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

A STANDARD FRAMEWORK FOR AI-DRIVEN OPTIMIZATIONS IN VARIOUS COMPLEX DOMAINS

Tobias Bosse[1,2], Evangelos Angelidis[1], Fei Fei Zhang[3], Chew Wye Chan[3], Boon Ping Gan[3], Matthias Werner[1], and Andrej Gisbrecht[1]

[1]Dept. of Manufacturing Digitalization - AI-Solutions, Simulation and WIP-Flow Optimization, Robert Bosch GmbH, Reutlingen, BW, GERMANY
[2]Karlsruhe Institute of Technology (KIT), BW, GERMANY
[3]D-SIMLAB Technologies Pte Ltd, SINGAPORE

ABSTRACT

Deploying AI in production is a challenging task that necessitates the integration of various standalone software solutions, including Manufacturing Execution System, Simulation, and AI. The process of discussing, defining, and implementing interfaces requires both time and financial resources. In this paper, we propose a standardized interface framework that not only facilitates the training of Deep reinforcement learning agents in the semiconductor industry but also offers flexibility, extensibility, and configurability for other use cases and domains. To achieve this, we present OpenAPI interface definitions that enable automatic code generation, along with a set of data schemas that adequately describe the semiconductor domain for most use cases. By adopting this framework, the coupling between software modules is reduced, enabling researchers to work independently on their solutions. Moreover, it ensures compatibility among modules, allowing for plug and play functionality, and simplifies the deployment process in production.

1 INTRODUCTION

Smart factories are environments where machines, sensors, and systems collaborate to improve production processes and increase efficiency. Their increasing complexity presents difficulties for conventional optimization methods. One of the most challenging domains for optimization in Industry 4.0 is semiconductor manufacturing. The industry is characterized by high-volume production, long cycle times, high uncertainty and a high degree of automation, making it an ideal candidate for AI-driven optimization.

Developing simulators is a research field in itself, while emerging machine learning based solution methods, such as Deep reinforcement learning (DRL), have their own intrinsic complexity that requires specialized experts. The same applies for systems that handle large amounts of data and Manufacturing Execution Systems (MES). This makes the deployment of AI solutions in production environments so challenging: It requires the integration of these diverse software systems. Defining and implementing the necessary interfaces between these systems is time-consuming, resource-intensive and a source for technical debts (Sculley et al. 2015). Ehm, Fowler, Mönch, and Schorn (2024) state that a digital testbed would be beneficial to foster collaboration between AI and domain experts.

We believe that in order to manage the increasing complexity of each system, a looser coupling between them is necessary. This in turn necessitates defined communication to support cross-disciplinary collaboration. Moreover, it is advantageous to be able to quickly adapt to new solutions, especially in rapidly evolving fields where disruptive solutions can emerge any time. The presented framework can not only accelerate research on large scaled optimization tasks but also provides a bridge to the deployment in industrial IT environments.

Encapsulated modules are easier to scale, more reliable and provide better reusability and comparability, not only within but also across domains. The proposed framework helps avoid vendor lock-in and allows each discipline to work in their preferred programming language to focus on their core expertise. In particular, it allows to connect different modules and to fit model parameters, for example for Deep reinforcement learning agents.

Existing approaches in the industry do not distinguish between software from different disciplines. While this approach is effective for demonstrators it leads to massive challenges when it comes to scaling due to the coordination demands of the different disciplines.

We present a standard framework that maintains the exchange of information via customizable interfaces. The source code is published at AISSI-Dissemination (2024). In Section 2 we give an overview of the current state of the art. The standard framework is introduced in Section 3 and its application to semiconductor scheduling is presented in Section 4. Section 5 provides a discussion and an outlook. Section 6 summarizes the paper.

2 BACKGROUND

This section summarizes recent developments in the fields of optimization, simulation and similar frameworks.

2.1 Optimization

Heuristics like rule based dispatching are the established solution in the industry for many cases. They are responsive, scalable, and interpretable, but the quality of the solutions is limited. The inability of fixed rules to adapt to the intricate problem structure prevent them from reliably finding near optimal solutions to NP-hard problems. Metaheuristics also struggle to guarantee optimality for these challenging combinatorial optimization problems, although they typically generate better solutions than simple heuristics by intelligently exploring the search space (Dauzère-Pérès et al. 2024).

Numerical Optimization techniques utilize solvers on a system of equations and constraints to find a near optimal solution. The approach is limited by time and available computing power, making it difficult to apply to large-scale problems. In addition, the formulation of the optimization goal can be expensive.

The rise of **Reinforcement Learning** (RL) is pulled by the demand for more adaptive strategies. They are trained to take near optimal decisions by interacting with the environment. The training requires even more computing power than numerical optimization but the inference of the trained model is computationally cheaper.

RL is an established research field with many different algorithms that have become increasingly complex but also more powerful. Recently, DRL has become powerful enough to tackle even the hardest problems, such as semiconductor fabrication (Tassel et al. 2023; Park et al. 2020).

With more and more development in this field, there is an increasing need to compare existing solutions and foster research of new techniques. For this reason, a framework was developed in RL research to standardize the API between RL agents and the environments in which the agents act. OpenAI has introduced the Gym library, which is now widely adapted in the community. Many algorithms and environments compatible with this API have been released. The Gym library has since migrated to Gymnasium, maintained by the Farama Foundation (Farama Foundation 2024).

Even though hard tasks, such as chess and go, are computationally challenging, the corresponding environments are extremely simple to describe, and the transition function is usually straightforward to compute. In such cases, the environments are often implemented as a relatively simple utility function of the much more complex RL algorithm. Training a DRL agent with Monte Carlo Tree Search (MCTS) requires not only an environment but also the ability to manage the states of the environment in the tree. Although the Gym API does not provide a standard interface for this task, this was not an issue since for simple environments, this can be achieved by simply copying the environment object.

For research in the Semiconductor (SC) manufacturing area, there also exist standard models. One such model is the MiniFab model (El-Khouly et al. 2009), which describes a simple theoretical factory designed for operations research. Another model is the SMT2020 model (Kopp et al. 2020), which exhibits a complexity comparable to a real-world factory.

This model is already so complex that developing a closed-form solution for the factory dynamics is not feasible, and discrete event simulation is used instead. There are publicly available simulators that are also compatible with the Gym API, such as Kovacs et al. (2022), and could be used for research and training of DRL agents. For the simulation to be useful in production, the simulation results should be as close as possible to reality. Such simulators are available as commercial products and standalone software modules.

When using such tools for training DRL agents, it is thus not obvious how to integrate them with AI, and in particular, how to manage multiple states in the search tree. Unfortunately, the Gym library does not provide an API for state management. To alleviate this limitation, we propose a framework and API definition to allow a much richer interaction between AI and simulation and enable training of DRL agents in complex environments such as the SC domain.

2.2 Simulation

Discrete event simulation (DES) is widely adopted in the semiconductor industry to support decision-making in enhancing operational efficiency (Mönch et al. 2012; Kovacs et al. 2022). Typical modelling elements required are: (i) process flow that defines the steps and associated recipe to build a wafer, (ii) equipment dedication that defines the equipment that is capable of processing which recipe, (iii) process time that defines the time required to process a production lot or batch of a certain recipe at an equipment, (iv) random events such as equipment availability, step sampling, and rework/scrap processes, (v) dispatch rules that mimic the lot selection process implemented in the production., and (vi) equipment behaviour such as wafer, lot, or batch processing. It is also crucial that the model is initialized with the current state of the production and fed with the associated wafer start plan for realistic modelling of the production (Seidel et al. 2020). In this project, we adopt the D-SIMCON Simulator (D-SIMLAB 2024b) to create a high fidelity model of the production line.

Besides having the ability to model wafer fabrication processes in high fidelity, it is crucial that the simulation package is able to interact with the RL agent for training purposes. Key requirements are the ability to start, pause, resume, branch, and rollback of simulation state. The branching feature is especially crucial because RL training requires the simulation package to instantiate a new simulation from a specific time point to enable independent and parallel scenario explorations. Rollback is required to ensure that the RL agent is able to unwind a not favourable exploration path and explore new paths from an earlier simulation state. In the context of the paper, the decision parameter is lot dispatching. This means that the simulation package needs to accept lots' dispatching orders from the RL agent and execute in accordance with the decision made. The RL agent also requires feedback from the simulation package to measure the rewards of the decision made. The rewards can be measured at different time periods for different durations. The D-SIMCON Simulator was extended to support this requirement. More details can be found in Section 3.8.

2.3 Existing Frameworks

AI-driven optimization techniques have been applied to tackle complex problems in various domains, enabling organizations to make data-driven decisions and streamline processes. However, the lack of standardized frameworks and interfaces for integrating these optimization methods poses a significant challenge to their widespread adoption. This section reviews some of the notable existing frameworks and highlights their limitations.

Khalloof et al. (2020) introduced a framework for parallelizing population-based metaheuristics by leveraging microservices, container virtualization, and a publish/subscribe messaging paradigm. Their framework facilitates the hybridization of different parallelization models and has been successfully applied to solve optimization problems in distributed energy resources and renewable energy generation. Although the framework uses REST for data exchange, the interfaces are not standardized, limiting wider adoption.

Cao et al. (2021) developed an Ontology-Based Holonic Event Driven Architecture for Autonomous Networked Manufacturing Systems based on Java. However, their work does not standardize the message exchange protocols, which hinders interoperability with other systems.

Kovacs et al. (2022) presented a customizable simulator for evaluating RL algorithms on flexible job shop scheduling problems, including a Gym environment interface. However, it is solely compatible with Python, which limits its usability in other programming languages.

The Ray framework (Ray 2024) , introduced by Moritz et al. (2018), supports parallelized training of reinforcement learning algorithms by facilitating the asynchronous execution of distributed Python functions. While Ray has been used to implement scalable parallel optimization algorithms, it is limited to the interface of the gym environment, which offers only basic functionality.

Established solutions in industry are developed in various languages. The restriction to certain programming languages limits their applicability and adoption. Moreover it remains a lack of standardized interfaces that define what information needs to be exchanged between different disciplines and how to model complex environments at the core. Without such standards it is difficult for researchers and practitioners to collaborate, compare, and build upon existing optimization techniques.

3 STANDARD FRAMEWORK PROPOSAL

In the following, the architecture of the framework is introduced and proposals for standard interfaces are presented. The OpenAPI specifications of the modules serve as the complete definition of each module's interfaces. All other aspects of the framework, such as documentation and code, are derived from these OpenAPI specifications. Relying on the code generator maintains a high code quality and consistency. Domain independence is achieved by defining modules and their communication messages agnostic to specific domain entities. These entities are encapsulated in an interchangeable data schema, ensuring that the flow of actions and reactions between modules remains invariant, even if the data model is replaced. Although the application of the framework is illustrated using the semiconductor domain, it is possible to replace the data model with that of another domain without changing the remaining framework.

The proposed standards are the result of research collaborations between various companies. Simulation as well as AI and domain experts that are experienced in operating real fabs were involved in their development. In the project AISSI (2024), several companies with decades of experience combined their know-how. In addition, the results of the SC3-Project (2023) were incorporated with even more companies.

3.1 Main Modules

Figure 1 illustrates the high level architecture of our framework: The main modules are Simulation, AI, Strategy, and Persistence. These modules are designed to be modular, allowing for easy replacement or omission depending on the specific requirements of the system. In addition, the architecture supports the inclusion of multiple instances within a single module. For example, the AI module can accommodate multiple agents, each using different optimisation techniques.

- The Simulation is the digital twin of the real environment. It contains all the features relevant to the use case and can be treated as a copy of it, allowing experimentation, testing, and optimization without impacting the real environment.
- The AI module contains everything related to decision making logic. The term Artificial Intelligence is used here broadly for any algorithm that can calculate decisions. Various optimization techniques can be implemented here, such as DRL, genetic algorithms, numerical solvers or heuristics. As long as they follow the same scope, for example getting a current state and return scores for every lot in WIP, they can easily be exchanged for comparison or combination.

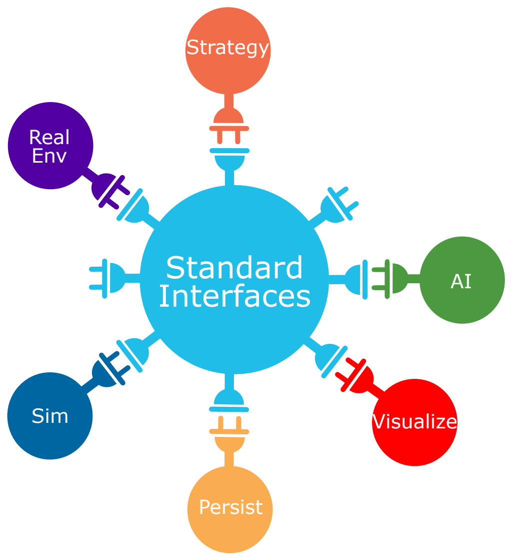

Figure 1: The high level architecture of the framework follows the principle of “divide and conquer”.

- The Strategy module’s task is to orchestrate higher level tasks, such as replaying a specific scenario or applying MCTS. The implementation of a new use case usually starts here.
- The Persistence module contains the data handling tasks. Solutions for extracting, transforming and loading historical data are based here. It stores relevant data and makes it available through the standard interface. It can be fed from different data sources such as MES or machine logs. It ensures that the data is efficiently managed and available for optimization tasks.

The proposed framework facilitates the integration of additional modules to extend functionality, as illustrated by the empty connectors in Figure 1. Examples of additional new modules include visualization and a real world connector. The visualization module could contain algorithms to visualize the current state of the real environment, the simulation, or the training progress of a reinforcement learning agent. The connector to the real environment could be used to deploy trained solutions in production. The interface is the same as for simulation.

3.2 Communication Adapters

For each module in the system, there are two automatically generated adapters based on the module’s specification: A receiving adapter, which is a server that provides specific services via incoming messages defined in the specification, and a sending adapter, which is a client that sends messages to the module in the format specified by the specification. Since both the server and client code are generated from the same specification, any potential mismatches or incompatibilities between them are eliminated. To message the module, any participant can integrate the generated corresponding client to send properly formatted messages. The adapter code is automatically generated in the desired target programming language using an OpenAPI generator (OpenAPI-Generator Contributors 2024). Many general purpose languages are supported. Developers can then reference the generated adapter code in their application.

The OpenAPI generator also enables the automatic generation of a human-readable website from the specification. This gives developers an overview of the API structure without having to read the specification file directly.

3.3 Event Driven Architecture

The event driven architecture enables asynchronous traffic, which facilitates scalability. The sender of a message does not wait for an immediate reply and thus prevents an idling waiting process. Instead, the sender receives an instant confirmation of message receipt, allowing it to continue processing without blocking. When a message is received, the associated function is triggered and executed in a separate thread. This enables parallel processing and ensures that the receiving component can handle incoming messages independently, without affecting the performance of other parts of the system. The receiving component can process the message and, if necessary, send messages to other participants as part of its own processing logic.

3.4 Standard Messages

The proposed standard defines several message types, some of which follow the same pattern. One message type for example contains the request for creating an instance of a module. The module constructs the instance, saves it locally, and an ID is returned to the requester. The body of the message is a serialized JSON string, consistent with the respective schema, which is constructed by the client. The format is defined in OpenAPI 3.0 and JSON Schema 2020-12. The receiving server parses the JSON string into the corresponding object and validates it against the defined schema. The possible responses defined in the specification follow the HTTP response status codes, e.g. "500" returns internal server errors according to the defined schema, which includes an error code and a descriptive message. Every message contains metadata defined in the "MessageMetaData" schema. It contains a universal unique identifier (UUID) of the sender, the message itself and the interaction. Also, the interaction type (request or response) is specified. This metadata helps track and manage the communication between modules, ensuring proper identification and context for each message exchanged. The persistence module provides four messages for every entity it manages: GET, POST, PUT and DELETE. GET is used to retrieve an entity or a collection of entities. POST is used to create new entities, while PUT is used to modify an existing entity, and DELETE is used to delete an entity or a collection of entities.

3.5 Semiconductor Domain Entities

The proposed semiconductor domain entities are intended to provide a comprehensive and standardized set of schemas for representing various aspects of the semiconductor manufacturing environment. The most common entities and their relationships are defined, covering most real-world use cases.

It can be viewed as a solid base configuration for representing an environment and is designed to be extended for individual environments while maintaining a consistent foundation.

Each organization has its own solutions for managing specific details, such as setup rules. The proposed approach allows everyone to extend the base configuration to meet their specific needs and existing solutions.

The schemas are uniquely named to avoid ambiguity and are intended to contain the relevant information needed for optimization tasks. In addition, information has been added that may be relevant to the optimization process or further calculations. These entity schemas cover various aspects of semiconductor manufacturing, such as products, routes, equipments, dedications, process parameters, chamber mappings, reticles, KPI-definitions and more. Laipple et al. (2018) provide a foundation which served as inspiration for defining the key entities and their relationships.

The entity schemas follow a hierarchical structure, with all objects inheriting from a base DataObject schema. This abstract base includes essential metadata such as a UUID, an internal ID, and the associated dataset ID.

Exemplary, the content of a state entity is displayed in Table 1. This schema represents the state of a real environment or a simulation, with a consistent format across both contexts.

"$>\{\dots\}$" indicates that further components belong to this schema. "[]" marks a list of elements. Most of the entities within the state schema are themselves defined by additional schemas, such as the list of

Table 1: State description.

State			
entity name	entity type	entity name	entity type
id	string($uuid)	equipments	[Equipment > {. . . }]
environmentId	string($uuid)	dedications	[Dedication > {. . . }]
stateId	string($uuid)	processParameters	[ProcessParameters > {. . . }]
currentDateTime	string($date-time)	chamberMappings	[ChamberMapping > {. . . }]
products	[Product > {. . . }]	chamberDedications	[ChamberDedication > {. . . }]
routes	[Route > {. . . }]	routeStagePreLeadTimes	[RouteStagePreLeadTime > {. . . }]
wip	[Wip > {. . . }]	equipmentState	[EquipmentE10State > {. . . }]

equipment states that represent the state of every equipment according to the SEMI E10 specification. The full collection of entities can be explored in the full specification available at AISSI-Dissemination (2024).

3.6 Extendability

A major strength of the OpenAPI specification is its extension support, which allows developers to add attributes or metadata to core elements. This modular and adaptable architecture allows use cases with different requirements to be addressed by selectively using core models and extending them as needed. For example, a use case may use only a subset of entities from the semiconductor domain, while incorporating additional entities or messages as needed. It is typical for companies to not reveal the specifics of their dispatching and scheduling strategies. However, by utilizing the semiconductor model described in subsection 3.5, it is possible for any individual to adapt it to their own specific requirements. The code generator streamlines extension integration. Developers can update the OpenAPI definition with necessary extensions and regenerate code in their chosen language. The updated generated code can then be referenced in the application code. New modules can also be added by defining new endpoints in the OpenAPI specification, guided by existing specifications.

3.7 Simulation-based Optimization

One use case which is addressed by this framework is simulation-based optimization. This approach examines performance under different parameters and scenarios. A typical workflow is outlined in the following pseudocode:

```
Procedure SimulationBasedOptimization
        Define potential sets of model parameters.    // in strategy module
        For each set of parameters
            Create AI Model with these parameters    // send request to AI module , receive ID
            Evaluate scenario performance with this model through simulation.
              // send request to simulation module, receive Key Performance Indicators
        End For
        Visualize the results    // send request to visualization module
End Procedure
```

During the evaluation process, the simulation module contacts the AI module several times to make sequential decisions. The objective is often non-trivial. It may be a multi-objective optimization where trade-offs between competing objectives must be carefully balanced. Hereby the relationship between competing KPI's can be highly nonlinear. Figure 2 shows an exemplary visualization of results of different scheduling algorithms. Cycle time and throughput are typically important in production, yet they are conflicting KPI's. Each point represents a simulated scenario with different parameters. For the non-optimal solutions (gray), there is another solution with a better value in one of the KPI's without sacrificing the value in the other.

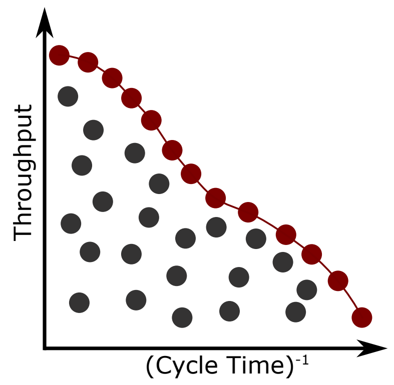

Figure 2: Pareto front.

The red points on the Pareto frontier represent Pareto-optimal solutions. Results above the Pareto frontier are infeasible.

A MCTS algorithm leverages this concept by applying it iteratively: A leaf of the search tree is linked to one state of the simulation. It is extended by different states which result from a number of different actions at the next time step. Then a selection method screens the whole search tree for the next leaf on which to deepen the search. The pseudocode for a MCTS in the framework is visualized in Figure 3 using a sequence diagram: Ten different decision algorithms are queried for each decision.

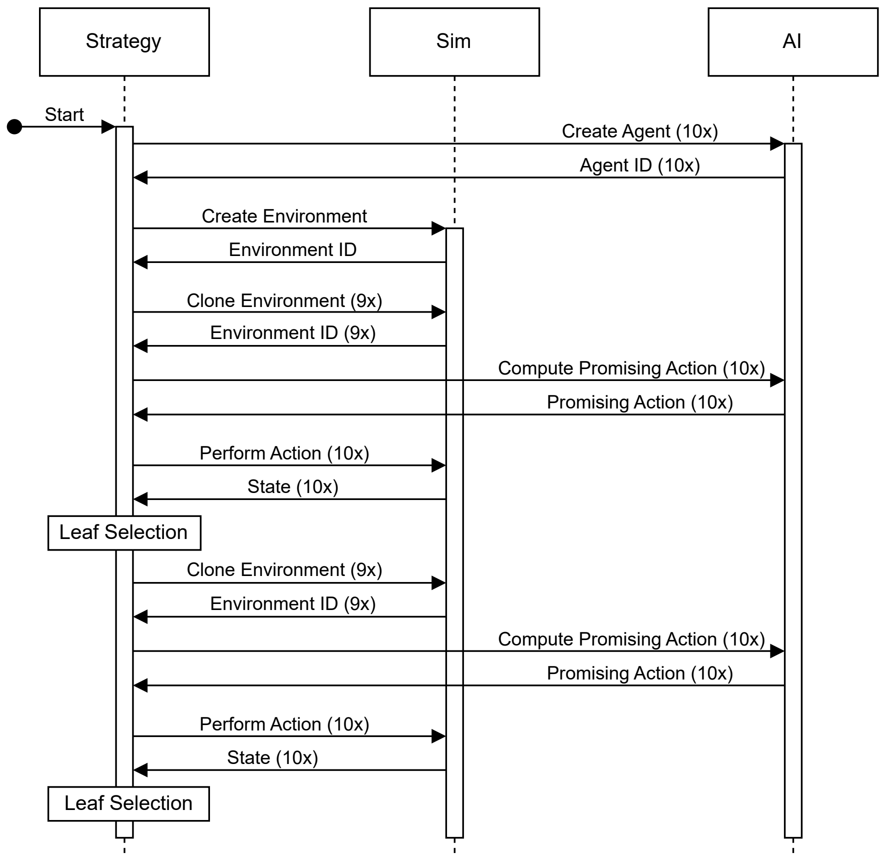

Figure 3: Sequence diagram of messages in a MCTS scenario.

3.8 Snapshot and Branching Mechanism in the Framework

MCTS is a powerful method for handling complex decision-making processes, especially in environments like semiconductor manufacturing, where numerous decisions interact in complicated ways. To enable MCTS to effectively manage complexities in our lot scheduling context, integrating a simulation platform with advanced execution control is crucial. The simulator used in this framework provides essential functionalities for managing simulation states and decision branches, which are pivotal for efficiently training MCTS.

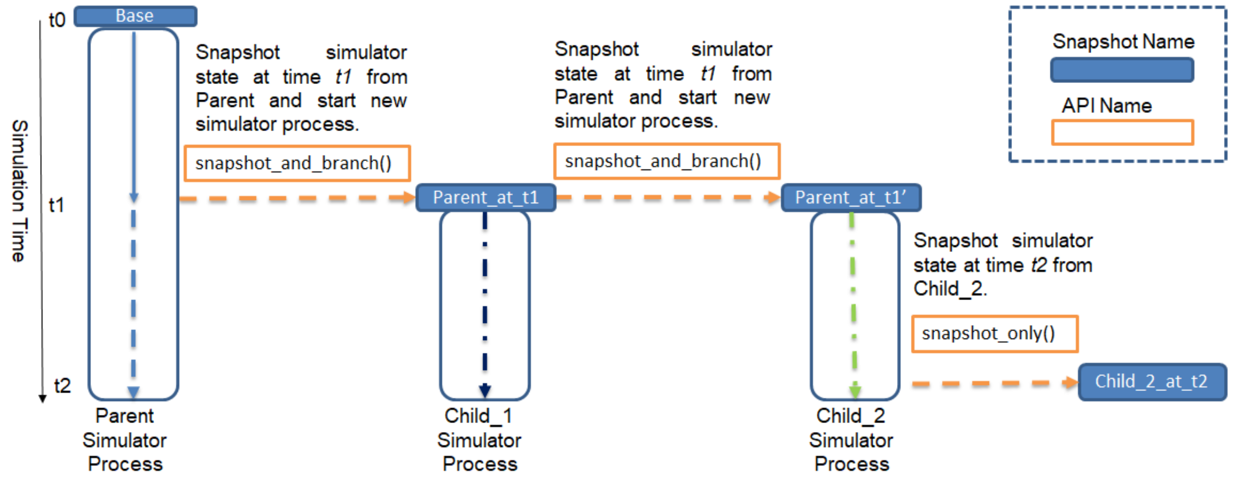
t0
Base
Simulation Time
t1
t2
Snapshot simulator state at time t1 from Parent and start new simulator process.
snapshot_and_branch()
Parent_at_t1
Snapshot simulator state at time t1 from Parent and start new simulator process.
snapshot_and_branch()
Parent_at_t1'
Snapshot Name
API Name
Snapshot simulator state at time t2 from Child_2.
snapshot_only()
Child_2_at_t2
Parent Simulator Process
Child_1 Simulator Process
Child_2 Simulator Process

Figure 4: Snapshot and branching mechanism of D-SIMCON Simulator.

Figure 4 shows the snapshot and branching functionality of the D-SIMCON Simulator. It allows for initiating multiple simulation instances from a single state at a specific time, enabling the parallel exploration of different actions. This capability is crucial for the MCTS method, where it is necessary to explore various search paths concurrently, focus on promising paths, and abandon less rewarding ones without redoing the entire simulation. For instance, the simulator can branch out Child_1 and Child_2 from the parent simulation instance at $t1$ to test different scheduling decisions, allowing these instances to run independently. This feature not only facilitates a broad exploration of potential strategies but also helps maintain an optimal number of active simulations, enhancing the overall efficiency of the simulation environment.

Furthermore, the simulator supports rollback functionalities that allow returning to any previously saved state. This is achieved by taking only snapshots of the simulation state, which are stored with unique labels, for example, *Child_2_at_t*2 from Figure 4, which could then be used to correspond to respective MCTS tree nodes. Such snapshots can be rolled back as required, thus eliminating the need to restart the Parent simulation from the beginning and significantly reducing the data transfer and storage overhead. Additionally, the framework's ability to manage and terminate simulation branches efficiently—either individually or collectively based on the tree structure—simplifies the overall management of simulation instances. These capabilities are integral to reducing computational resource consumption.

4 APPLICATION TO FRONTEND SCHEDULING

This section demonstrates the application of the framework to scheduling in the semiconductor domain. The proof of concepts code and setup are publicly available at the AISSI Homepage (AISSI 2024) for repeatability and further exploration. For the sake of simplicity and to maintain focus on the use of the framework's interfaces, we consider a MiniFab environment (El-Khouly et al. 2009) where the scheduling is performed using the First-In-First-Out (FIFO) principle.

The proof of concept consists of three modules: Simulation, AI and Strategy. In addition, it contains a template that serves as a reference for additional modules. In this case the generated clients and servers are integrated via an installation as a local pip package. The modules are independent of each other and could run on different machines. Unlike the other modules, which only start their respective servers, the strategy module starts the server in a parallel thread and triggers the start of the scenario in its main function.

The process begins with the strategy module triggering the creation of an optimization instance (agent) within the AI module and a virtual environment within the simulation module by means of a message constructed and sent using the client adapter of the corresponding module.

Upon receiving a message, the module sends an acknowledgement to confirm receipt. In parallel, a thread is started for the creation process. Once completed, this thread sends another message containing the new instance ID back to the strategy module.

The strategy's main function waits for the response of simulation and AI while the server process keeps running in the background, ready to handle incoming requests at any time. Once it has processed the receipt, the request to start the simulation is sent to the simulation module. The message contains the environment ID of the instance to be simulated and the agent ID of the AI instance to be queried for decisions.

This initiates the simulation for a duration of 48 simulated hours. Then main() program of the strategy module ends here. The remaining procedure is then executed based on the responses from different modules to the messages they receive. At simulated hourly intervals, the current state is transmitted to the optimization agent, which in turn generates a new dispatching list valid for the subsequent hour of simulation. Each lot is assigned a score. Waiting lots are then sorted according to their score in descending order. At the end of the simulation period, KPI's from the run are sent back to the strategy module for analysis and evaluation. The framework is sufficiently robust to facilitate an exchange between factories in the semiconductor environment, e.g. from MiniFab to SMT2020, without the necessity of modifying the remaining code.

5 DISCUSSION

The proposed standards reflect a wide collection of know-how and experience. The data schemas are designed to cover the most standard concepts from SC fabrication and the communication events allow training and deployments of diverse algorithms from simple heuristics to complex DRL agents. The objective of this paper is not to develop a new AI model but to propose a standard communication to train, evaluate, and deploy arbitrary AI models. The framework was developed and validated on a first productive scheduling use case. In the near future, we plan to further use this framework to train a reinforcement learning agent and deploy it in a productive environment.

We believe that discussions between interdisciplinary experts on common interfaces can be very effective, with the proposed standards serving as a robust foundation. They provide a comprehensive set of commonly used attributes, and if something is missing from the collection, the schemas can be flexibly extended on a case-by-case basis.

By implementing the communication adapter, encapsulated solutions, such as commercial simulators, can preserve their intellectual property while seamlessly integrating their solutions into the customers' framework. This facilitates the adoption of AI-driven optimization techniques in real-world applications and helps to bridge the gap between academic research and industrial practice. First industrial grade simulator already supports the standard framework interfaces to provide specific functionality for AI training (D-SIMLAB 2024a). Furthermore, the adoption of the standard framework could lead to the creation of a standard benchmark for simulation and AI applications. This would provide a consistent and reliable way to evaluate and compare different optimization techniques. This, in turn, would foster innovation and accelerate progress in the field. We look forward to further discussion and refinement of these standards as they are adapted and applied.

6 CONCLUSION

As the complexity of Industry 4.0 environments increases, the high level division of disciplines becomes increasingly important. This work proposes a novel framework architecture to facilitate the development of data-driven optimization techniques in complex domains. The standardized OpenAPI interfaces enable automatic code generation and provide data schemas that adequately describe the semiconductor domain for most use cases. It is designed to allow greater design freedom within individual modules, accelerate the adaptation to new modules and facilitate reinforcement learning in large-scale cyber-physical systems. By adopting this framework, researchers and experts from different disciplines can collaborate effectively. This is a first contribution to possible future industry standards.

ACKNOWLEDGEMENTS

This work has received funding from the Eureka ITEA-4 Cluster AI Call 2020 under grant agreement no. 20212 (project AISSI – Autonomous Integrated Scheduling for Semiconductor Industry), which is financially supported by the Federal Ministry for Economic Affairs and Climate Action in Germany and Enterprise Singapore. This support is gratefully acknowledged.

REFERENCES

AISSI 2024. "Autonomous Integrated Scheduling for Semiconductor Industry". https://aissi-project.com/, accessed 1st May 2024.

AISSI-Dissemination 2024. "Exploitable results of AISSI project". https://aissi-project.com/en/publications_and_links, accessed 28th June 2024.

Cao, H., X. Yang, and R. Deng. 2021, January. "Ontology-Based Holonic Event-Driven Architecture for Autonomous Networked Manufacturing Systems". *IEEE Transactions on Automation Science and Engineering* 18(1):205–215.

OpenAPI-Generator Contributors 2024. "OpenAPI Generator". https://openapi-generator.tech/, accessed 1st May 2024.

D-SIMLAB 2024a. "D-SIMCON AI Training and Evaluation Platform". https://d-simlab.com/event/d-simlab-launches-a-high-fidelity-simulation-platform-for-training-of-artificial-intelligent-enabled-agents-and-evaluation-of-scheduling-policies, accessed 15th April 2024.

D-SIMLAB 2024b. "Forecaster and Scenario Manager". http://d-simlab.com/category/d-simcon/products-d-simcon/forecaster-and-scenario-manager, accessed 15th April 2024.

Dauzère-Pérès, S., J. Ding, L. Shen, and K. Tamssaouet. 2024, April. "The flexible job shop scheduling problem: A review". *European Journal of Operational Research* 314(2):409–432.

Ehm, H., J. Fowler, L. Mönch, and D. Schorn. 2024. "Decision-Making Techniques for Smart Semiconductor Manufacturing (Dagstuhl Seminar 23362)". *Dagstuhl Reports* 13(9).

El-Khouly, I. A., K. S. El-Kilany, and A. E. El-Sayed. 2009. "Modelling and Simulation of Re-entrant Flow Shop Scheduling: An Application in Semiconductor Manufacturing". In *2009 International Conference on Computers & Industrial Engineering*. July 6th-9th, Troyes, France, 211-216.

Farama Foundation 2024. "Gymnasium Documentation". https://gymnasium.farama.org/, accessed 1st May 2024.

Khalloof, H., P. Ostheimer, W. Jakob, S. Shahoud, C. Duepmeier and V. Hagenmeyer. 2020. "A Distributed Modular Scalable and Generic Framework for Parallelizing Population-Based Metaheuristics". In *Parallel Processing and Applied Mathematics*, 432–444. Cham: Springer International Publishing.

Kopp, D., M. Hassoun, A. Kalir, and L. Monch. 2020, November. "SMT2020—A Semiconductor Manufacturing Testbed". *IEEE Transactions on Semiconductor Manufacturing* 33(4):522–531.

Kovacs, B., P. Tassel, R. Ali, M. El-Kholany, M. Gebser and G. Seidel. 2022, May. "A Customizable Simulator for Artificial Intelligence Research to Schedule Semiconductor Fabs". In *2022 33rd Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 1–6. Saratoga Springs, NY, USA: IEEE.

Laipple, G., S. Dauzere-Peres, T. Ponsignon, and P. Vialletelle. 2018, December. "GENERIC DATA MODEL FOR SEMICONDUCTOR MANUFACTURING SUPPLY CHAINS". In *2018 Winter Simulation Conference (WSC)*, 3615–3626. Gothenburg, Sweden: IEEE https://doi.org/10.1109/WSC.2018.8632349.

Mönch, L., J. W. Fowler, and S. J. Mason. 2012. *Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems*, Volume 52. Springer Science & Business Media.

Moritz, P., R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang *et al.* 2018, October. "Ray: A Distributed Framework for Emerging AI Applications". In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 561–577. Carlsbad, CA: USENIX Association.

Park, I.-B., J. Huh, J. Kim, and J. Park. 2020. "A Reinforcement Learning Approach to Robust Scheduling of Semiconductor Manufacturing Facilities". *IEEE Transactions on Automation Science and Engineering* 17(3):1420–1431.

Ray 2024. "Ray documentation". https://docs.ray.io/en/latest/index.html, accessed 1st May 2024.

SC3-Project 2023. "C3 Semantically Connected Semiconductor Supply Chains". https://sc3-project.automotive.oth-aw.de/, accessed 1st May 2024.

Sculley, D., G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner *et al*. 2015. "Hidden Technical Debt in Machine Learning Systems". In *Advances in Neural Information Processing Systems*, edited by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Volume 28: Curran Associates, Inc.

Seidel, G., C. F. Lee, A. Y. Tang, S. L. Low, B. P. Gan and W. Scholl. 2020. "Challenges Associated with Realization of Lot Level Fab Out Forecast in a Giga Wafer Fabrication Plant". In *Proceedings of the 2020 Winter Simulation Conference*, 1777–1788. IEEE https://doi.org/10.1109/WSC48552.2020.9384046.

Tassel, P., B. Kovács, M. Gebser, K. Schekotihin, P. Stöckermann and G. Seidel. 2023. "Semiconductor Fab Scheduling With Self-Supervised And Reinforcement Learning". In *2023 Winter Simulation Conference (WSC)*, 1924–1935 https://doi.org/10.1109/WSC60868.2023.10407747.

AUTHOR BIOGRAPHIES

TOBIAS BOSSE is a research engineer in the semiconductor operations team by Robert Bosch GmbH. His research interests include the complex job shop scheduling problem, its influencing factors and machine learning applications. His email address is tobias.bosse@de.bosch.com.

EVANGELOS ANGELIDIS is a simulation engineer and IT-architect of the semiconductor operations team by Robert Bosch GmbH. He received his M.S. degree in computer science from Dresden University of Technology. He has years of experience in developing simulation and optimization software in the domain of production and logistics. His research focuses on the simulation and optimization of complex assembly lines such as semiconductor and aerospace manufacturing. His email address is evangelos.angelidis@de.bosch.com.

FEIFEI ZHANG is a Product Manager at D-SIMLAB Technologies (Singapore). He is responsible for lot scheduling and dispatching products for customers in the semiconductor industry. He earned a Bachelor of Computer Science from the National University of Singapore. His email address is zhang.feifei@d-simlab.com.

CHEW WYE CHAN is a Software Engineer of D-SIMLAB Technologies (Singapore). He holds a Master of Computing degree from the National University of Singapore. He is currently working as a doctoral student at the School of Computer Engineering at Nanyang Technological University, Singapore. His research interests include machine learning, data science, and simulation-based optimization. His email address is chew.wye@d-simlab.com.

BOON PING GAN is the CEO of D-SIMLAB Technologies (Singapore). He has been involved in simulation technology application and development since 1995, with primary focus on developing parallel and distributed simulation technology for complex systems such as semiconductor manufacturing and aviation spare inventory management. He led a team of researchers and developers in building a suite of products in solving wafer fabrication operational problems. He was also responsible for several operations improvement projects with wafer fabrication clients which concluded with multi-million dollar savings. He holds a Master of Applied Science degree, specializing in Computer Engineering. His email address is boonping@d-simlab.com.

MATTHIAS WERNER is a senior manager in the Semiconductor Operations Organization by Robert Bosch GmbH. He received his Diploma from the Karlsruhe Institute of Technology and his PhD in particle physics from the University of Freiburg. He is responsible for the material flow optimization and application of AI in the semiconductor manufacturing network of Robert Bosch GmbH. His email address is matthias.werner@de.bosch.com.

ANDREJ GISBRECHT is a simulation engineer at the Robert Bosch GmbH. He received his Diploma from the Clausthal University of Technology and his PhD in computer science from the Bielefeld University. He currently works on artificial intelligence and numerical optimization techniques in the area of semiconductor fabrication. His email address is andrej.gisbrecht@de.bosch.com.