

DREAM-ON GYM: A Deep Reinforcement Learning Environment for Next-Gen Optical Networks

Nicolás Jara¹ ^a, Hermann Pempelfort¹ ^b, Erick Viera¹, Juan Pablo Sanchez¹, Gabriel España²
and Danilo Borquez-Paredes² ^c

¹Department of Electronics Engineering, Universidad Tecnica Federico Santa Maria, Valparaíso, Chile

²Faculty of Engineering and Sciences, Universidad Adolfo Ibáñez, Viña del Mar, Chile

Keywords: Optical Networks, Framework, Deep Reinforcement Learning, Simulation Technique.

Abstract: A novel open-source toolkit for a straightforward implementation of deep reinforcement learning (DRL) techniques to address any resource allocation problem in current and future optical network architectures is presented. The tool follows OpenAI GYMNASIUM guidelines, presenting a versatile framework adaptable to any optical network architecture. Our tool is compatible with the Stable Baseline library, allowing the use of any agent available in the literature or created by the software user. For the training and testing process, we adapted the Flex Net Sim Simulator to be compatible with our toolkit. Using three agents from the Stable Baselines library, we exemplify our framework performance to demonstrate the tool's overall architecture and assess its functionality. Results demonstrate how easily and consistently our tool can solve optical network resource allocation challenges using just a few lines of code applying Deep Reinforcement Learning techniques and ad-hoc heuristics algorithms.

1 INTRODUCTION


Deep Reinforcement learning (DRL) has recently emerged as a powerful machine learning paradigm that can be used in diverse real-world scenarios. Recent studies suggest that reinforcement learning could soon become an essential tool in various industries, including manufacturing, energy, healthcare, and finance (Naeem et al., 2020). In particular, DRL has been successfully applied to the field of telecommunication architectures such as optical networks (ON), where it can be used to build various solutions such as classification techniques based on data gathering, routing decision protocols, and traffic scheduling algorithms, among others (Zhang et al., 2020). These applications can help optical networks operate more efficiently by improving the quality of their services and reducing their overall deployment and operation costs.


Optical networks handle massive data volumes, and optimizing data routing is crucial (Klinkowski et al., 2018). Reinforcement learning, a trial-and-


error machine learning technique, offers a solution where traditional methods falter in finding efficient routes. Traditional optimization struggles with the complex constraints of optical networks (Naeem et al., 2020), but reinforcement learning can discover more effective and innovative solutions.

The main challenge facing the application of DRL to optical networking is the complexity of the problem domain. Since optical networks consist of a complex set of interconnected components, each with its functions and limitations, it is difficult for the system to make consistent decisions concerning traffic routing through the worldwide network. To overcome this challenge, researchers have developed many models that can be used to approximate the effects of different network components and test various resource allocation strategies (Chen et al., 2019; El Sheikh et al., 2021; Morales et al., 2021). Using these models, it is possible to identify possible bottlenecks in the network and ways of improving network traffic performance and quality.

In optical networks, reinforcement learning can improve resource allocation for two main purposes. The first is to design efficient routing strategies that consider the current allocation of resources within the network and the anticipated changes in demand

^a  <https://orcid.org/0000-0003-2495-8929>

^b  <https://orcid.org/0000-0002-6896-5787>

^c  <https://orcid.org/0000-0001-6590-2329>

that may occur in the future. The second is to develop and implement a dynamic resource allocation scheme that can respond to real-time changes in the network environment to maximize network performance. For example, a reinforcement learning algorithm could design new routing strategies for better use of the link resources (e.g., wavelengths, frequency spectrum, fiber cores, spectral bands), ensuring a certain quality level and enabling available capacity to handle future increases in traffic demand. It is also possible to implement dynamic resource allocation schemes that adjust the allocation of resources to different parts of the network based on real-time measurements of traffic patterns and current resource availability.

The main benefit of using reinforcement learning in optical networks is that it allows these systems to make optimal decisions based on current conditions and predicted changes to the network environment. An advantage of this approach is that it allows the network to respond quickly and efficiently to new situations without reprogramming or re-configuring to perform a new function. Another advantage of using reinforcement learning in optical networks is that it enables these systems to make decisions based on previous experiences rather than human input or intervention. This advantage makes it highly resistant to human error, making it much more reliable than a human-operated system.

Using reinforcement learning in optical networks presents a significant drawback: the high implementation and maintenance costs. Operators must invest time and resources into training the system to adapt to changing network conditions. This complexity increases operational costs and the difficulty of implementation. Setting up these models is time-consuming, with a steep learning curve, often requiring development from scratch. Moreover, numerous parameters must be configured for training and evaluation, varying depending on the problem context, necessitating expertise in optical communications and machine learning techniques. To our knowledge, one toolkit intends to accelerate the implementation of Deep Reinforcement Learning for Optical Networks (Natalino and Monti, 2020), called Optical-RL-GYM. This toolkit provides a hierarchical application for static and elastic optical network resource allocation problems. However, the same hierarchical architecture makes adapting the software to new optical networks difficult and may be hard to learn due to a lack of documentation.

For this reasons, we developed DREAM-ON GYM, a Deep Reinforcement LEarning frAMework

for Optical Networks¹. Following the principles established by the OpenAI GYMNASIUM (Towers et al., 2023), we propose a straightforward and versatile framework for solving many current and future optical network architecture resource allocation problems, such as routing, spectrum or wavelength allocation, spectrum band or fiber core selection. Even more, it can be adapted for any other optical architecture context. This tool reduces the time and complexity of using Deep Reinforcement Learning in Optical Network architectures, providing easy-to-use functions and modules. The framework is compatible with the Stable Baselines library (Hill et al., 2018), allowing the use of any agents available in the literature, even though any operator can create their agents or include any ad-hoc heuristic algorithm solutions if needed. Finally, for the training and testing evaluations, we adapted the Flex Net Sim Simulator (Falcón et al., 2021) to perform with our tool appropriately, harvesting its straightforward and fast performance for simulating optical communications infrastructure.

The remainder of the paper is as follows: the software architecture of our tool is presented in Section 2. The Flex Net Sim simulator is succinctly explained in Section 3. In Section 4, we exemplify the use, simplicity, and versatility of our tool. Finally, Section 5 presents further conclusions and final comments.

2 DREAM-ON GYM TOOLKIT

Deep Reinforcement learning is an area of artificial intelligence concerned with developing computer programs that can learn to perform tasks without being explicitly programmed to do so. The ultimate goal of reinforcement learning is to help machines become more intelligent and learn to do things independently without having to be instructed or programmed by humans. Although reinforcement learning is still in its infancy, it is beginning to gain significant traction in the computer engineering and computer science fields.

DRL systems are made of two main interacting entities: agent and environment. The agent performs a specific action following a strategy (or policy) that modifies the environment's state. In return, the environment computes a goodness score, called a reward, that assesses the selected action's quality. Maximizing the cumulative reward, the agent learns the best way to perform a particular task in the environment.

The framework consists of a customized gym environment compatible with elastic optical networks

¹Code available at: <https://gitlab.com/IRO-Team/dream-on-gym-v2.0>

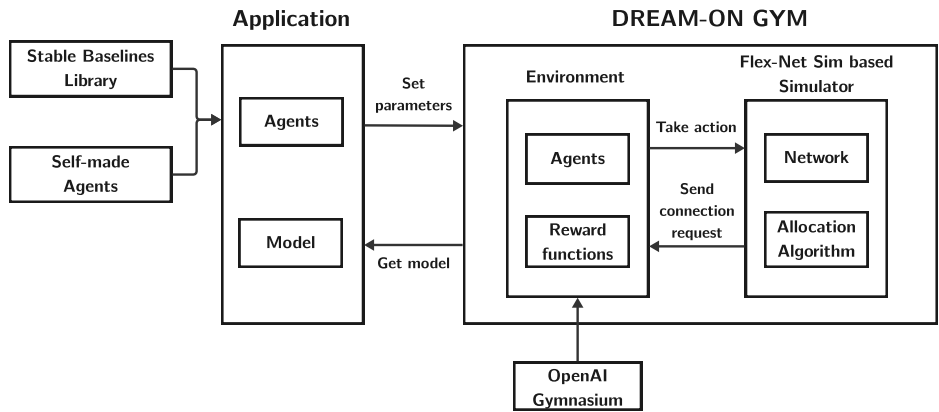


Figure 1: DREAM-ON GYM architecture.

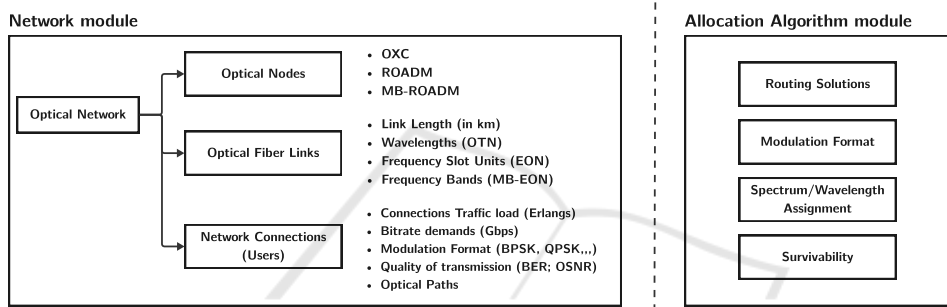


Figure 2: Networks and Allocation Algorithm modules on our DREAM-ON GYM Architecture.

(EON) and multiband elastic optical networks (MB-EON), which can be used for different optical architectures. Figure 1 shows the architecture of the implemented framework. The application created by the user must select an agent and set the network parameters, such as topology, routes, and bitrates. After that, the environment is initialized, and the agent starts training. The simulation component of the framework sends connection requests to the environment composed of the triplet (source, destination, bitrate). In contrast, the environment decides where to assign this request through the agent. After training, the framework returns a model to the application that will be used to allocate resources in this network.

Within the framework are overwritten `__init__`, `step`, and `reset` methods, which are mandatory for any gym environment.

- The `__init__` method is responsible for initializing the environment. To meet the dynamism required by the environment, a particular elastic or multiband optical network simulation module inspired by Flex Net Sim (Falcón et al., 2021) was created, which is initialized here. This module works with three files: topology, routes, and bitrates. The topology file contains the connection information of the network nodes and links. The

routes file contains the existing routes between each pair of network nodes, which can be more than one. The bitrate file contains information on the bands, modulation formats, and bitrates that the simulator can use. The module generates a connection request between any network nodes, using a uniform distribution, and with a bitrate obtained from a JSON file associated with a modulation format and the number of slots. The parameters normally set in this simulator are the number of connections and arrival and departure rates. In addition, a resource allocation function must be set, coded, and passed as a parameter to the simulator.

- The `step` method is responsible for processing the connection requests one by one. In this way, each connection is worked separately and sent to the agent set before training.
- Finally, the `reset` method allows restarting the episodes.

Figure 2 shows the structure of the Network and Allocation Algorithm modules, reflecting optical network architectures, their users (network connections), and the standard problems to be solved with a network operation perspective.

```

1 env = gymnasium.make("rlonenv-v0")
2 obs, info = env.reset()
3
4 #set space
5 env.action_space = gymnasium.spaces.
   MultiDiscrete(np.array([4]))
6 env.observation_space = gymnasium.
   spaces.MultiDiscrete([100] *
   2720)
7
8 #Set reward function and states
9 env.setRewardFunc(reward)
10 env.setStateFunc(state)
11
12 #Set topology and connection routes
13 env.initEnviroment(NETWORK, PATHS)
14
15 #Connections settings and warm-up
   time
16 env.getSimulator().goalConnections =
   100
17 env.getSimulator().setMu(1)
18 env.getSimulator().setLambda(1000)
19 env.getSimulator().setLambda(FF)
20 env.getSimulator().init()
21
22 #Train process
23 env.start()
24 env.getSimulator().setAllocator(
   agent_algorithm)
25
26 policy_args = dict(net_arch=5*[128],
   activation_fn=th.nn.ReLU)
27 model = TRPO(MLP_TRPO, env, verbose
   =0, tensorboard_log, policy,
   gamma)
28 model.learn(timesteps, log)
29
30 #Save trained model
31 model.save('MODEL')

```

Listing 1: DREAM-ON GYM code example.

The Network module holds the network architecture, considering the optical nodes, optical fiber links, and network connections. These components must reflect the optical architecture considered in the study. We included three possible architectures: Optical Transport Networks (OTN), Elastic Optical Networks (EON), and Multiband Elastic Optical Networks (MB-EON). In considering Optical Transport Networks (OTN), the nodes perform as OXC (Optical Cross Connect), and each link capacity is composed of 80 wavelengths of 50 GHz each wavelength, corresponding to the standard communications on the C frequency band. The Elastic Optical Networks consider nodes with ROADM (Reconfigurable Optical Add-Drop Multiplexing) and Wavelength Selective Switches (WSS), including optical fiber links with

344 frequency slot units (FSU) of 12,5 GHz each for the C-Band. Last, we consider the MB-EON architecture, with MB-compatible optical nodes such as MB-ROADM (Multiband-ROADM), and the capacity of the links is composed of several frequency bands corresponding to the C, L, S, and E Bands with 344, 480, 760, and 1136 FSU each.

The Allocation Algorithm module comprises the algorithms to solve the standard problems to be found in optical communications, such as the routes to be selected by each network connection (Routing), the appropriate modulation format for each network connection considering the Quality of Transmissions and their path distances (Modulation Format), a piece of frequency spectrum on each link belonging to each path (Wavelength/Spectrum Assignment), or including fault tolerance capabilities to the network (Survivability). This module connects with the Controller module on the Simulator, which will be explained in the following section. Remark that, in this module, the solutions can be obtained by the DRL agents or any ad-hoc heuristic procedure in the literature.

Listing 1 exemplifies the few lines of code required to execute our tool. We will refrain from showing Python dependencies and some functions needed, such as reward and state functions, but they can be found in the available Gitlab code.

The OpenAI gymnasium framework is called in the first two lines, and the environment is reset. From lines 5 to 6, the environment action and observation space is set. In lines 9 and 10, the reward and state functions are chosen. Line 13 sets the network topologies and the possible paths to be chosen. From lines 16 to 20, the connection settings are configured, such as the number of total connection arrivals, interarrival (λ), and service rate (μ). We perform a warm-up time execution on line 19 (Function called FF), executing the common First-Fit algorithm. The warm-up time is when the simulation will run before collecting results, allowing the queues (and other aspects in the simulation) to get into conditions typical of normal running conditions in the system you are simulating.

The training process is executed from lines 23 to 27, in which the environment is started, and the search for network resources such as paths, wavelengths, and slot units is done using the *agent_algorithm* (see GitLab code example) using one of the Stable baseline agents (in this case using TRPO). Based on Q-learning or policy learning, the agents are responsible for observing the environment and taking action to improve the cumulative reward. These agents can be obtained using the Stable Baseline library² or created by the developer if needed. All the learning processes

²<https://stable-baselines.readthedocs.io/en/master/>

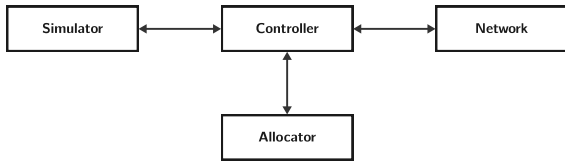


Figure 3: Flex Net Sim Simulator.

are saved on line 28.

Finally, the trained model can be stored on line 31 for further use.

In summary, the user must create the resource allocation function (in the example called *agent_algorithm*, choose the agent, and load the previously mentioned JSON files to use the framework. The user must also code the state and the reward function, yet we provide a standard example on the Gitlab code. The environment is trained using the optical networks environments provided and the agents of choice, and finally, the model can be saved for further evaluation.

3 SIMULATION TOOL

The tool presented in this article uses the core of the Flex Net Sim simulator (Zitkovich et al., 2023) through its flexible optical networks option. This core is in charge of generating the connection and disconnection requests that will be processed by the agent and will finally end up in an assignment or a blocking to the network. As presented in Figure 3, Flex Net Sim is a very versatile simulator whose architecture is composed of four components: Network, Controller, Simulator, and Allocator, shown in Figure 3.

The Network component configures all the network resources and comprises three Classes: Link, Node, and Network. The Link class contains the status of each link, including slot occupancy. The Node class is a glue between the links that form the network. Finally, the Network class manages the network resources, generating the optical network graph. In this way, the Network component contains the state of the simulated network.

The Controller component is in charge of managing the allocation of resources over the network. The purpose of this component is to allocate or de-allocate network resources. It comprises a class with the same name as the component (Controller), which contains two essential methods: Allocate and Unallocate resources. These methods call for the different methods of the Network component to carry out the connections. In turn, this component maintains the status of the active network connections. The main difference between this component and the Network component

is that the Network component manages the state of the network links. Still, these states are not associated with a connection. On the other hand, the Controller component maintains the identifier of each connection, associating it with a network state. In this way, when it is necessary to release a connection, the Controller component gives the order in which network slots must be released.

The Simulator component contains all the random and event-driven logic. Here are initialized random variables related to the arrival and departure of connections and the choice of the bitrate of each incoming connection. At the same time, this component stores a list of events that are updated with each new connection arrival. These events can be one of the two types: Arrival and Departure. Arrival events correspond to the arrival of a connection request to the network, while departure events correspond to the release of network resources by a connection. The simulator component communicates with the Controller component, sending it a triplet (src, dst, b) with the information of a new connection, where src corresponds to the source node, dst is the destination node, and b is the bitrate required by this connection. The Controller component is in charge of asking the Allocator component how to allocate these resources.

Thus, in general terms, the simulation works as follows: The Simulator component generates a new connection request and sends it to the Controller component so that it is in charge of assigning it. The Controller component takes the request and sends it to the Allocator component, which checks where this connection can be allocated in the network. The Allocator component communicates with the agent, which processes the request and responds to the Allocator component, indicating the links and slots where this connection should be allocated. The Allocator component takes the information and passes it to the Controller component, which sends it to the Network component (saving the connection information). The Network component assigns the slots in their respective links, delivering a confirmation code to the Controller component, which sends this information to the Simulator component, which follows with the next event.

4 EXAMPLES AND APPLICATIONS

This section illustrates our framework’s straightforward usage and versatility for creating DRL models for Optical Networks by comparing several agent decisions on dynamic elastic optical networks. This

Table 1: Simulation parameters.

Parameter	Value
Network Topology	UKNet
Network connections	420
Candidate routes	5
Bit rate (Gbps)	10, 40, 100, 400, 1000
Number of episodes	200
Training Steps	1000 per episode
Agents	PP0, A2C and DQN
Neural Network Layers	2
Neural Network neurons	64
Learning rate	$3 \cdot 10^{-4}$
Reward function	1, -1

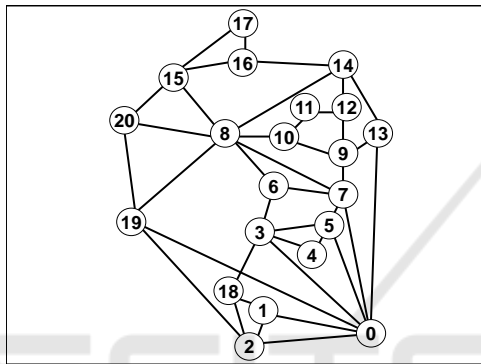


Figure 4: UKNet Network Topology, with 21 nodes, 78 links, and 420 node pair source-destination.

example focuses on the Routing, Modulation format, and Spectrum Assignment problem (RMSA). The agents choose a path among five precomputed shortest paths and choose a modulation format according to the bitrate demanded (10, 40, 100, 400, or 1000 Gbps). The bitrate is set to the connections randomly with a uniform distribution (using the same seed for replicability of the different exercises). On the other hand, the search for a portion of a contiguous frequency spectrum on the chosen path (spectrum assignment) is performed by using the First-Fit algorithm. In this example, the environment is set as a list of arrays, each representing an optical network fiber link. The components of each array represent a frequency slot unit available (or not) to be used by the network connections. In this sense, the agents will choose for each connection requesting transmission a subset of links composing the source-destination path and a portion of the FSUs within the link arrays. Remark that this is just an example, and the operator adjusts the environment to represent optical network operation.

Table 1 presents the main parameters set on the framework in this example. We train three different agents, PPO (Schulman et al., 2017), A2C (Mnih

et al., 2016) and DQN (Fan et al., 2020) provided by the Stable Baseline library on the UKNet network topology (21 nodes, 78 links, and 420 node pair source-destination). The learning rate was set to $3 \cdot 10^{-4}$, using the standard configuration of the agents with two layers with 64 neurons each. We perform 10^5 training steps, comprising 200 episodes with 1000 training steps each.

The reward function was set using the standard values in the literature (Morales et al., 2021), returning a value equal to 1 when the action is successful and -1 when rejected. Figure 5 illustrates the cumulative reward obtained by the PPO, A2C, and DQN agents solving the routing, modulation format, and spectrum assignment problem on the UKNet network topology. In the figure, we can see that DQN starts with a slow learning curve. However, after 50 episodes, it improves, reaching results similar to those of A2C and PPO agents. On the other hand, A2C reaches a steady cumulative reward of around 250 within a few episodes.

Figure 6 complements the episode blocking probability performed by the PPO, A2C, and DQN agents, solving the routing, modulation format and spectrum assignment problem on the UKNet network topology. Similarly, we can see that A2C quickly obtains a steady value episode blocking probability. Meanwhile, DQN takes more training to obtain results similar to those of A2C and PPO agents.

Figure 7 presents in-depth charts showing the routes the same three agents decide during the training process in percentage. Figure 7a shows the path usage distribution when 10.000 training steps are performed, and Figure 7b shows the same path usage distribution for 100.000 training steps. The training decisions changing over time can be seen in these two figures. PPO and DQN choose a path that is more distributed at the beginning. Meanwhile, A2C prioritizes its decision on the shortest path (Path 1). In the end, at 100.000 training steps, the three agents prioritize the shortest path.

The previous exercise shows merely a simple example of all the possible studies that can be done within our tool. For example, the previous example demands hyperparameter tuning to be competitive with the best-performing solutions in the literature. Our toolkit is compatible with all optimization processes and hyperparameter tuning available for Deep Reinforcement Learning, such as setting different hyperparameters such as neurons, layers, learning rates, or any other hyperparameter needed. Our framework is compatible with hyperparameter tuning software such as Optuna software (Akiba et al., 2019). The operator can perform many different agents, create their

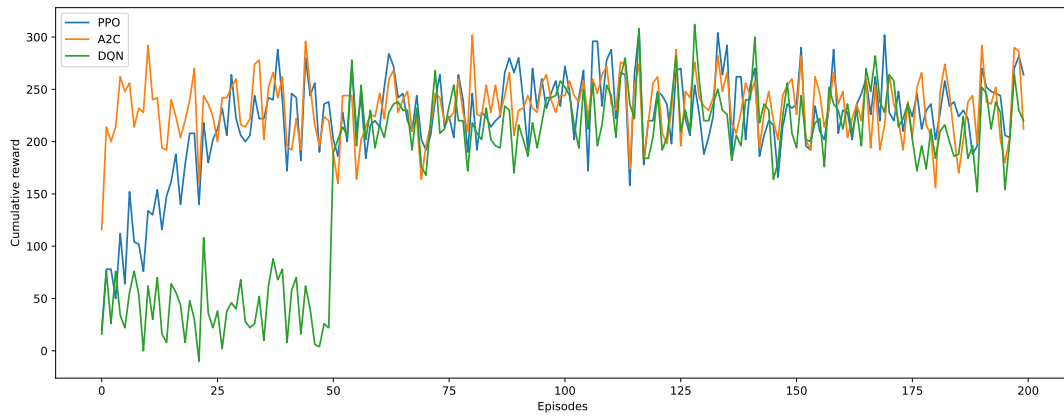


Figure 5: Training results showing cumulative reward for PPO, A2C, and DQN agents for the routing problem on the UKNet network topology using 200 episodes of 1000 training steps each.

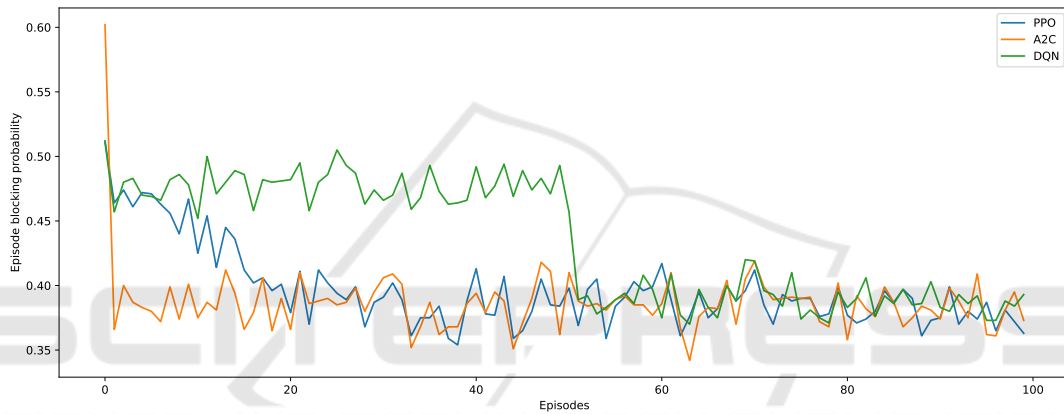
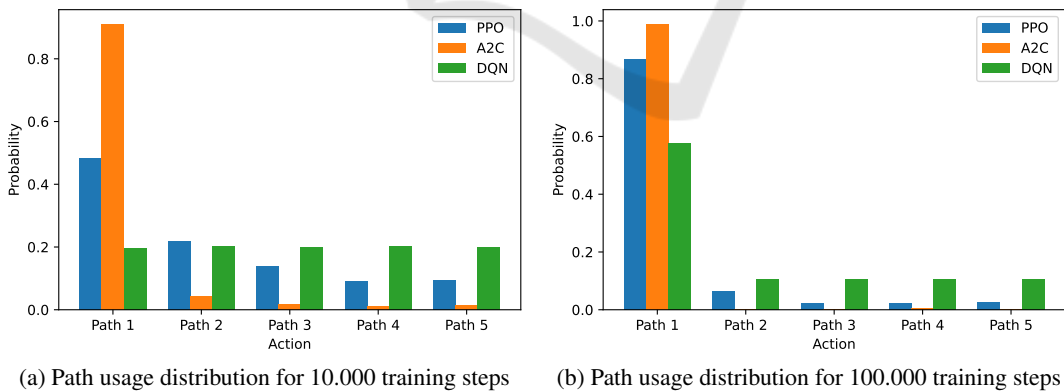


Figure 6: Training results the episode blocking probability for PPO, A2C, and DQN agents for the routing problem on the UKNet network topology using 100 episodes of 1000 training steps each.



(a) Path usage distribution for 10.000 training steps

(b) Path usage distribution for 100.000 training steps

Figure 7: Path usage distribution obtained by training PPO, A2C and DQN Agents for the routing problem on the UKNet network topology for (a) 10.000 training steps and (b) 100.000 training steps.

own reward functions, and reduce action space techniques and invalid action masking, among other optimization processes for DRL.

In addition, our tool can perform Interpretable and Explainable (XAI) artificial intelligence methods,

such as applying highly interpretable agents based on Linear Regression, Decision Tree, Logistic regression, or Random Forest methods or applying Imitation Learning to best-performing neural networks techniques. This may allow understanding of the

agent's decision to create new and non-trivial routing and spectrum assignment protocols for optical networks.

5 CONCLUSIONS AND FINAL REMARKS

We present a new Deep Reinforcement Learning Framework for Optical Networks called DREAM-ON GYM. The framework allows the implementation of deep reinforcement learning in a straightforward and versatile manner to solve resource allocation problems in optical network architectures, such as routing, spectrum or wavelength allocation, and band or core selection in multiband or multicore architectures. To this end, we provide a set of functions and modules allowing agents and environments to interact to train the models. The application relies on adapting the Flex Net Sim Simulator to train and evaluate the agents. This way, we reduce the time and complexity of implementing and evaluating DRL for Optical Network problems.

We exemplify the usability of our framework by choosing a path with three different agents in an elastic optical network. With this example, we demonstrated the easy-to-use tool showing the difference in the performance of the three agents. In addition, by creating an app, we can make an application for a simple evaluation and training for a given optical network context.

In future works, we will use the framework to allow interpretability and generalization of the models while training and evaluating DRL in optical networks and adding new capabilities to the framework, such as compatibility for survivability problems.

ACKNOWLEDGEMENTS

Financial support from FONDECYT Iniciación 11220650 is gratefully acknowledged.

REFERENCES

- Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631.
- Chen, X., Li, B., Proietti, R., Lu, H., Zhu, Z., and Yoo, S. B. (2019). Deepprmsa: A deep reinforcement learning framework for routing, modulation and spectrum assignment in elastic optical networks. *Journal of Lightwave Technology*, 37(16):4155–4163.
- El Sheikh, N. E. D., Paz, E., Pinto, J., and Beghelli, A. (2021). Multi-band provisioning in dynamic elastic optical networks: a comparative study of a heuristic and a deep reinforcement learning approach. In *2021 International Conference on Optical Network Design and Modeling (ONDM)*, pages 1–3. IEEE.
- Falcón, F., España, G., and Bórquez-Paredes, D. (2021). Flex net sim: A lightly manual.
- Fan, J., Wang, Z., Xie, Y., and Yang, Z. (2020). A theoretical analysis of deep q-learning. In *Learning for Dynamics and Control*, pages 486–489. PMLR.
- Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines. <https://github.com/hill-a/stable-baselines>.
- Klinkowski, M., Lechowicz, P., and Walkowiak, K. (2018). Survey of resource allocation schemes and algorithms in spectrally-spatially flexible optical networking. *Optical Switching and Networking*, 27(September 2017):58–78.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.
- Morales, P., Franco, P., Lozada, A., Jara, N., Calderón, F., Pinto-Ríos, J., and Leiva, A. (2021). Multi-band environments for optical reinforcement learning gym for resource allocation in elastic optical networks. In *2021 International Conference on Optical Network Design and Modeling (ONDM)*, pages 1–6. IEEE.
- Naeem, M., Rizvi, S. T. H., and Coronato, A. (2020). A gentle introduction to reinforcement learning and its application in different fields. *IEEE Access*, 8:209320–209344.
- Natalino, C. and Monti, P. (2020). The optical rl-gym: An open-source toolkit for applying reinforcement learning in optical networks. In *2020 22nd International Conference on Transparent Optical Networks (ICTON)*, pages 1–5. IEEE.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Shen, A. T. J., and Younis, O. G. (2023). Gymnasium.
- Zhang, Y., Xin, J., Li, X., and Huang, S. (2020). Overview on routing and resource allocation based machine learning in optical networks. *Optical Fiber Technology*, 60:102355.
- Zitkovich, M., Saavedra, G., and Bórquez-Paredes, D. (2023). Event-oriented simulation module for dynamic elastic optical networks with space division multiplexing. In *Proceedings of the 13th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, volume 1, pages 295–302.