M. Ferens *et al*., "Deep Reinforcement Learning Applied to Computation Offloading of Vehicular Applications: A Comparison," *2022 International Balkan Conference on Communications and Networking (BalkanCom)*, Sarajevo, Bosnia and Herzegovina, 2022, pp. 31-35, https://doi.org/10.1109/BalkanCom55633.2022.9900545

Deep Reinforcement Learning Applied to Computation Offloading of Vehicular Applications: A Comparison

Mieszko Ferens
Universidad de Valladolid and Aalborg Universitet
Valladolid, Spain/Denmark
mieszko.ferens@ribera.tel.uva.es

Diego Hortelano
Universidad Rey Juan Carlos and Universidad de Valladolid
Madrid/Valladolid, Spain
0000-0001-7776-6862

Ignacio de Miguel
Universidad de Valladolid
Valladolid, Spain
ignacio.miguel@tel.uva.es
0000-0002-1084-1159

Ramón J. Durán Barroso
Universidad de Valladolid
Valladolid, Spain
0000-0003-1423-1646

Juan Carlos Aguado
Universidad de Valladolid
Valladolid, Spain
0000-0002-2495-0313

Lidia Ruiz
Universidad de Valladolid
Valladolid, Spain
0000-0001-6241-5998

Noemí Merayo
Universidad de Valladolid
Valladolid, Spain
0000-0002-6920-0778

Patricia Fernández
Universidad de Valladolid
Valladolid, Spain
0000-0001-5520-0948

Rubén M. Lorenzo
Universidad de Valladolid
Valladolid, Spain
0000-0001-8729-3085

Evaristo J. Abril
Universidad de Valladolid
Valladolid, Spain
0000-0003-4164-2467

***Abstract*—An observable trend in recent years is the increasing demand for more complex services designed to be used with portable or automotive embedded devices. The problem is that these devices may lack the computational resources necessary to comply with service requirements. To solve it, cloud and edge computing, and in particular, the recent multi-access edge computing (MEC) paradigm, have been proposed. By offloading the processing of computational tasks from devices or vehicles to an external network, a larger amount of computational resources, placed in different locations, becomes accessible. However, this in turn creates the issue of deciding where each task should be executed. In this paper, we model the problem of computation offloading of vehicular applications to solve it using deep reinforcement learning (DRL) and evaluate the performance of different DRL algorithms and heuristics, showing the advantages of the former methods. Moreover, the impact of two scheduling techniques in computing nodes and two reward strategies in the DRL methods are also analyzed and discussed.**

Keywords—Deep Reinforcement Learning, Vehicular Applications, Computation Offloading, Edge Computing

I. INTRODUCTION

The popularity of portable devices with computational resources and connectivity is ever growing. This trend goes parallel to the development of new services for these devices, with increasing computational requirements. As the demand of these services grows, the design of the devices must include more resources, but this can prove difficult, as limitations such as physical space, overheating, energy requirements and costs become an issue. To face this problem, edge computing, and in particular a recent proposal by the European Telecommunications Standards Institute (ETSI), the multi-access edge computing (MEC) paradigm, has become a promising solution [1]. Edge computing proposes the deployment of computing resources in the edge of the network, and thus near the operating devices. Unlike other paradigms such as cloud computing, this paradigm ensures lower communication delays while still providing strong computational resources.

For vehicular applications, a computing strategy can be associated with the development of a hierarchy of network elements equipped with computational resources: the vehicle, a MEC site, a regional data center (RDC), and the cloud. Obviously, additional network resources like wireless access points and a wired (typically optical) network between the vehicles and the MEC site (and other computing points) are also required for connectivity. However, in this paper we focus on computing resources and just consider the throughput and delay of the networking resources between computing nodes. The resulting hierarchy of the computing nodes is such that the higher the node in the hierarchy, the more computational resources are present (Fig. 1). This comes at the cost of higher communication delays when moving up in the hierarchy, and so, decisions must be made to select where to process each task that the vehicles generate. For this aim, different solutions have been proposed, and the use of deep reinforcement learning (DRL) is a promising technique for decision making accounting for a variety of factors [2].

In this paper, we model the problem of computation offloading of vehicular applications to solve it by means of DRL, and we compare four DRL algorithms together with four heuristics, with the aim of determining which type of algorithms is preferable for the implementation of controllers for these systems. Moreover, two different techniques to schedule tasks within computing nodes are analyzed: (i) placing the tasks at the end of a queue for each computing resource, and (ii) exploiting voids in the scheduling tables of those computing resources.

II. VEHICULAR ENVIRONMENT AND SCHEDULING TECHNIQUES IN COMPUTING NODES

We consider a vehicular environment based on a hierarchical network topology with a set of computing

This work has been supported by Consejería de Educación de la Junta de Castilla y León and the European Regional Development Fund (Grant VA231P20), and the Spanish Ministry of Science of Innovation and the State Research Agency (Grant PID2020-112675RB-C42 funded by MCIN/AEI/10.13039/501100011033 and RED2018-102585-T). This work has also received funding from the European Union Horizon 2020 research and innovation programme under the grant agreement No 856967.

resources located at different distances from the vehicles (Fig. 1), as well as a set of applications which are launched at each vehicle (Table I).

In this section, we first define a set of heterogeneous applications with different features and computing and delay requirements. Then, we describe the network topology and its computing resources, as well as two different strategies that the computing nodes can use to schedule the execution of arriving tasks.

A. Application set

We assume that vehicles run a set of heterogeneous applications independently. Each application in each vehicle generates tasks, i.e., sets of data (data packets) that require to be processed in any node of the network or the vehicle. Applications are heterogeneous and thus have different features in terms of tasks generation (frequency and size) and of computing and delay requirements. Thus, based on [3-5], five different features have been defined, and a set of six different applications has been considered, as shown in Table I. The features of the applications are defined as follows:

1) Processing cost (A_c): CPU cycles necessary to process each bit of the task (data packet).

2) Input data (A_{in}): Size in bits of the data packet to be processed.

3) Output data (A_{out}): Size in bits of the data packet to be returned to the vehicle after the node processes the task.

4) Maximum latency (A_d): Maximum tolerable delay in milliseconds since the vehicle initiates the task (generates the input data packet) until it receives the output of the processing (reception of the output data packet).

5) Average generation period (A_t): Average time between generating consecutive data packets (tasks).

B. Network elements and scheduling techniques

For the network topology, we have considered the structure shown in Fig. 1, which is based on [6-11]. It consists of four types of nodes: vehicle, MEC, RDC and cloud, each with increasing computing capabilities.

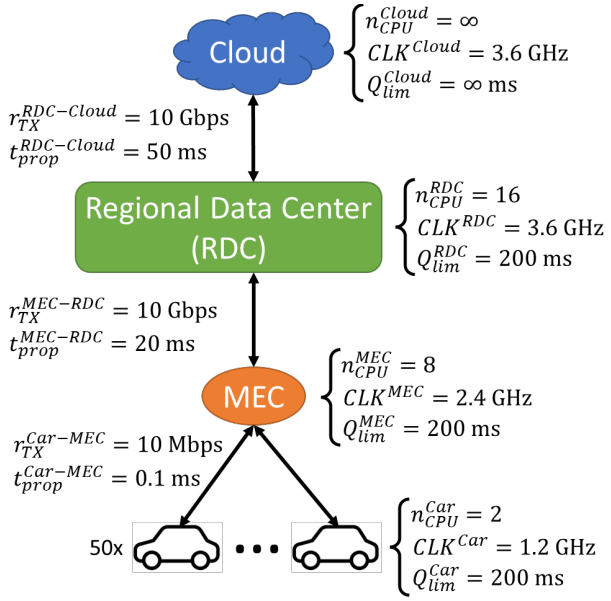

Fig. 1. Hierachical network topology with computing resources.

TABLE I. APPLICATION SET AND PARAMETERS

App	*Processing cost, A_c* (CPU cycles/bit)	*Input data, A_{in}* (kbits)	*Output data, A_{out}* (kbits)	*Maximum latency, A_d* (ms)	*Avg. generation period, A_t* (ms)
#1	1	700	200	1	100
#2	400	10	5	4	100
#3	1000	900	25	500	100
#4	3000	100000	25000	200000	100
#5	200	650	500	200	100
#6	500	40	1	30	100

Nodes are characterized by the number of computing units, i.e., CPUs (n_{CPU}^{node}), their processing rate (CLK^{node}), and the size of their scheduling queues to process pending tasks (Q_{lim}^{node}). Links connecting nodes are characterized by their data rate (r_{TX}^{link}) and their propagation delay which also includes the delay associated to communication protocols (t_{prop}^{link}). As we are dealing with vehicles, factors such as energy levels are not critical and are omitted.

As we have just mentioned, each CPU has a scheduling queue that determines the order in which computation tasks are processed. These tasks are represented by data packets generated by the applications initiated in the vehicles, and which require processing. A controller (agent) must decide at which node (vehicle, MEC, RDC or cloud) to execute that processing, and then triggers a reservation process in the scheduling queues of the selected node. The reservation process is not managed by the agent, but rather by the node itself. A review of current CPU scheduling techniques can be found in [12]. Based on that reference, and also inspired by void filling techniques for resource allocation [13], two methods for scheduling reservations are considered and analyzed in this paper and shown in Fig. 2: reservations with planning and without planning.

A reservation causes the task to be allocated to a dynamic time slot in the scheduling queues of the node based on the estimated arrival time of the data packet ($t_{arrival,est}$) and its estimated processing time at that node ($t_{proc,est}$). The arrival time of the data packet is estimated based on transmission and propagation delays from the vehicle to the computing node. The processing time is estimated based on the amount of data to process (A_{in}), the processing cost (A_c) associated to the application, and the processing rate of the node (CLK^{node}). Multithreading has not been considered and each task must be processed by a single CPU.

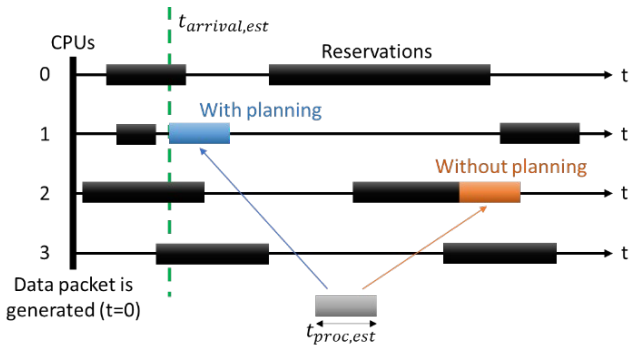

Fig. 2. Example of reservation with and without planning.

When reservation with planning is used, the allocated slot for processing that task will be placed as early as possible considering all cores in the node. On the other hand, when reservation without planning is used, the same rule applies but the slot cannot be placed between others and must be allocated at the end of a queue. Thus, in the example shown in Fig. 2, when planning is used, the task is scheduled to be processed by CPU number 1 (the earliest available slot to process the task since its arrival), but when planning is not used, the task is assigned to CPU 2 (earliest end of the queues).

III. Decision-Making Algorithms for Computation Offloading

To determine where to process each task two different approaches have been analyzed: the use of deep reinforcement learning techniques, and the use of heuristics.

A. Deep reinforcement learning algorithms

Reinforcement learning is a subset of machine learning whose algorithms are characterized by learning through interaction with an environment. An agent observes the state of the environment and interacts with it by executing actions. Then, the agent receives a reward from the environment, and the environment typically evolves to a different state. By means of these interactions, the agent can learn to make better decisions by trying to maximize the cumulative reward. In scenarios where the number of states is huge, approximation techniques like deep learning are generally used, thus becoming deep reinforcement learning methods.

We next describe our proposal to define observations, rewards, actions and learning procedures in order to solve the computation offloading problem by means of DRL.

As we have just mentioned, to interact with the environment, i.e., the network and the vehicular applications, the DRL agents receive an observation and a reward. For each generated task a request is passed to the DRL agent. This is done in the form of an observation, which is a vector (O) with multiple elements as described in Eq. (1),

$$O = [Q_1, \dots, Q_X, A_c, A_{in}, A_{out}, A_d, D_1, \dots, D_K, V_1, \dots, V_M], \quad (1)$$

where K represents the number of candidate nodes to process that task (which includes the vehicle generating the task), X is the number of candidate CPUs to process that task (i.e., the sum of the number of CPUs in each candidate node), and M is the number of MEC sites in the network. There are four main elements in the observation. The first is a set of values (Q_x), which represent the fraction of time that each CPU is currently available (i.e., has no reservations). It is calculated as shown in Eq. (2), dividing the available time in the scheduling queue of the CPU by the limit time for scheduling in the node where that CPU is located (Q_{lim}^{node}),

$$Q_x = \frac{\text{available time for scheduling in queue } x}{Q_{lim}^{node}}. \quad (2)$$

The second set of elements, $(A_c, A_{in}, A_{out}, A_d)$, represents the value of those parameters for the corresponding application (Table I). However, in the implementation, those values are normalized by the maximum value of all defined applications. The third is a set of values (D_k) calculated as shown in Eq. (3), which compares the maximum allowed latency with the expected delay due to data sending and processing at node k (without taking into account the state of the queues),

$$D_k = \max\left(0, 1 - \frac{t_{T,est}^k}{A_d}\right). \quad (3)$$

A value of D_k close to 1 indicates that the total estimated delay ($t_{T,est}^k$) is much lower than the maximum latency (A_d), while a value close to 0 implies that there is not much margin. If the calculation produces a value less than 0 (estimated total delay greater than maximum latency), it is truncated to 0. A value of 0 implies that, in principle, the task cannot be processed complying with latency requirement. The fourth and final part is a set of values (V_m) that represent the MEC sites in the network. All these values are 0 except for the MEC associated to the base station to which the vehicle generating the task is connected, which will be 1.

With each observation, the environment also provides a reward, which is the sum of the rewards associated to the successful execution (or not) of the tasks processed since the previous step. We have defined and analyzed two different types of rewards for each task: the penalty reward (PR) and the weighed binary reward (WBR), as shown in (4) and (5),

$$PR = \begin{cases} -1000, & \text{if unable to process} \\ -100 - (t_T - A_d), & \text{if } t_T > A_d \\ 0, & \text{if } t_T \leq A_d \end{cases} \quad (4)$$

$$WBR = \begin{cases} -A_p, & \text{if } t_T > A_d \text{ or unable to process} \\ +A_p, & \text{if } t_T \leq A_d \end{cases} \quad (5)$$

where t_T is the total real delay that was required to complete the task, and A_p is a weight that can be used to give priority to some applications. PR provides a null reward if a task has been processed fulfilling latency requirements, and a negative reward otherwise. In contrast, WBR provides a constant positive reward when a task is processed complying with latency requirements, and a constant negative reward otherwise.

The DRL agent performs the action of choosing the node that will process each task. Therefore, its action space is equal to the number of candidate nodes. In this paper, we consider that the candidate nodes are always the cloud node, the RDC node, the MEC node to which the vehicle generating the task is associated, and the vehicle itself. The policy for decision making is updated based on the observations, actions and rewards obtained, and for that aim we have employed four different DRL algorithms, Double Deep Q-Network (DDQN) [14], State-Action-Reward-State-Action (SARSA) [15], Persistent Advanced Learning (PAL) [16] and Trust Region Policy Optimization (TRPO) [17], with the help of the open-source Python library ChainerRL [18]. All implemented DRL agents use a neural network with one hidden layer of sixty neurons, with all neurons activated by a hyperbolic tangent. The discount factor is set to 0.995 and the exploration is constant with a probability of 20% on each time step.

B. Heuristics

For comparison purposes we have also implemented four heuristics that select between four possible choices for processing (at the vehicle, MEC, RDC, or cloud):

1) Maximum distance processing (MDP): Delegates the processing of tasks as high in the hierarchy as possible, as long as an estimate of the total delay (considering propagation and computing times) does not exceed the maximum allowed latency for the application.

2) Uniform distribution processing (UDP): Makes offloading decisions with equal probably between all nodes in the hierarchy.

3) Cloud processing (CP): Always processes in the cloud.

4) Local processing (LP): Always processes locally, i.e., in the vehicle.

IV. Simulation Results

To test the performance of the different methods, we have run simulations in a custom environment implemented using the Python library OpenAI Gym [19]. This environment simulates the network shown in Fig. 1, with a set of 50 vehicles running all the applications defined in Table I, and assuming that all vehicles are served by a single MEC site. All applications have been assumed to have the same priority ($A_p = 1$). The interarrival time of the tasks associated with each application has been generated according to an exponential distribution with average A_t, as shown in Table I, where the size of the tasks and their computation and latency requirements are also indicated. Essentially, each application running at each vehicle has its own exponential distribution of mean A_t, with a maximum precision in the simulator of 10^{-10} milliseconds. The interaction of the DRL agent and the environment is as described in the previous section.

For the DRL algorithms, the simulations have consisted of three distinct phases: learning, validation and testing. For heuristics, only the testing phase has been used, as the others are unnecessary. The learning phase consists of 10^6 time steps (requests for task offloading) during which the DRL agents update their policies. The validation and testing phases consist of 10^5 time steps during which the DRL agents do not update their policies and follow the one they obtained during the previous learning phase. All phases have 10^3 additional time steps at the beginning for warming-up the simulator (not counted towards metrics). Each DRL agent has been trained eight times in each scenario and the performance of the eight policies found have been evaluated in a validation phase. Then, the best one has been evaluated a second time (test phase) to arrive at the final result. During the validation/testing phases, we have assessed the average reward obtained, and also the success rate, which is defined as the percentage of tasks that are executed complying with latency requirements. However, we focus on the last metric, as it is directly related to network performance.

Table II shows the success rates of the DRL methods and heuristics. Note that heuristics do not use rewards (thus, PR/WBR are not used), but computing nodes handle reservation queues differently if planning is activated or not (independently of which technique is used to make offloading decisions). Looking at the results, the use of DRL is considerably superior to the heuristics in terms of success ratio. However, the heuristics have the advantage of being much simpler from a computational point of view.

In the studied scenario, it is clear that simple solutions such as processing locally (LP), in the cloud (CP) or distributing the load uniformly (UDP) are not good enough, and accounting for the state of the network and the type of application is important. The MDP heuristic gets better results, as it makes decisions estimating propagation and computing delays in the different nodes and considering the application requirements in terms of latency. However, compared with these methods, the DRL techniques get a significant improvement in performance. The agents are able to learn how to act in different environments considering factors that are hard to consider when designing heuristics and, as such, obtain better results. With proper training we observe a success rate of up 90.38% and 92.12% for the DDQN and PAL agents, respectively. This is considerably superior to the best result of MDP, which gets a success rate of around 80.32%.

TABLE II. Experimental Success Rates

		With planning		Without planning	
		PR	***WBR***	***PR***	***WBR***
Heuristics	MDP	80.32%		78.39%	
	UDP	30.69%		30.61%	
	CP	33.62%		33.17%	
	LP	44.57%		44.92%	
DRL methods	DDQN	78.04%	**90.38%**	88.64%	82.55%
	SARSA	78.27%	**84.64%**	77.86%	79.09%
	PAL	85.60%	90.36%	78.22%	**92.12%**
	TRPO	64.14%	**80.80%**	63.10%	74.50%

Regarding the reward strategy, we observe a general increase in success rate for every DRL agent when using WBR instead of PR. The only exception to this is DDQN when planning is not used, as it gets a better success rate with PR. That said, DDQN still obtains its best result with WBR (when planning is used). As for the difference between reservations with or without planning, there are no signs of improvement for the heuristics apart from a slight difference for MDP. However, the DRL methods tend to perform better with the use of planning, that is, the DRL methods generally learn to exploit the advantages of the planning strategy adopted by the computing nodes, thus translating in an increase of the success ratio.

V. Conclusion

This paper has studied the performance of four DRL algorithms, and four heuristics, for computation offloading of vehicular applications in an edge computing and communication network. For the defined network structure with 50 vehicles, each running a set of six different applications, the success rate (i.e., the percentage of processed tasks with compliance to their maximum latency requirement) has been analyzed.

The results show the superiority of DRL over the use of heuristics, which do not have the necessary flexibility to account for all relevant factors. Improving the design of these heuristics becomes exponentially difficult as the complexity of the problem increases, while DRL algorithms can adapt to this increase in complexity while utilizing very limited information during their training and testing. As seen in the defined observation for the agents, they only require simple estimates for the state of the network in the form of rough

occupancy of the queues of each core and predefined knowledge of the running applications, network nodes and delay estimations. Additionally, this paper has shown that a correct definition for the reward the agents receive when learning is important, as it can improve certain metrics. In this case, the proposed WBR method is a good reward strategy for maximizing success rates. We have also shown that the success rate can be generally improved if WBR is combined with a planning strategy in the computing nodes. Finally, we have also shown that the DDQN (Double Deep Q-Network) and PAL (Persistent Advanced Learning) are the DRL algorithms providing the best results. Future work will include analyzing more complex scenarios (e.g., with multiple MEC locations) and optimizing the underlying neural network or the DRL agents.

REFERENCES

[1] European Telecommunication Stardards Institute (ETSI), "Multi-access Edge Computing" [Online]. Available: https://www.etsi.org/technologies/multi-access-edge-computing

[2] A. Shakarami, M. Ghobaei-Arani and A. Shahidinejad, "A survey on the Computation Offloading Approaches in Mobile Edge Computing: A Machine Learning-Based Perspective", Computer Networks, vol. 182, 2020.

[3] H. Liu, H. Zhao, L. Geng and W. Feng, "A Policy Gradient Based Offloading Scheme with Dependency Guarantees for Vehicular Networks", IEEE Globecom Workshops (GCWkshps), pp. 1-6, 2020.

[4] H. Liu, H. Zhao, L. Geng, Y. Wang and W. Feng, "A Distributed Dependency-Aware Offloading Scheme for Vehicular Edge Computing Based on Policy Gradient", 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud), 7th IEEE International Conference on Edge Computing and Scalabe Cloud (EdgeCom), pp. 176-181, 2021.

[5] 5G Automotive Association (5GAA), "C-V2X Use Cases Methodology, Examples and Service Level Requirements Whitepaper" [Online]. Available: https://www.everythingrf.com/whitepapers/details/3617-C-V2X-Use-Cases-Methodology-Examples-and-Service-Level-Requirements

[6] M. H. Eiselt and F. Azendorf, "Accurate Measurement of Propagation Delay in a Multi-Span Optical Link", International Topical Meeting on Microwave Photonics (MWP), pp.1-3, 2019.

[7] Wikileaks, "Map of Amazon's Data Centers" [Online]. Available: https://wikileaks.org/amazon-atlas/map/

[8] Next Generation Mobile Networks (NGMN) Alliance, "5G WHITE PAPER", February 2015 [Online]. Available: https://www.ngmn.org/wp-content/uploads/NGMN_5G_White_Paper_V1_0.pdf

[9] E. Coronado, G. Cebrian-Marquez and R. Riggio, "Enabling Computation Offloading for Autonomous and Assisted Driving in 5G Networks", IEEE Global Communications Conference (GLOBECOM), nº 1-6, 2019.

[10] Amazon, "Amazon EC2" [Online]. Available: https://aws.amazon.com/es/ec2/

[11] Intel, "Intel Xeon E Processors" [Online]. Available: https://www.intel.es/content/www/es/es/products/details/processors/xeon/e.html

[12] N. A. Majedkan, A. J. Ahmed and L. M. Haji, "CPU Scheduling Techniques: A Review on Novel Approaches Strategy and Performance Assesment", Journal of Applied Science and Technology Trends, vol. 1, no. 2, pp. 48-55, 2020.

[13] A. Buttaboni, M. De Andrade and M. Tornatore, "A Multi-Threaded Dynamic Bandwidth and Wavelength Allocation Scheme With Void Filling for Long Reach WDM/TDM PONs," Journal of Lightwave Technology, vol. 31, no. 8, pp. 1149-1157, April 15, 2013.

[14] H. v. Hasselt, A. Guez and D. Silver, "Deep Reinforcement Learning with Double Q-learning", Proc. of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 2094-2100, 2016.

[15] R. S. Sutton and A. Barto, Reinforcement Learning: An Introduction (Second Edition), MA: MIT Press, 2018.

[16] M. G. Bellemare, G. Ostrovski, A. Guez, P. S. Thomas and R. Munos, "Increasing the Action Gap: New Operators for Reinforcement Learning", Proc. of the Thirtieth AAAI Conference on Artificial Intelligence, pp. 1476-1483, 2016.

[17] J. Schulman, S. Levine, P. Moritz, M. I. Jordan and P. Abbeel, "Trust Region Policy Optimization", Proc. of the 31st International Conference on Machine Learning (ICML), pp. 1889-1897, 2015.

[18] Y. Fujita, T. Kataoka, P. Nagarajan and T. Ishikawa, "ChainerRL" [Online]. Available: https://github.com/chainer/chainerrl

[19] OpenAI, "Gym" [Online]. Available: https://github.com/openai/gym