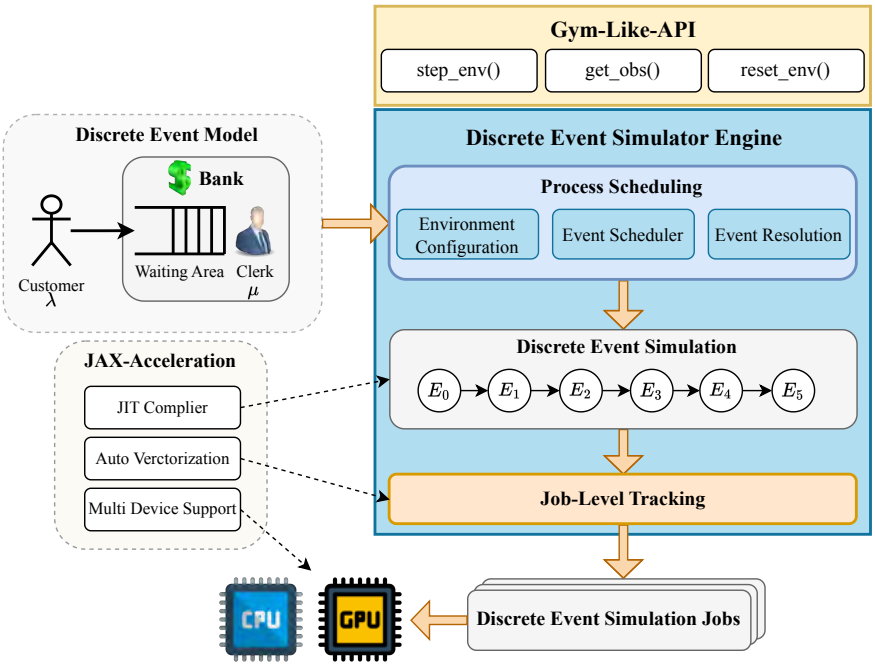


Figure 1: The DES-Gymnax framework provides a powerful simulator for discrete-event simulation. Its discrete-event Simulator Engine utilizes a Process Scheduling module to create corresponding simulation processes. Leveraging JAX's auto-vectorization capabilities, the framework enables parallel processing of jobs, while JAX's multi-device support facilitates parallelization across hardware. A gym-like API provides a standardized interface for seamless integration with RL and other AI algorithms.

3 THE DES-GYMNAX FRAMEWORK

This section elaborates on the DES-Gymnax framework, detailing its core design principles, its utilization of JAX for performance enhancement, and the architecture of its discrete-event simulator engine. The overall architecture of DES-Gymnax is depicted in Figure 1.

3.1 Design Principles

The development of DES-Gymnax has been guided by four fundamental design principles that collectively optimize its performance for AI applications: **(1) Python-Native Implementation**: DES-Gymnax is constructed entirely within the Python ecosystem, facilitating seamless integration with existing AI workflows while maintaining flexibility for customization; **(2) Parallel Execution Architecture**: The framework incorporates comprehensive support for parallel execution across computational resources, significantly enhancing simulation efficiency when training AI policies; **(3) GPU Acceleration**: DES-Gymnax leverages hardware acceleration through GPU computing to efficiently process batched simulations, enabling the generation of thousands of parallel training samples; **(4) AI-Friendly Interface**: DES-Gymnax exposes a Gym-Like API. This provides a familiar and convenient interface for AI researchers and practitioners, thereby streamlining the integration of sophisticated decision-making algorithms (like RL agents) within the simulation environment.

3.2 JAX Acceleration

JAX offers several key features for high-performance computing, including just-in-time (JIT) compilation, automatic vectorization, multi-device support, and automatic differentiation. DES-Gymnax leverages the

first three to enable scalable, parallel discrete-event simulations. As our setting does not require gradient-based optimization, automatic differentiation is not utilized. The following paragraphs describe how each of these features contributes to the performance and scalability of DES-Gymnax.

3.2.1 Just-In-Time Compilation

JAX's just-in-time (JIT) compilation via `jax.jit` transforms Python functions into XLA (Accelerated Linear Algebra) optimized routines. This compilation process is particularly beneficial for simulation, where the same computational patterns are repeatedly executed with different inputs. For DES-Gymnax, JIT compilation provides several advantages: The JIT compilation eliminates Python's interpretation overhead, resulting in performance comparable to compiled languages like C++. It converts high-level Python code into optimized machine instructions that execute directly on the target hardware without the performance penalties associated with interpreted languages. The compilation process enables optimizations such as function inlining, loop unrolling, and constant folding, which are particularly effective for the repetitive calculations common in simulation. By analyzing the computational graph of the simulation functions, JAX can identify and eliminate redundant operations, merge multiple operations into more efficient compound operations, and pre-compute values that don't change across iterations. This approach ensures that the performance-critical path of simulation execution benefits from the full suite of JAX's optimization capabilities.

3.2.2 Auto Vectorization

JAX's vectorization capabilities through `jax.vmap` (vectorized mapping) allow DES-Gymnax to efficiently process batches of simulations in parallel. This vectorization transforms functions that operate on single simulations into functions that process batches of simulations simultaneously, utilizing the parallel computing capabilities of modern hardware. For RL applications, this means that thousands of environment rollouts can be performed in parallel, dramatically reducing the time required for data collection during training. The `vmap` transformation is particularly powerful because it preserves the simplicity of writing code for a single simulation while enabling efficient batch processing. Developers can focus on the logic of their simulation model without the complexity of manual parallelization, yet still benefit from the performance advantages of vectorized execution. Vectorization is especially effective on GPUs, which are designed for performing the same operations across large arrays of data. By structuring the simulation to take advantage of this capability, DES-Gymnax achieves significantly better hardware utilization compared to sequential approaches.

3.2.3 Multi Device Support

JAX's transparent support for GPU acceleration allows DES-Gymnax to execute simulations on specialized hardware without requiring environment-specific code modifications. This hardware flexibility is achieved through JAX's array handling, which automatically manages device memory transfers and execution. For simulation-intensive applications, GPU acceleration can provide 10-100 times performance improvements over CPU execution.

3.3 Discrete-event Simulator Engine

The core of DES-Gymnax is its event-driven simulation engine, responsible for managing the scheduling and processing of events according to their timestamps. A detailed examination from the perspective of DES reveals how our implementation builds upon established DES principles while strategically adapting them for compatibility with and acceleration by JAX. Traditional DES frameworks typically require users to define simulation entities and their associated properties—for instance, in an M/M/1 model, parameters such as server service rates and customer arrival rates must be specified. Timestamps are assigned to events to determine their occurrence times and establish precedence relationships. Most DES implementations

adopt an event-scheduling approach, advancing simulation time discretely based on the next scheduled event, thereby updating the system state only when necessary. Conventional simulators often implement this logic using structures like priority queues and procedural functions (e.g., a `process()` method) that execute events sequentially based on time, often involving mutable state updates.

DES-Gymnax adheres to this fundamental event-driven architecture but introduces crucial innovations for JAX compatibility and performance. Specifically, it employs a functional approach to **Process Scheduling** for handling event logic within a single simulation instance and utilizes **Job-Level Tracking** combined with JAX's vectorization capabilities to achieve massive parallelization across multiple independent simulation instances. These components are detailed below.

3.3.1 Process Scheduling

Traditional discrete-event simulators typically implement process scheduling through mutable data structures such as priority queues, with object-oriented designs that directly modify system state during event execution. These conventional approaches rely on imperative programming patterns with explicit conditional branching to manage event sequencing and state transitions, often using pointer-based structures that are incompatible with accelerator hardware. DES-Gymnax reimagines process scheduling within JAX's functional paradigm through three integrated components.

Environment Configuration uses immutable dataclasses to define simulation parameters like any other discrete-event simulators. Listing 1 gives an example for M/M/1 Model.

Listing 1: JAX array based EnvParams for M/M/1 Model.

```
@struct.dataclass
class EnvParames(environment.EnvParams):
   """Parameters for configuring the simulation environment."""
   max_timestap: int = ...
   clerk_processing_time: float = ...
   customers_arriving_time: float = ...
   initilized_time: float = datetime().timestamp(...)
```

Event Scheduler utilizes immutable JAX arrays containing events with their corresponding timestamps, serving as the system state representation and replacing conventional mutable priority queues employed in most discrete-event simulators. Listing 2 shows an environment state example for the M/M/1 Model. During each iteration, the simulator executes the `step_env` function based on the current system state, allowing the simulation to progress by processing the event with the minimum temporal distance from the current `clock_time`, following established discrete-event simulation principles. Subsequently, the system updates its state and generates a new immutable EnvState object compatible with JAX operations. In contrast to conventional discrete-event simulators that simulate entire processes continuously based on predetermined time intervals, DES-Gymnax offers event-granular simulation, providing system state information at each event occurrence. This approach facilitates the integration of decision-making processes during simulation execution, creating an AI-friendly environment for training RL and other AI algorithms.

Listing 2: JAX array based EnvState for M/M/1 Model.

```
@struct.dataclass
class EnvState(environment.EnvState):
   customers_in_the_queue: float
   last_customer_enter_time: float
   last_clerk_processing_time: float
   clock_time: float
   served_customers: float
```

Event Resolution transforms traditional imperative control flow into functional constructs using JAX's conditional primitives, as shown in Listing 3. It employs `lax.cond` for efficient branching without Python

control flow, preserving JIT compatibility. The algorithm compares the expected arrival time of the next customer with the processing completion time of the current one to determine the next event: an arrival, a completion, or a rare simultaneous occurrence. Each branch delegates to a handler that updates the environment state in a pure, side-effect-free manner. Instead of a mutable priority queue, the next event is selected by comparing event timestamps directly within the data, and resolved using functional branching. The simulation runs as a loop over discrete event steps, implemented using `lax.scan`, which repeatedly calls `process_next_event` while threading the immutable environment state. This enables the entire simulation to be compiled, vectorized, or parallelized using JAX's transformations such as `jit` and `vmap`.

Listing 3: JAX conditional event resolution logic for M/M/1 Model.

```
def resolve_event_case():
   return lax.cond(
      expected_next_arriving_time < expected_next_processing_time,
      lambda _: self.updatWhileCustomerArrive(key, state, params),
      lambda _: lax.cond(
         expected_next_arriving_time > expected_next_processing_time,
         lambda _: self.updateWhileClerkProcess(key, state, params),
         lambda _: self.handleEqualTime(key, state, params),
         operand=None)
```

3.3.2 Job-Level Tracking and Parallelization

Conventional discrete-event simulators typically execute simulations in a sequential manner, necessitating bespoke parallel implementations that frequently entail intricate thread management, message passing interfaces, or distributed computing architectures. For instance, integrating simulators with general-purpose parallel frameworks such as MPI presents significant limitations, as MPI exclusively supports CPU parallelization, rendering GPU parallelization exceedingly challenging to implement. DES-Gymnax introduces a job-level abstraction that fundamentally transforms this paradigm by encapsulating individual simulation instances as discrete computational units, where each *job* represents a single, independent discrete-event simulation duplication, initialized with distinct random seeds and configurations, and executed independently. This implementation leverages the `jax.vmap` function to utilize JAX's automatic vectorization capabilities. Unlike traditional parallel computing libraries such as OpenMPI, which require explicit thread management and typically parallelize at the process or thread level, `vmap` operates at the function level, enabling the transformation of a single-instance simulation function into a batched form that executes multiple duplications in parallel. This batching is deeply fused with JAX's just-in-time (JIT) compilation and hardware acceleration pipelines, allowing efficient parallel executions using single-instruction, multiple-data (SIMD) paradigms.

Listing 4: Parallelization with jobs and JAX's automatic vectorization.

```
   def simulate(params, max_time=10000):
   """Run a single job with the given parameters"""
      state = initialize_simulation(params)

      def step_fn(state, _):
         next_state = process_next_event(state)
         return next_state, next_state

      final_state, trajectory = jax.lax.scan(
      step_fn, state, jnp.arange(max_steps))
      return final_state, trajectory

   def batch_simulate(rng_input, env, env_params):
      batch_simulate_fn = jax.vmap(rollout, in_axes=(0, None, None))
      return batch_simulate_fn(rng_input, env, env_params)
```

3.3.3 Gym-Like API

DES-Gymnax provides a familiar, Gym-Like interface that simplifies integration with existing RL algorithms and frameworks. The core API implements the standard Gym interface methods which are inherited from the Gymnax environment architecture. This API design follows the well-established OpenAI Gym convention, providing a consistent set of methods for RL researchers: `reset_env()`: Initializes or resets the simulation to its starting state, returning both the initial observation and internal state; `get_obs()`: Extracts relevant features from the internal state to form observations for the agent, including queue length, simulation time, customer throughput, and waiting time metrics; `step_env()`: Advances the simulation by applying an action and resolving the next event, then returns the new observation, reward, termination status, and auxiliary information; `action_space` and `observation_space`: Define the format and constraints of valid actions and observations, enabling automatic validation and standardized agent interfaces.

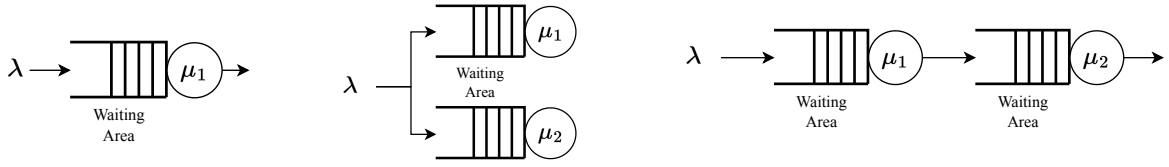
This standardized interface ensures that DES-Gymnax environments can be readily used with popular RL libraries like gymnax-blines, without requiring custom adaptations. The implementation leverages JAX features like `lax.stop_gradient` to prevent unnecessary gradient computation for non-differentiable operations, optimizing training efficiency. Researchers familiar with the Gym interface can quickly apply their existing knowledge to discrete-event simulations without a steep learning curve, while benefiting from the performance advantages of JAX acceleration.

4 EXPERIMENT

We conducted comprehensive experiments to evaluate the performance of DES-Gymnax. This section presents performance results across diverse simulation scenarios under various computational settings (CPU-only and GPU-accelerated). We benchmark DES-Gymnax against the established Python-based discrete-event simulator Salabim, including comparisons with Salabim's MPI implementation to assess parallel processing capabilities.

4.1 Experiment Setup

To rigorously evaluate DES-Gymnax's performance, we implemented a series of controlled experiments using three distinct simulation scenarios: the M/M/1 model, M/M/C model, and tandem queue model. Experiments were conducted on both consumer-grade and high-performance computing platforms. For small-scale experiments, we utilized a MacBook Pro with 16GB RAM and an M1 Pro 8-core CPU. Large-scale performance testing was conducted on a virtual machine provisioned on a GPU server with 24 CPU cores, 128GB RAM, and an NVIDIA A100 GPU with 80GB VRAM. To emphasize, the three benchmark scenarios are illustrated in Figure 2 and described below:



(a) M/M/1 Model. (b) M/M/C Model. (c) Tandem Queue Model.

Figure 2: The three scenarios in the experiment.

- **M/M/1 Model**: A fundamental single-server queuing system characterized by:
  - **Arrival Process**: Customers arrive according to a Poisson process with a mean rate of λ.

  - **Queue Discipline**: A single queue holds waiting customers, served in First-In, First-Out (FIFO) order. Queue capacity is typically assumed to be infinite unless specified otherwise.
  - **Service Process**: A single server processes customers with service times drawn independently from an exponential distribution with a mean service rate of μ.
- **M/M/C Model**: A multi-server queuing system with C identical servers operating in parallel, characterized by:
  - **Arrival Process**: Customers arrive according to a Poisson process with a mean rate of λ.
  - **Queue Discipline**: New customers join the shortest available queue.
  - **Service Process**: C identical servers operate in parallel. Each server processes customers with service times drawn independently from an exponential distribution with a mean service rate of μ (per server).
- **Tandem Queue Model**: A multi-stage service system where customers proceed sequentially through multiple service nodes:
  - **Arrival Process**: Customers arrive at the first stage according to a Poisson process with a mean rate of λ.
  - **System Structure**: Customers flow sequentially through N distinct service servers.
  - **Service Process (per stage)**: Each server i typically operates as an independent queue. Service times at stage i are drawn independently from an exponential distribution with a mean service rate of μ.

Table 2: Simulation parameters for performance testing.

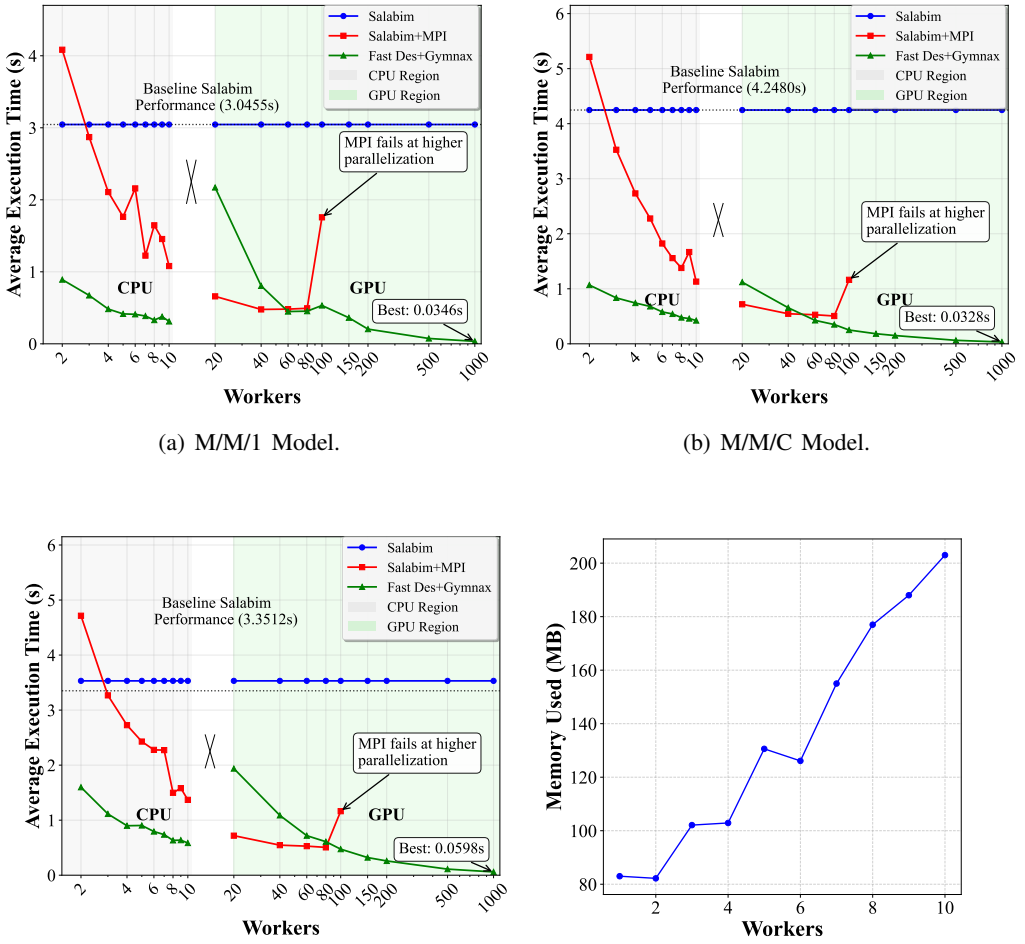
Parameter	Description	M/M/1 Model	M/M/C Model	Tandem Queue Model
λ	Arrival rate (customers/minute)	4.8	4.8	4.8
μ	Service rate (customer/minute)	2	2	2
C	Number of servers	1	2	2

4.2 Performance Testing Results

Our experimental results validate the substantial performance gains achieved by DES-Gymnax over conventional Python-based simulators like Salabim. The specific parameters employed for all simulation scenarios are summarized in Table 2. And we use 'workers' to refer to the number of parallel simulation instances—executed via vectorization (e.g., using vmap) in DES-Gymnax or distributed across processes using MPI in Salabim—used to evaluate simulation throughput. Note that non-steady-state parameters were intentionally used for these benchmarks, leading to unbounded queue growth. This choice focused the evaluation on raw simulator throughput and efficiency, not steady-state system analysis. Across the tested configurations, DES-Gymnax consistently achieved speedups ranging from 6 to 10 times compared to Salabim under baseline execution conditions (e.g., single-core CPU). When leveraging GPU acceleration across 1,000 parallel simulation instances (or 'workers'), this performance gap widened dramatically, with DES-Gymnax demonstrating speedups of approximately 100x relative to Salabim.

Figure 3(a) illustrates performance comparisons for the M/M/1 Model. With just 10 workers, DES-Gymnax achieved a 10 times speedup over Salabim. When scaled to 1000 GPU-accelerated workers, DES-Gymnax delivered nearly 100 times performance improvement, completing simulations in an average of 0.0346 seconds compared to Salabim's 3.0455 seconds. Notably, the MPI implementation exhibited scaling limitations beyond 100 workers. For the M/M/C model, Figure. 3(b) shows similar performance patterns. DES-Gymnax maintained a 10 times speedup with 10 workers and exceeded 100 times improvement with 1000 GPU-accelerated workers (0.0328 seconds versus Salabim's 4.2480 seconds). The MPI implementation again demonstrated scalability constraints beyond 100 workers.

The result on tandem queue model showed consistent acceleration trends as depicted in Figure. 3(c). DES-Gymnax achieved a 6 times speedup with 10 workers and approximately 60 times improvement



(a) M/M/1 Model.

(b) M/M/C Model.

(c) Sequential Queue Model.

(d) Used memory for simulating bank.

Figure 3: Average used times for different simulators on different scenarios and the memory use for bank.

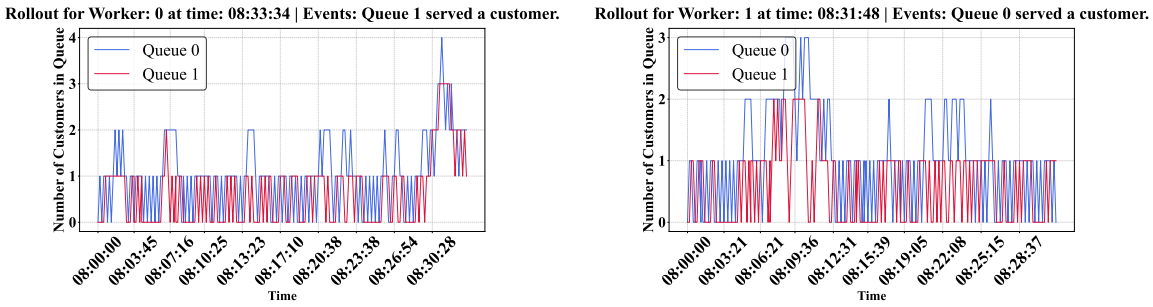
with 1000 GPU-accelerated workers (0.0598 seconds versus Salabim's 3.5312 seconds). Similar to other scenarios, MPI scaling deteriorated beyond 100 workers.

4.3 Correctness and Memory Usage

Beyond raw performance metrics, we validated the simulation accuracy of DES-Gymnax by comparing its results with theoretical values derived from queueing theory, following the stationary analysis outlined in (Allen 1978). Table 3 makes 10 simulations with 10,000 events using different random seed, and the results show that the DES-Gymnax can get the correct simulation for the M/M/1 model. To further verify statistical accuracy, we conducted detailed experiments with the M/M/C model using $\lambda = 3$, $\mu = 3$, and $n = 2$ servers. As shown in Figure 4, we performed both small-scale (2 simulations, 2 workers, 200 events) and large-scale (10 simulations, 10,000 events) validation runs. The large-scale results yielded mean waiting times of $\hat{W} = 22.037 \pm 0.223$ and mean queue lengths of $\hat{L} = 1.099 \pm 0.011$, closely approximating the theoretical values of $W = 26.667$ and $L = 1.333$, thereby confirming the statistical validity of our simulation approach. Besides, Figure 3(d) illustrates memory consumption patterns in the M/M/1 model. We observed non-linear memory growth as worker count increased, due to increased resource contention, system overhead, and complex concurrency interactions that emerge in highly parallel simulation environments.

λ (per/min)	μ (per/min)	W(s)	$\hat{W}$(s)	L(per)	$\hat{L}$(per)
3	6	10.00	10.012 $\pm$ 0.010	0.50	0.502 $\pm$ 0.002
1.5	3	20.00	20.000 $\pm$ 0.000	0.50	0.499 $\pm$ 0.002
1.5	2.4	41.67	49.348 $\pm$ 4.902	1.04	1.231 $\pm$ 0.159
1.5	2	90.45	97.352 $\pm$ 7.82	2.26	2.403 $\pm$ 0.163
1.5	1.5	Null	increasing	Null	increasing

Table 3: Simulation Correctness for the M/M/1 Model. W and L represent the theoretical waiting time and queue length, respectively, while $\hat{W}$ and $\hat{L}$ denote their simulated counterparts. Error bars ($\pm$) indicate standard errors. "Null" signifies that the steady-state values could not be computed due to system instability; theoretically, the queue grows without bound, which is also observed in practice.



(a) Worker 0.

(b) Worker 1.

Figure 4: Simulation Results for the M/M/C Model.

5 CONCLUSION

DES-Gymnax represents a significant advancement in discrete-event simulation, offering high performance, parallelization, and AI-friendly capabilities. The integration of JAX's JIT compilation, automatic vectorization, and multi-device support enables DES-Gymnax to achieve substantial speedups in large-scale simulation tasks. The standardized Gym-like API facilitates seamless integration with RL libraries, fostering deeper integration of DES with AI techniques. Future work will focus on further performance optimization, particularly in parallel processing of Queue Networks, and the addition of benchmarks for RL methods to test sample efficiency in decision problems like job scheduling. We will also explore how to better handle decomposition of complex Discrete Event Systems (DES), which remains an open problem.

REFERENCES

Agalianos, K., S. Ponis, E. Aretoulaki, G. Plakas and O. Efthymiou. 2020. "Discrete Event Simulation and Digital Twins: Review and Challenges for Logistics". *Procedia Manufacturing* 51:1636–1641.

Allen, A. O. 1978. *Probability, Statistics, and Queueing Theory*. New York: Academic Press.

Allen, T. T. 2011. "Introduction to ARENA software". In *Introduction to Discrete Event Simulation and Agent-based Modeling: Voting Systems, Health Care, Military, and Manufacturing*, 145–160. Springer.

Blondel, M., Q. Berthet, M. Cuturi, R. Frostig, S. Hoyer, F. Llinares-López, *et al*. 2022. "Efficient and Modular Implicit Differentiation". In *Proceedings of the 36th International Conference on Neural Information Processing Systems*. November 28th–December 9th, New Orleans, LA, USA, 5230–5242.

Borshchev, A. 2014. "Multi-Method Modelling: AnyLogic". In *Discrete-Event Simulation and System Dynamics for Management Decision Making*, edited by B. D. Sally Brailsford, Leonid Churilov, 248–279. Hoboken, NJ: John Wiley & Sons, Ltd.

Bradbury, J., R. Frostig, P. Hawkins, M. J. Johnson, C. Leary and D. Maclaurin. 2018. *JAX: Composable transformations of Python+NumPy programs, version 0.3.13*. Available at https://github.com/jax. Accessed 1st Septemeber 2025.

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang *et al*. 2016. "Openai gym". *arXiv preprint arXiv:1606.01540*.

Chapuis, G., S. Eidenbenz, N. Santhi, and E. J. Park. 2015. "Simian integrated framework for parallel discrete event simulation on GPUS". In *2015 Winter Simulation Conference (WSC)*, 1127–1138 https://doi.org/10.1109/WSC.2015.7408239.

Chaturvedi, S., A. El-Gazzar, and M. van Gerven. 2024. "Foragax: An Agent-Based Modelling Framework Based on JAX". *arXiv preprint arXiv:2409.06345*.

Chen, H., A. Li, E. Che, J. Dong, T. Peng and H. Namkoong. 2024. "QGym: Scalable Simulation and Benchmarking of Queuing Network Controllers". In *Advances in Neural Information Processing Systems 37, Datasets and Benchmark Track*. December 10th–15th, Vancouver, BC, Canada, 92401–92419.

Freeman, C. D., E. Frey, A. Raichuk, S. Girgin, I. Mordatch and O. Bachem. 2021. "Brax–a differentiable physics engine for large scale rigid body simulation". *arXiv preprint arXiv:2106.13281*.

Fujimoto, R. M. 1990. "Parallel discrete event simulation". *Communications of the ACM* 33(10):30–53.

Greasley, A. 2005. "Using system dynamics in a discrete-event simulation study of a manufacturing plant". *International Journal of Operations & Production Management* 25(6):534–548.

Koyamada, S., S. Okano, S. Nishimori, Y. Murata, K. Habara, H. Kita *et al*. 2023. "Pgx: Hardware-accelerated Parallel Game Simulators for Reinforcement Learning". In *Proceedings of the 37th International Conference on Neural Information Processing Systems*. December 10th–16th, New Orleans, LA, USA, 45716–45743.

Lanctot, M., E. Lockhart, J.-B. Lespiau, *et al*. 2019. "OpenSpiel: A framework for reinforcement learning in games". *arXiv preprint arXiv:1908.09453*.

Lange, R. T. 2022. *Reinforcement Learning Environments in JAX*. Version 0.0.4, available at https://github.com/RobertTLange/gymnax. Accessed 1st Septemeber 2025.

Legrand, I. 2001. "Multi-threaded, discrete event simulation of distributed computing systems". *Computer Physics Communications* 140(1-2):274–285.

Liu, J. 2020. "Simulus: Easy Breezy Simulation in Python". In *2020 Winter Simulation Conference (WSC)*, 2329–2340 https://doi.org/10.1109/WSC48552.2020.9383886.

Makoviychuk, V., L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin *et al*. 2021. "Isaac Gym: High Performance GPU Based Physics Simulation For Robot Learning". In *Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. December 6th–14th, Virtual Conference.

Manuj, I., J. T. Mentzer, and M. R. Bowers. 2009. "Improving the rigor of discrete-event simulation in logistics and supply chain research". *International Journal of Physical Distribution & Logistics Management* 39(3):172–201.

Pernas-Álvarez, J. and D. Crespo-Pereira. 2024. "Open-source 3D discrete event simulator based on the game engine unity". *Journal of Simulation*:1–17.

Sanderson, K. 2023. "GPT-4 is here: what scientists think". *Nature* 615(7954):773.

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

Thulasidasan, S., L. Kroc, and S. Eidenbenz. 2014. "Developing Parallel, Discrete Event Simulations in Python—First Results and User Experiences with the SimX Library". In *Proceedings of the 4th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. August 28th–30th, Vienna, Austria, 188–194.

van der Ham, R. 2018. "salabim: Discrete Event Simulation and Animation in Python". *Journal of Open Source Software* 3(27):767.

Wu, S.-Y. D. and R. A. Wysk. 1989. "An application of discrete-event simulation to on-line control and scheduling in flexible manufacturing". *The International Journal of Production Research* 27(9):1603–1623.

Yu, Y. 2018. "Towards Sample Efficient Reinforcement Learning". In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. July 13th–19th, Stockholm, Sweden, 5739–5743.

Yun, H., L. Jun, and W. Xiangfeng. 2025. *DES-Gymnax: High-performance Discrete Event Simulator in JAX*. Available at https://github.com/hyyh28/DES-Gymnax. Accessed 1st Septemeber 2025.

Zhang, J., J. Kim, B. O'Donoghue, and S. Boyd. 2021. "Sample Efficient Reinforcement Learning with REINFORCE". In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*. February 2nd–9th, Virtual Conference, 10887–10895.

Zinoviev, D. 2024. "Discrete Event Simulation: It's Easy with SimPy!". *arXiv preprint arXiv:2405.01562*.

AUTHOR BIOGRAPHIES

YUN HUA is a postdoctoral researcher at the Antai College of Economics and Management, Shanghai Jiao Tong University. His primary research interests include reinforcement learning and agent-based modeling. His email address is hyyh28@sjtu.edu.cn.

JUN LUO is a professor in the Antai College of Economics and Management at Shanghai Jiao Tong University. His primary research interests include stochastic simulation and simulation optimization. His email address is jluo_ms@sjtu.edu.cn.

XIANGFENG WANG is a Professor in the School of Computer Science and Technology, East China Normal University. His primary research interests include agents (optimization, reinforcement learning, multi-agent reinforcement learning, and LLM-based approaches) and their applications. He can be reached at xfwang@cs.ecnu.edu.cn.