This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2021.3078501, IEEE Journal on Selected Areas in Communications

Toward Using Reinforcement Learning for Trigger Selection in Network Slice Mobility

Rami Akrem Addad[1], Diego Leonel Cadette Dutra[2], Tarik Taleb[1,4,5] and Hannu Flinck[3]
[1] Aalto University, Espoo, Finland
[2] Federal University of Rio de Janeiro, Rio de Janeiro, Brazil
[3] Nokia Bell Labs, Espoo, Finland
[4] Centre for Wireless Communications (CWC), University of Oulu, Oulu, Finland
[5]Computer and Information Security Department, Sejong University, Seoul, South Korea

***Abstract*—Recent 5G trials have demonstrated the usefulness of the Network Slicing concept that delivers customizable services to new and under-serviced industry sectors. However, user mobility's impact on the optimal resource allocation within and between slices deserves more attention. Slices and their dedicated resources should be offered where the services are to be consumed to minimize network latency and associated overheads and costs. Different mobility patterns lead to different resource re-allocation triggers, leading eventually to slice mobility when enough resources are to be migrated. The selection of the proper triggers for resource re-allocation and related slice mobility patterns is challenging due to triggers' multiplicity and overlapping nature. In this paper, we investigate the applicability of two Deep Reinforcement Learning based algorithms for allowing a fine-grained selection of mobility triggers that may instantiate slice and resource mobility actions. While the first proposed algorithm relies on a value-based learning method, the second one exploits a hybrid approach to optimize the action selection process. We present an enhanced ETSI Network Function Virtualization edge computing architecture that incorporates the studied mechanisms to implement service and slice migration. We evaluate the proposed methods' efficiency in a simulated environment and compare their performance in terms of training stability, learning time, and scalability. Finally, we identify and quantify the applicability aspects of the respective approaches.**

***Index Terms*—5G, Network Slicing, Multi-access Edge Computing, Network Softwarisation, Deep Reinforcement Learning, Agent-based Resource Orchestration.**

I. Introduction

Being re-architected from the ground up in comparison to the previous mobile network generations, 5G will unlock new business models linked to new vertical industries, e.g., automotive, e-health, public safety, and smart grids, while imposing unique requirements regarding flexibility, scalability, and availability [1]. 5G networks adopted the principles of Network Softwarization and Programmability through the use of Network Function Virtualization (NFV) and Software-Defined Networking (SDN) paradigms to cope with these stringent requirements, thus logically separating the Network Functions (NFs) from its physical infrastructure [2], [3]. Recent 5G industrial trials have validated the overall architecture by demonstrating considerable improvements in network capabilities, ultra-low latency, ultra-high bandwidth, and massive connectivity [4]–[6].

Sharing of the same underlying infrastructure among the isolated and self-contained networks as well as the availability of edge computing, such as Multi-access Edge Computing (MEC), that provides powerful service delivery with minimal delay for latency-sensitive services in the vicinity of users, has led to the concept of Network Slicing (NS) [7]–[9] that is one of the key features of 5G networks. NS is a logical network with independent control and management and provides flexibility to meet on-demand Service Level Agreements (SLA) of a specific service [10]–[12]. As a central emerging technology for next-generation networking, NS has earned attention from different Standards Development Organizations (SDOs), such as the European Telecommunications Standards Institute (ETSI) and the Third Generation Partnership Project (3GPP) [13], [14].

NSs are provisioned with dedicated resources for a certain purpose to meet the required SLAs. However, such slice specific resources are not likely to be available everywhere in the network but require careful resource allocation policies and control [15]–[17]. Therefore, there is a strong need to extend the notion of mobility that is traditionally considering user devices or services only without much concern of combined resource consumption needed for NS [18], [19]. For this purpose, the authors of [20] have introduced the notion of Network Slice Mobility (NSM) that considers not only the user mobility and service migration but combines them with the mobility of network resources of a slice. The authors argue that to ensure service continuity, all these three elements need to be taken into account when migrating a slice from a service area to another. An NSM action or pattern may be triggered by user mobility, service mobility, or a network connectivity event, e.g., network coverage issues. Thus, the authors in [20] designed and presented different NSM patterns with their corresponding grouping methods and relevant mobility triggers. NSM is implemented by a collection of methods that ensure service continuity to the end-users of a slice. These methods should also consider that after a successful NSM pattern, the slice can be scaled up and down, i.e., slice breathing, in case of sudden resource saturation or resource conflicts. NSM extends MEC through the coordinated Virtual Network Functions (VNF) live migration capability [21]. Furthermore, the authors categorized NSM patterns based on the mobility

0733-8716 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Oulu University. Downloaded on May 31,2021 at 05:44:38 UTC from IEEE Xplore. Restrictions apply.

of end-users, resource availability, service consumption, and security concerns. Hence, the classification of NSM patterns into three principal categories: i) full slice mobility; ii) partial slice mobility, which includes slice breathing, slice splitting, and slice merging; and iii) slice mobility optimizer, which contains slice shrinking pattern.

The dynamicity of these networks establishes the triggers as the main catalyst for NSM scenarios in real-life environments. Thus, the definition of six tightly coupled mobility triggers, by authors in [20], as enablers for allowing smooth NSM patterns. The six triggers defined in [20] broadly relate to users' mobility, the availability of physical and network resources, resource efficiency utilization, service reliability, and security. Nevertheless, since triggers are non-orthogonal and can overlap, the mobility action selection process becomes complex and unambiguous. Meanwhile, Artificial Intelligence (AI) techniques have been introduced to optimize the use of system resources, e.g., latency, bandwidth, RAM, processor, disk, and I/O in [22]–[25] based on given polices. The increase in the availability of affordable hardware resources [26] for data processing paves the way for extensive use of Machine Learning (ML) techniques in both Clouds and MEC [27]–[29]. Besides, standardization communities, e.g., ETSI, and 3GPP, expect that AI will be the inherent path of smart and responsive next-generation networks [30], [31]. Hence our interest in applying AI and the use of Deep Reinforcement Learning (DRL) algorithms to automate the selection of triggers responsible for service and resource migration and NSM action selection. The main contributions are as follows:

- Introduce a DRL-based agent capable of selecting the best actions and triggers to refine system resources consumption in NSM patterns;
- Design and implement an agent-based on two different DRL algorithms; the first solution relies on a value-based method, while the second solution is a hybrid approach;
- Present extensive simulations and accuracy results assessments for permitting a fine-grained trigger selection pattern within the 5G network.

The remainder of this paper is structured as follows. Section II reviews relevant work in the literature. Section III details the envisioned architecture, the system model, and the developed agent responsible for action/trigger selection in NSM patterns. In Section IV, we present the detailed design, operating principles, and parameters of our proposed agent's neural networks. Section V shows the experimental environment and evaluation results. Finally, Section VI concludes the paper and introduces future research directions. Table I summarizes the abbreviations used in the manuscript.

II. Related Work

Farahnakian et al. [32] introduced a predictive approach based on CPU load variation to enable reliable and optimized inter-data-centers live migration operations. The authors used a Linear Regression technique, a supervised ML method, to achieve their proposal. The authors use the CPU usage history in each host to approximate the short-time future CPU utilization, therefore detecting both over-utilized and under-utilized hosts. Using the current CPU load, they designed a method that forecasts the expected CPU usage for the next period. This approach allowed the authors to schedule Virtual Machines (VMs) migrations and determine optimal decisions within a data-center environment. The authors of [33] proposed a predictive anti-correlated VM placement algorithm to reduce resource consumption in cloud computing domains. The authors started by monitoring each cloud host, then predicting the appropriate VMs to be migrated, and finally placing them on the adequate target hosts. The authors used a multi-layer perceptron model, an ML technique, based on CPU consumption features to select VMs. When evaluated based on real workload traces, the proposed approach reduced energy consumption and SLA violations. Although both approaches minimize the energy cost and SLA violation rate more efficiently than previous proposed solutions and techniques, both works avert cases where the internal workload of VMs is irregular, i.e., either high or low resource variations. Thus, actions such as scaling up or down various inner resource types, e.g., CPU, RAM, and DISK, are neglected. Indeed, these approaches may cause performance degradation for 5G applications with dynamic workloads.

Ravi and Hamead [34] developed an energy-efficient solution for Green Computing. The proposed approach leverages live migration of the VMs to reduce energy consumption. The authors utilized Q-Learning, a Reinforcement Learning (RL) method, to optimize decision-making in green cloud environments [35]. Their approach delivers a cost-efficient service provisioning by refining the trigger engine's decisions, i.e., responsible for starting live migrations. They designed and implemented an agent that learns optimal policies to follow and incorporates the trigger engine changes in pre-processing data storage. The authors of [36] proposed an approach based on both live migration and ML techniques to handle virtual network relocation problems. The authors used an RL agent, ML method, to dynamically select non-critical virtual networks' resources and migrate them while satisfying Quality of Service (QoS) requirements. The obtained test results show the proposed approach's efficiency compared to static decision-making approaches or randomized ones. Duggan et al. [37] presented a procedure to dynamically select the best VMs susceptible to reduce energy consumption within a cloud data-center domain when shifted away, i.e., migrated. The authors leveraged on a single agent RL approach to autonomously refine energy efficiency. Moreover, the authors exploited over-utilized hosts for choosing VMs to reduce the number of migrations while preserving fair energy usage through the usage of Q-learning. Considering that the CPU utilization is proportional to power usage, the authors built their RL modeling entirely based on the CPUs' variations across different hosts. The CloudSim framework [38] was used for practical tests, demonstrating the feasibility of the preliminary results' approach and analysis. Because of the

Authorized licensed use limited to: Oulu University. Downloaded on May 31,2021 at 05:44:38 UTC from IEEE Xplore. Restrictions apply.

TABLE I: List of abbreviations used in the manuscript.

Abbreviation	Description	Abbreviation	Description
NFV	Network Function Virtualization	SDN	Software Defined Networking
NFs	Network Functions	MEC	Multi-access Edge Computing
NS	Network Slicing	SLA	Service Level Agreements
SDO	Standards Development Organizations	ETSI	European Telecommunications Standards Institute
3GPP	Third Generation Partnership Project	NSM	Network Slice Mobility
AI	Artificial Intelligence	ML	Machine Learning
DRL	Deep Reinforcement Learning	RL	Reinforcement Learning
VMs	Virtual Machines	QoS	Quality of Service
DNN	Deep Neural Networks	DFQL	Dynamic Fuzzy Q-learning
MEP	Mobile Edge Platform	MEC app	MEC applications
VIM	Virtualized Infrastructure Manager	NFVI	NFV Infrastructures
VNFM	Virtual Network Function Manager	MEPM-V	Mobile Edge Platform Management
EM	Element Management	NFVO	NFV Orchestrator
MEAO	Mobile Edge Application Orchestrator	SMDM	Slice Mobility Decision Maker
RM	Request Manager	LE	Learning and Exploration module
TSE	Trigger Selector and Exploitation	DAC	DRL Algorithms Comparator
OSS	Operation Support Systems	BSS	Business Support Systems
MC	Metric Collector	RAT	Resource Availability Trigger
SCT	Service Consumption Trigger	ROT	Request Overload Trigger
SLT	Service Load Triggers	DQN	Deep Q-Network
A2C	Advantage Actor-Critic	C-SMDM	Core-Slice Mobility Decision Maker
ReLU	Rectified Linear Unit	DDPG	Deep Deterministic Policy Gradient

complexity and large-scale networks of the 5G ecosystem, the application of Q-Learning algorithms may be infeasible in practice [39]. Indeed, those methods have scalability issues representing the input/output features of advanced scenarios. Besides, only VM migration decisions were taken into consideration, while scaling methods were overlooked.

Masoumzadeh and Hlavacs [40] introduced a procedure to detect overloading and underused hosts in a cloud-based environment for ensuring low energy consumption using a threshold-based algorithm. Their algorithm employed an RL technique combined with Dynamic Fuzzy logic and Deep Neural Networks (DNN), resulting in a Dynamic Fuzzy Q-learning (DFQL) method for intelligently computing hosts' upper and lower limits. They deployed their proposal using a set of minions, i.e., hosts, and an orchestrator. The orchestrator acts as a global manager, collecting input data from hosts and regarding these data as experiments gained through a learning procedure by interacting with the environment. They use the obtained model to improve the CPU utilization threshold forecast, thus allowing an efficient service migration. Notwithstanding the usage of efficient methods for handling large state/action spaces in the proposed approach, the authors omitted the usage of crucial information, such as hosts' RAM or Disk overload, that may deliver fine-grained results. The authors did not consider using scaling operations to reduce the network overhead as migration is a costly operation in highly dynamic networks such as new generation architectures and 5G systems.

Compared to the previously cited works, in this work, we develop an AI-based decision-making system able to autonomously triggering service migration/slice mobility, scaling up/down operations, and remaining static if no changes are required. The proposed approach acts without prior explicit knowledge about 5G environments and the availability of known public data-sets.

III. Envisioned Architecture & System Model

A. Envisioned Architecture

Fig. 1 depicts an overview of the envisioned architecture, incorporating an agent capable of autonomously selecting triggers and actions for allowing various NSM patterns. The envisioned architecture is built to target numerous use cases that go beyond what the current device-centric mobility approaches can support. Autonomous moving equipment, e.g., drones, robotic vehicles, or a fast-moving train carrying mobile users enjoying infotainment services, are prime examples defined in [20]. The use case related to drones defines the perfect example for predictable paths and high mobility, showcasing a new mobility pattern, i.e., full slice mobility, not perceived before in the previous generations. While the use case related to fast-moving trains illustrates the need for partial slice mobility patterns, i.e., slice splitting and slice merging, to reduce the bottleneck in high-speed streaming services.

The envisioned architecture, introduced in Fig. 1, is divided into two separate layers to efficiently enable the defined and upcoming use-cases, mainly the Orchestration and MEC

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2021.3078501, IEEE Journal on Selected Areas in Communications

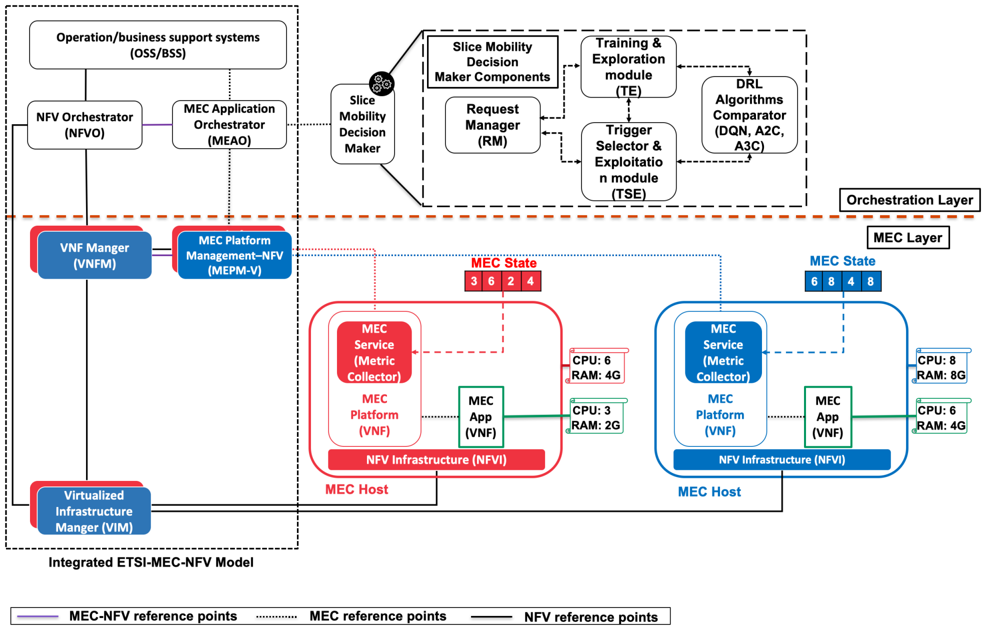
Operation/business support systems (OSS/BSS)
NFV Orchestrator (NFVO)
MEC Application Orchestrator (MEAO)
Slice Mobility Decision Maker
Slice Mobility Decision Maker Components
Request Manager (RM)
Training & Exploration module (TE)
Trigger Selector & Exploitation module (TSE)
DRL Algorithms Comparator (DQN, A2C, A3C)
Orchestration Layer
MEC Layer
VNF Manger (VNFM)
MEC Platform Management–NFV (MEPM-V)
MEC State
3 6 2 4
6 8 4 8
MEC Service (Metric Collector)
MEC Platform (VNF)
MEC App (VNF)
CPU: 6 RAM: 4G
CPU: 3 RAM: 2G
CPU: 8 RAM: 8G
CPU: 6 RAM: 4G
NFV Infrastructure (NFVI)
MEC Host
Virtualized Infrastructure Manger (VIM)
Integrated ETSI-MEC-NFV Model
MEC-NFV reference points
MEC reference points
NFV reference points

Fig. 1: Architecture for Smart Triggers Selection in Network Slice Mobility.

layers. This layering model helps manage applications by casting MEC in NFV paradigms, hence complying with ESTI's MEC and NFV standards [41]. Considering the MEC-NFV standards, both the Mobile Edge Platform (MEP) and MEC applications (MEC app) are VNFs. Therefore, elements of the NFV domain hosted in the MEC layer, i.e., the Virtualized Infrastructure Manager (VIM), NFV Infrastructures (NFVI), and VNF Manager (VNFM), manage their life-cycle. The Mobile Edge Platform Management (MEPM - V) acts as Element Management (EM) in the NFV architecture, thus providing application management features to the MEP. The NFV Orchestrator (NFVO) and the Mobile Edge Application Orchestrator (MEAO), in the Orchestration layer, share service application information and the network service information in the MEC-NFV domain to provide a reliable orchestration system. It is worth noticing that we omitted the reference points details between MEC and NFV components for clarity.

The Slice Mobility Decision Maker (SMDM) agent is an additional plugin to MEAO [42]. The main components of SMDM are the Request Manager (RM), the Learning and Exploration (LE) module, the Trigger Selector and Exploitation (TSE) module, and the DRL Algorithms Comparator (DAC) module. The SMDM agent interacts with the MEC layer, i.e., environment, through the RM module, retrieves states, selects decisions such as scaling up/down diverse types of resources, e.g., RAMs, CPUs, and disks, or migrating MEC apps, and receives rewards for its decisions. Sections III-C and III-D further explain the states, actions, and rewards properties. In accordance with ETSI-MEC-NFV directives, the SMDM agent communicates with the Operation/Business Support Systems (OSS/BSS) for executing administrative and billing instructions. It also leverages the NFVO to command all migrations and scaling operations between MEC hosts/nodes. In our proposed architecture, the information between the SMDM agent and the environment, i.e., MEC hosts, transits via the standardized interfaces of MEAO and MEPM elements.

To ensure a reliable system, we designed both a testing environment within the LE module and a real-world framework used for production through the TSE module. The testing environment simulates the impact of actions on the real production environment, implementing shared features to represent both states and actions. We deployed our testing environment following the principles and the conventions of OpenAI Gym [43]. In our setup, a set of simulated MEC hosts is created, then for each MEC host, a random number of simulated MEC apps is instantiated. Meanwhile, the production environment follows a master/slave architecture and depends on real working implementations using hypervisors and containers, i.e., LXC [44], for virtualization and ONOS as an SDN networking component for connectivity purposes [45]. The slaves, MEC hosts in the MEC layer, collect the local data using a MEC service dubbed Metric Collector (MC) in the MEP. Each MC service sends the collected data to its respective MEPM-V, which in turn shares its content with the MEAO. Finally, the master through the SMDM agent retrieves and processes the received data from the MEAO, i.e., Fig. 1. Alternatively, as long as the SMDM agent is not adept, the RM module will be forwarding the received requests to the testing environment in the LE module. With this, we can avoid network disasters and users' dissatisfaction resulting from sub-optimal action selection during the learning process, i.e., this part will be further elaborated in section IV. In the following subsection, we will provide detailed information on

0733-8716 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Oulu University. Downloaded on May 31,2021 at 05:44:38 UTC from IEEE Xplore. Restrictions apply.

the collected data and features.

B. Environment Description and Type of Collected Data

We ensure that the type of data and features collected from the real and simulated environment are comparable, thus allowing us to use the testing environment results for the production environment. On the other hand, based on [20], we know that the triggers are non-orthogonal and overlapping, thus complicating the decision process; however, related triggers can be grouped. Resource Availability Trigger (RAT), Service Consumption Trigger (SCT), and Request Overload Trigger (ROT) introduced in the aforementioned work can be used as components of an aggregated trigger dubbed "Service Load Triggers (SLT)", as they are all related to system resources. By refining and combining the triggers, the proposed agent, i.e., SMDM, can form optimized policies regarding trigger selection for NSM. Therefore, as SLT is a set or a combination of tightly related triggers, i.e., RAT, SCT, and ROT, follow data needs to be collected from the underlying MEC system to the SLT trigger based on its sub-components:

1) Resource Availability Trigger (RAT): The RAT trigger deals with the aggregate system-level resources related to the under-laying nodes, i.e., MEC hosts, hosting virtualization instances, i.e., container-based MEC apps in our case. Thus, each node, i.e., MEC host, is sending information of CPU, Memory, and Disk contained locally to the SMDM agent for processing. The details provided by the RAT trigger are the CPU, RAM, and DISK capacities of the MEC host as well as their current consumption. Upon receiving the information from all the MEC hosts constituting the environment, the SMDM agent in the orchestration layer computes the percentage of the used resources in each MEC host. Moreover, all computation operations are done within the orchestration layer to free the MEC hosts of any additional overhead.

2) Service Consumption Trigger (SCT) & Request Overload Trigger (ROT): Compared to the RAT trigger, the SCT and ROT triggers cope with the performance of a single service, i.e., internal resource consumptions of the MEC app. The main idea behind developing these triggers is to monitor the services themselves instead of watching only the MEC hosts, allowing bigger flexibility and exploring a more comprehensive range of new actions such as scale-up/down operations. The details of the triggers, i.e., SCT and ROT, are the CPU and RAM of each container-based MEC app and their current consumption. Moreover, we can expand these details to cover different parameters such as the number of requests/MEC app, while the disk details are missing because container-based virtualization uses the notion of file-system instead of volumes. Identically as the previous trigger, each MEC host sends those local data to the orchestration layer for computation to reduce as much as possible computation overhead.

Based on the updates and preliminary exploitable models conceived leveraging the LE module, the TSE module will enable fine-grained trigger selection depending on the encountered situation for NSM. To corroborate the obtained model, both the LE and TSE modules utilize the DAC module instructed to apply and select formal DRL algorithms. This module can select which DRL algorithm is suitable for accurate decisions and trigger selection through a verification and comparison method based on the generations of scenarios. Furthermore, in sections IV-A and V, we detail and analyze the proposed verification method of the SMDM agent.

C. Primer on Reinforcement Learning

In this section, we present a brief introduction to the RL techniques due to their importance for the rest of the work. Unlike supervised and unsupervised ML algorithms, RL techniques are independent of prior data [46]. RL algorithms or RL-based agents learn to perform complex tasks and effective decisions through interaction with the environment based on trial and error processes. We define the interaction in terms of specific states/observations, actions, and rewards. Precisely, an RL agent interacts periodically with an environment "E", observes the current state s_t, then executes an action a_t. Subsequently, the agent will observe a new state s_{t+1} and receives a corresponding reward r_{t+1} [47]. This process keeps repeating while adjusting the policy $\pi(s_t, a_t)$ until the convergence phase, i.e., optimal policy [48]. A policy π has for objective to map states to actions, i.e., $\pi : S \rightarrow A$, by maximizing the discounted reward over the discrete-time steps. The cumulative discounted reward G_t at each given time t is defined by:

$$G_t = \sum_{m=0}^{\infty} \gamma^m r_{t+m+1} \tag{1}$$

Where γ is the discount factor defined between [0-1].

Considering a concrete mapping of the RL components with our envisioned architecture introduced in Fig. 1:

1) The agent is the smart constituent of the architecture. It implements various modules to handle communication and decision making. Besides, the agent manages both training and exploitation/production phases, i.e., LE and TSE modules, by perceiving states and learning to perform optimal actions through time. The conformal mapping of the agent, in this case, is the SMDM agent.
2) The environment provides the target for the optimization problem. It also delivers new observations based on the agent's actions. The MEC layer, with its set of MEC hosts, Fig. 1, is the environment in the defined problem.
3) The states, actions, and rewards are the main characteristics allowing the agent to solve complex tasks without having dedicated programs. Those elements will be presented in detail in the following subsection, i.e., III-D.

D. System Model

Based on the elements introduced in III-C, we need to select and define various components such as the state space, action space/generation, and the adequate reward function to establish a proper RL system.

1) State Space: Following the reception of information from all MEC hosts and their respective MEC apps, i.e., container-based MEC app [49], from the MEAO, the SMDM agent aggregates them to form the input state to be fed into a function approximator [50], [51], i.e., DNN in our case.

Our environment is a widely distributed system made up of a set of N MEC hosts. Each MEC host $n \in N$ is hosting a given number of MEC apps, i.e., containers, $c \in \mathcal{C}$. We define the following variable to introduce the MEC app to MEC host mapping:

$\forall c \in \mathcal{C}, \forall n \in N :$

$$\mathcal{X}_{c,n} = \begin{cases} 1 & \text{if a container-based MEC app } c \text{ is running on} \\ & \text{top of the MEC host } n. \\ 0 & \text{Otherwise} \end{cases}$$

Our representation of the state space consists of taking into account the system resources of each MEC host. Each MEC host consists of an ordered list, whereby each element in that list describes the percentage of used resources on that node, such as CPU, Memory, and Disk for the RAT trigger while considering Memory and CPU percentages usage of the container-based MEC apps to cover SCT and ROT triggers.

Using the percentage-based representation, we get a hundred levels of variations for each selected feature. Therefore, we represent each MEC host by $100^{\mathcal{F}_n}$, with $\mathcal{F}_n$ being the number of selected features. We derive the number of features as follows:

$$\forall n \in N, \mathcal{F}_n = f_n + \sum_{\forall c \in \mathcal{C}} f_c \times \mathcal{X}_{c,n} \tag{2}$$

Where f_n and f_c denote the number of features in MEC hosts and their container-based MEC apps, respectively.

By aggregating MEC host entries, the number of states obtained will be equal to:

$$\mathcal{S} = \prod_{\forall n \in N} 100^{\mathcal{F}_n} \tag{3}$$

In Fig. 2, we introduce a detailed state-space formation and representation along with their relationship to triggers. We omitted both the integrated ETSI-MEC-NFV model and the SMDM agent presented in Fig. 1 to concentrate on states' formation. Our example consists of two MEC hosts, i.e., MEC Host 1 in red and MEC Host 2 in blue, each one of them is hosting a container-based MEC app [49], i.e., MEC app (VNF) in green, delivering services to end-users. We assume that f_n and f_c features are CPU and RAM; thus, both f_n and f_c are equal to two. Knowing that each MEC host has one MEC app, we deduct that $\mathcal{F}_n$ for each MEC host is equal to four, producing an ordered list of length four to represent the SLT trigger.

Our state handling passes through four distinct steps to implement equation 3. The first two steps, i.e., steps 1 and 2, are executed in the MEC layer, i.e., within MEC hosts. The remaining steps, i.e., steps 3 and 4, are done at the orchestration layer level. In step 1, each MC in a MEC host collects the data related to the MEC host and the MEC app. Step 2 splits and organizes the collected data into two separate ordered lists following the RAT and SCT/ROT definitions. It also shares the data with the SMDM agent, i.e., explained in detail in III-B. Being executed at the orchestration level, step 3 computes the percentage of the used resources in each MEC host then forms an ordered list based on the obtained $\mathcal{F}_n$, i.e., four in this example. Finally, in the last step, i.e., step 4, our SMDM agent constitutes the state to be used in the DAC module by aggregating each SLT component together. Note that the RAT components are always at first positions as each SLT length, i.e., 4 in this case, may vary due to the increasing/decreasing number of container-based MEC apps or migration operations. By doing so, we guarantee that our representation of state-space will not be dependent on variations of MEC apps.

2) Action space: The state-space defined above allows us to obtain a state at each time-step. We need to define an action space to be able to transit from one state to another. In our model, the action space is represented by:

- no-action, i.e., conserves the current resources distribution.
- migrate from a given source MEC host to a given target MEC host.
- scale up/down various resource types such as CPU and RAM.

The SMDM agent explores various potential states and their respective rewards using these actions. The number of available actions within the proposed model can be expressed as follows:

$$\mathcal{A} = \sum_{\forall n \in N} \sum_{\forall c \in \mathcal{C}} \mathcal{X}_{c,n} \times (N + \Phi(c)\mathcal{R}_c) \tag{4}$$

In which $\Phi(c)$ is a function that returns the number of authorized operations for a container-based MEC app $c \in \mathcal{C}$ except for migration actions. Migration actions and no-actions are equal to the number of available MEC hosts "N", i.e., when the destination MEC host is equal to the source MEC host, it is a no-action; the remaining are all migration actions. For instance, $\Phi(c)$ may return the value "two" to express scale up and scale down operations. $\mathcal{R}_c$, a scalar value, denotes the number of resource types used for each authorized operation returned by the function $\Phi(c)$. For instance, scaling up operations may be applied to CPU and RAM; thus, $\mathcal{R}_c$ will be equal to two.

3) Reward Function: The reward function combines information related to system resources and operation time. We start by measuring the operation time, i.e., time for completing an action, e.g., migration or scaling up/down RAM, from the agent. In the simulation environment, the times for scaling-up and scaling-down operations are static. The scaling-down operation is slower as this action in practical implementation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2021.3078501, IEEE Journal on Selected Areas in Communications

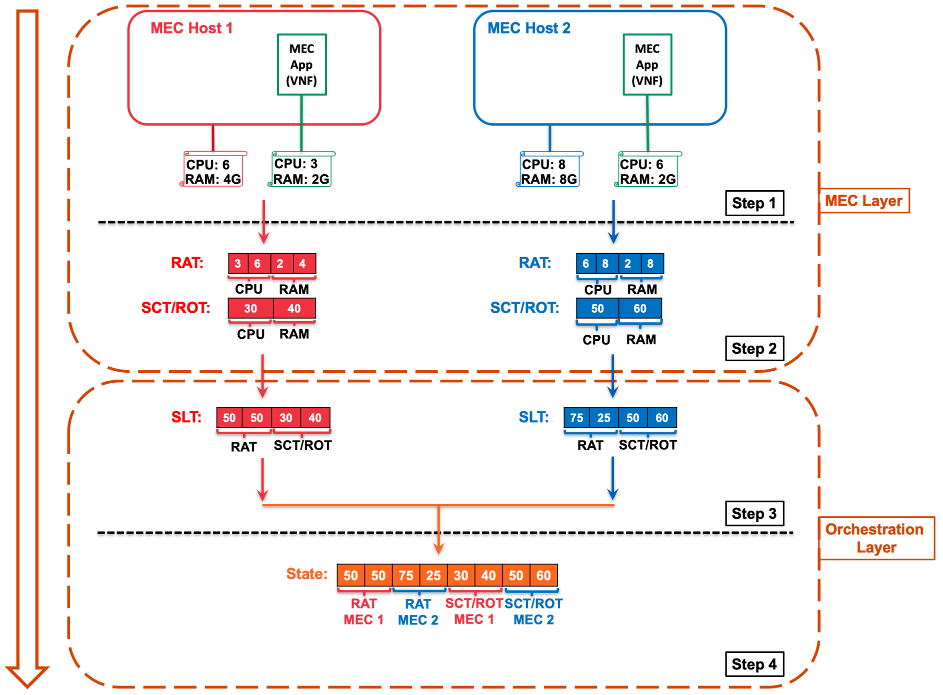

Fig. 2: States space representation and formation using the proposed modeling.

requires waiting time before execution. The migration time in the simulation environment depends on the current disk size. These values can be directly measured in a production environment. We then reverse it to obtain a progressive reward. The reward is inversely proportional to the operation time; the longer the operation time is, the lower the reward will be. Regarding the system resources, we know that when the percentage of exploitation increases, the performances decrease. Thus, we follow the same logic as the operation time, and we invert the sum of all system resources, e.g., CPU and RAM of MEC hosts. With this, we come to the following reward function $\mathcal{R}$:

$$\mathcal{R} = 1/\mathcal{T} + \sum_{\forall n \in N} \sum_{\forall j \in \mathcal{K}} 1/g(n,j) + \sum_{\forall c \in \mathcal{C}} \sum_{\forall j \in \mathcal{K}} 1/g(c,j) \quad (5)$$

Where $\mathcal{T}$ represents the operation time and $g(x,j)$ is a function that returns the percentage of a given resource j in the set of resources $\mathcal{K}$ belonging to MEC hosts or MEC apps, i.e., $x \in N$ or $x \in \mathcal{C}$.

Finally, we added coefficients to time and resource usage to make one parameter more influential than the other. In our case, we consider the time as the main parameter.

$$\mathcal{R} = \alpha * (1/\mathcal{T}) + \beta * (\sum_{\forall n \in N} \sum_{\forall j \in \mathcal{K}} 1/g(n,j) + \sum_{\forall c \in \mathcal{C}} \sum_{\forall j \in \mathcal{K}} 1/g(c,j)) \quad (6)$$

IV. Design of the Slice Mobility Decision Maker Agent

A. Operational Mechanisms

The following section presents the internal operational mechanisms of the SMDM agent and the design of its adopted RL algorithms.

Equation 3 demonstrates that the large number of states generated by our problem formulation makes it intractable, as noted in Section III-A. Moreover, Mnih et al. [39] have shown that RL methods struggle to find an optimal policy in a reasonable time when state space is considerably large. Hence, in the SMDM agent, the DAC module integrates and uses a DNN for approximating the state-space in the case of value-based approaches. The combination of DNNs and RL principles rendered scalable ML algorithms capable of handling large state environments dubbed as DRL algorithms. Precisely, our developed DAC module employs Deep Q-Network (DQN), a DRL value-based technique, to obtain an optimal policy regarding trigger selection for NS mobility patterns [52].

However, equation 4 shows that the number of actions is growing linearly with the number of MEC hosts and MEC apps, thus producing a large action space in large-scale networks [53]. Consequently, the hybrid DRL method, i.e., a combination of both value-based and policy-based approaches, called Advantage Actor-Critic (A2C), is adopted by our DAC module [54]. In the A2C-based approach, we took advantage of the DRL value-based method to measure the action's quality while optimizing the agent's behavior, leveraging a DRL policy-based method [55], i.e., determining the policy function π without worrying about a value function. Albeit

0733-8716 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Oulu University. Downloaded on May 31,2021 at 05:44:38 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2021.3078501, IEEE Journal on Selected Areas in Communications

hybrid methods can solve problems that value-based methods cannot, they usually converge on a local maximum rather than on the global optimum [56]. Therefore, we introduce a verification function that makes a selection between DQN and A2C in the DAC module. The main reason for comparing and introducing value-based, i.e., DQN, and hybrid, i.e., A2C, methods is to decide which type of algorithm family, i.e., value-based or hybrid, is suitable for trigger selection in NSM as both types of algorithms present two distinctive and divergent objectives. Once setting a clear winner, we can extend the work to compare and benchmark other algorithms within the winning category.

Before describing both the DQN and the A2C algorithms and their hyper-parameters, we provide the pseudo-code detailing the SMDM agent's functionalities in **Algorithm 1**. The proposed agent implements three main principles:

- A request-based control interface: The implemented algorithm depends on requests to be either in the training phase or in the exploitation phase of the models;
- The training phase: Begins with neural network initialization then allows our agent to learn triggers and actions for the optimal policy through time and requests;
- The exploitation phase: The agent selects actions and triggers for the optimal policy to be deployed.

Algorithm 1, called Core-SMDM (C-SMDM), serves to describe in detail the aforementioned principles. Initially, C-SMDM is reactive to requests, which means that once a request is received, the C-SMDM algorithm is executed. Each request contains the "input state, i.e., $\mathcal{S}$" and the "request number, i.e., req_n". While the first parameter is fed into the DQN and A2C algorithms or one of them depending on the situation, the second parameter is used to track request numbers and for initialization purposes. Regarding the "req_n", the first-ever request marks the beginning of the training phase through the initialization of three variables "iteration", "drl_type", and "$trained$" in lines 1 to 5 respectively in C-SMDM, i.e., algorithm 1.

As long as the variable "$trained$" is equal to "$False$", the input features, i.e., $\mathcal{S}$, req_n, will be fed to the training phase directly, i.e., C-SMDM line 6. Each state "$\mathcal{S}$" is routed to the LE module through the RM module, i.e., C-SMDM line 7. Then, the LE module will input the state "$\mathcal{S}$" for both DQN and A2C algorithms in the DAC module; this will ensure the training of both algorithms. Note that the initialization step of the two algorithms and their neural networks by the DAC module is omitted from the C-SMDM for the sake of clarity. After that, in line 10 of C-SMDM, we increment the variable "iteration" by one for each request. If the number of iteration/request is bigger than "M", the DAC module generates a set of scenarios and compare both the DQN and the A2C algorithms. This step has for objective to determine the best algorithm choice for selecting the right triggers to the NSM patterns by updating the variable "drl_type" with the selected algorithm, i.e., lines 12 and 13. A complete evaluation and validation process will be presented in section V to emphasize the criticality of the verification and comparison step used in the DAC module. It should be emphasized that "M" is the number of iterations used during the training phase and fixed to 10000 in the following section. Besides, we set the variable "$trained$" to "$True$" for allowing our SMDM agent to switch to the production phase, i.e., the exploitation phase, starting from the next request, see C-SMDM line 14.

Algorithm 1: Core-Slice Mobility Decision Maker (C-SMDM)

```
Input  :
         $\mathcal{S}$: Input states/features.
         $req\_n$: request number.
Output:  decision: action to carry out (migrate, scale
         up/down CPU/RAM/Disk).
1  if $req\_n$ == 1 then
2      iteration ← 0;
3      drl_type ← $None$;
4      $trained \leftarrow False$;
5  end
6  if $trained == False$ then
7      RM.route(LE);
8      LE.input($\mathcal{S}$, DQN, A2C);
9      DAC.train();
10     iteration ← iteration + 1;
11     if iteration $\geq$ M then
12         DAC.generate_scenarios();
13         drl_type ← DAC.compare(DQN, A2C);
14         $trained \leftarrow True$;
15     end
16 end
17 else
18     RM.route(TSE);
19     TSE.input($\mathcal{S}$, drl_type);
20     decision ← DAC.deliver();
21 end
```

Once the variable "$trained$" is equal to "$True$", the SMDM agent will use the RM module to route the requests to the TSE module with the state "$\mathcal{S}$" and the "drl_type" as parameters. Finally, the TSE module contacts the DAC module by requesting only the winning algorithm. This last step will deliver accurate decisions regarding trigger selection in the context of NSM, i.e., lines 17 and 21 in C-SMDM. Furthermore, we precede each function/method with the module's name that executes it to improve the understanding of the core features of the proposed SMDM agent.

The subsequent descriptions detail the functioning of DQN and A2C algorithms used by the DAC module during both the training and the exploitation phases.

1) Deep Q-Networks: The proposed DQN uses experience replay to break the correlation between subsequent time-steps and allowing a stable learning curve [57]. In each batch size, DQN calculates the Temporal Difference (TD) error by taking

0733-8716 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Oulu University. Downloaded on May 31,2021 at 05:44:38 UTC from IEEE Xplore. Restrictions apply.

the difference between Q-targets (maximum possible value from the next states s_{t+1}) and the predicted Q-values [58]. Thus, allowing the agent to reduce/minimize the training errors while updating various network weights and learning an optimal policy [59].

2) Advantage Actor-Critic: To cope with the previously mentioned constraint related to the growth of action space, we propose using a continuous/pseudo-continuous action space aware algorithm, namely A2C. The A2C has two main networks, mainly the Actor and the Critic networks. Using the current weights of the network, the Actor observes the environment "E", then selects a given action by outputting a probability distribution across the action space. After that, the Critic evaluates the quality of the selected action regarding both the current state s_t and the next state s_{t+1}. Finally, at each time-step, the A2C algorithm computes the loss of Critic and Actor and updates network parameters [54].

B. Design of Neural Networks

We build our DNN with the following specifications to realize the proposed states to actions mapping:

1) DQN hyper-parameters: We use two neural networks, mainly the Q-network and the target Q-network, to predict the current Q-values and the next Q-values, respectively. We adopt the "ϵ-Greedy" policy to allow fair exploration/exploitation repartition. For both Q-Networks, we adopted Adam optimizer for adjusting the parameters of the network [60]. The learning rate is identical for both Q-Networks, i.e., $2 \cdot 10^{-5}$, while the discount factor γ is 0.99. We update the main Q-Network after every 32 iterations, i.e., the batch size is 32, while we alter target Q-Network weights every four episodes. We use two fully-connected hidden layers, each of which has 64 Rectified Linear Unit (ReLU) activation function [61]. The ReLU activation function, denoted by equation 7, is linear for all positive values and zero for all negative values. Therefore, it offers computational simplicity and better convergence features compared to other activation functions. The input and output layers depend on the number of used features and their generated actions, i.e., both introduced in equations 3 and 4. Since we are using the DQN algorithm, the selection of actions in the output layer depends on finding the maximal action value.

$$ReLU(z) = \begin{cases} z & \text{if z} > 0 \\ 0 & \text{if z} \leq 0 \end{cases} \tag{7}$$

2) A2C hyper-parameters: A2C uses two neural networks: one network is used by the Actor and the Critic to determine the probability to select a given action and its evaluation, while the second network is used only by the Critic to evaluate the next action. For both networks, we adopt the Adam optimizer for adjusting the networks' parameters while we use the value $2 \cdot 10^{-5}$ as a learning rate, with a discount factor γ of 0.98. We use a similar representation of hidden layers for both networks; mainly, we use two fully-connected hidden layers in which the number of units, i.e., activation functions, is 64 and 256, respectively. For the hidden layers, we adopt the ReLU activation functions, introduced in equation 7. For the output layer, given the fact that the Actor is building an action classifier, we use Softmax as an activation function [62]; see equation 8. In this case, the output layer has a length equal to the number of actions $\mathcal{A}$ defined in equation 4. The highest value of $\sigma(a_i)$ is selected from the set of action values. It is worth noticing that Critic networks have a unique output used to evaluate actions at various states.

$$\sigma(a_i) = \frac{exp(a_i)}{\sum_{j=1}^{\mathcal{A}} exp(a_j)} \tag{8}$$

V. Experimental evaluation

We developed a simulator using Python to evaluate our proposed agent for trigger selection in NSM patterns. We use Pytorch [63] to define our DNN models, weights computations, i.e., forward/back propagations, and optimization operations. The proposed approach is studied in active learning, where the SMDM agent directly interacts with the environment without prior explicit knowledge and the availability of known public or private data-sets. Thus, in the evaluation, data is generated by the simulation setup that is composed of discrete events. Each event represents an iteration in the learning process where the SMDM agent observes the current states, executes actions on the environment, and receives a reward depending on the newly observed state. We generate MEC hosts and MEC apps, along with their system requirements, i.e., CPU, RAM, and DISK, through a random process. The requirements of each MEC app and MEC host in terms of different resources follow a discrete uniform distribution over a dynamic interval indicated in the implementation files. Table II details the environmental simulation specifications used in the training process. In the following evaluations, we compare these two DRL-methods and analyze their performance to validate our model. It shall be noted that in our current experiments, we consider an SMDM variant that only uses DQN and another SMDM variant that only uses A2C to properly and separately evaluate each algorithm.

TABLE II: Environmental simulation specifications.

Parameter	Value
MEC hosts	3 – 10
MEC applications	3 – 100
MEC hosts CPU	64 – 128 (cores)
MEC applications CPU	1 – 4 (cores)
MEC hosts RAM	64G – 128G
MEC applications RAM	1G – 4G
MEC hosts DISK	128G – 512G
MEC applications DISK	512M – 4G

We start with the evaluation of the SMDM agent using the DQN based algorithm. In this experiment, 10000 episodes are run, changing the resources randomly for MEC host and MEC apps in the underlying layer. Fig. 3 presents the SMDM

Authorized licensed use limited to: Oulu University. Downloaded on May 31,2021 at 05:44:38 UTC from IEEE Xplore. Restrictions apply.

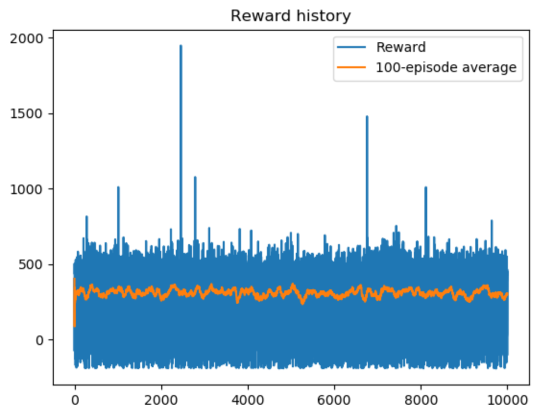

Fig. 3: SMDM agent based on Deep-Q Learning algorithm.

DQN-based agent training leveraging on a non-linear Q-value approximation based on DNN to express the state space. The SMDM agent also uses the replay memory to store training steps and reduce correlation. In Fig. 3, the Y-axis represents the rewards, while the X-axis portrays the number of episodes in the training process. We also plot the 100-episodes average in the same figure, i.e., orange color. Even though the agent can learn a useful policy behavior, we still observe a rapid change in the rewards' values during the training of our simulated environment.

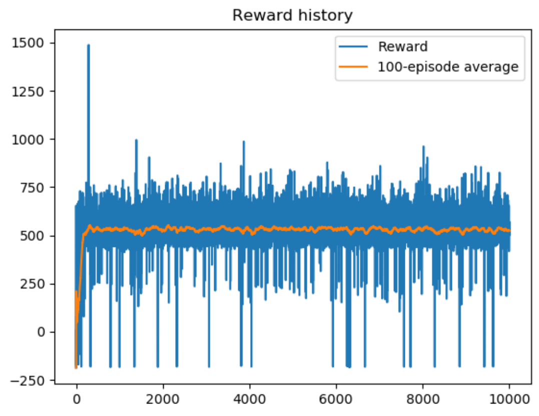

Fig. 4: SMDM agent based on Advantage-Actor-Critic algorithm.

Next, we evaluate the SMDM agent based on an A2C algorithm approach in our second experimental scenario. Fig. 4 illustrates the episodic cumulative reward as well as the same metric averaged over 100 episode iterations, i.e., orange color. We draw the graph by considering applying our DRL agent using Actor-Critic, i.e., A2C, based on TD(0) principle, i.e., no need for waiting until the end of an episode to perform updates, updating TD(0) in every time-step. We kept the same representations as before for both axis, i.e., Y-axis and X-axis, and we conserved the same number of training episodes, i.e., 10000. As an initial reflection, the agent-based on A2C can achieve a higher cumulative reward while delivering a stable learning curve compared to the previous approach based on DQN.

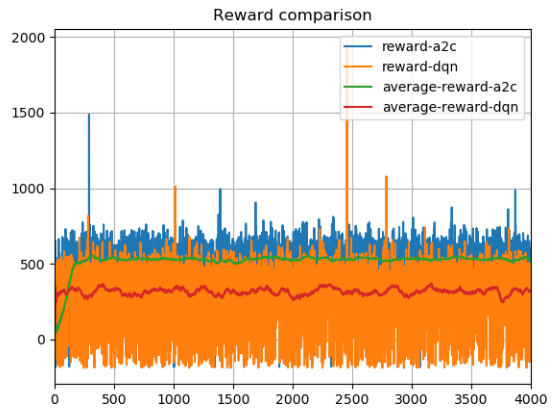

Fig. 5: Rewards comparison.

To have an accurate comparison of algorithms, we decided to plot the respective rewards/average-rewards of both algorithms together. Fig. 5 features the comparison between the SMDM agent based on the DQN network and the one based on the A2C network. The SMDM-DQN agent is represented with the orange and the red colors for the rewards and the average rewards, respectively. The rewards and the average rewards of the SMDM-A2C agent are illustrated using the blue and green colors. As Fig. 5 is the union of both previous experiments, we kept the same axis representations. It is worth noticing that we reduce the X-axis output to 4000 episodes to show the main differences in their respective behavior. The variations of the graphs after these 4000 episodes were almost identical for both algorithms. Results in Fig. 5 shows the efficiency of the A2C-based agent compared to the DQN-based agent in terms of average/cumulative rewards and learning stability.

Our previous experimental results showed that our proposed approach based on the A2C agent achieves better performances than the DQN-based solution. We decided to extend the comparison further to determine and set a clear winner. To this end, we compare the accuracy of DQN and A2C based agents, in Fig. 6, for different MEC hosts, MEC app deployments, and configurations, i.e., 3 MEC hosts and 30 MEC apps, 10 MEC hosts and 10 MEC apps, 10 MEC hosts and 30 MEC apps. Due to the absence of the term accuracy in the RL context, our DAC module, in its verification method, starts by generating customized scenarios, i.e., lines 12 and 13 in C-SMDM. Then, it evaluates and compares both algorithms, i.e., DQN and A2C, while trying to ensure "L" consecutive

Authorized licensed use limited to: Oulu University. Downloaded on May 31,2021 at 05:44:38 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2021.3078501, IEEE Journal on Selected Areas in Communications

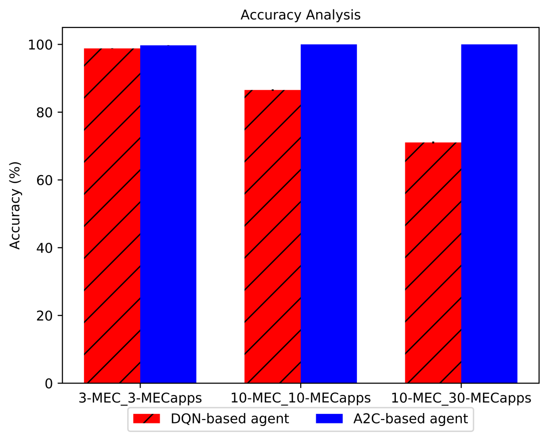

Fig. 6: Accuracy comparison.

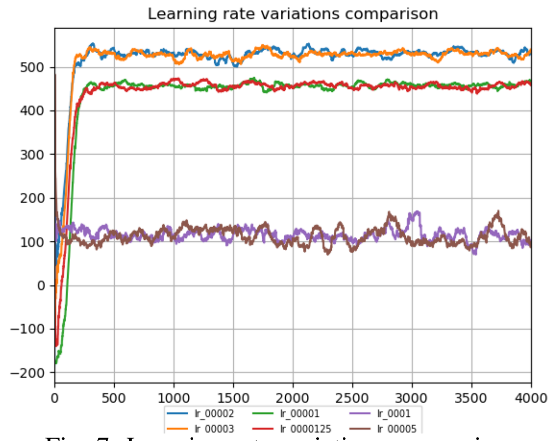

Fig. 7: Learning rate variations comparison.

actions without failing, i.e., over-utilized or under-used MEC hosts/MEC app, for “B” times consecutively. To carry out the experiment, we set “L”, i.e., scaling up/down and migration operations, equal to “30” and “B” equal to “10”. In Fig. 6, we show the agent using the DQN algorithm with a slashed red bar and use the standard blue bar to the agent exploiting the A2C algorithm. The X-axis represents the types of DRL algorithms used and the employed characteristics, while the Y-axis represents the percentage accuracy. For each agent, we also plotted the 95% confidence interval of the mean. We find a mean accuracy of 98.8%, with a 95% confidence interval of 0.07% for the 3MEC-30MECapps configuration concerning the DQN agent case. For the A2C based agent, we obtained a mean accuracy of 99.7% with 0.03% as a 95% confidence interval. Regarding the configuration 10MEC-10MECapps, we achieved an accuracy of 86.53% and 100%, respectively, for both DQN and A2C agents. Finally, on the subject of the 10MEC-30MECapps scenario, the mean accuracy is 71.05% for the DQN-based agent with a 95% confidence interval of 0.28%, while these values are 100% and 0%, respectively, for the agent leveraging A2C.

After selecting the A2C-based algorithm for the SMDM agent, we explore and evaluate additional hyper-parameters susceptible to improve the agent’s policy. Fig. 7 portrays the effect of different learning rates, i.e., hyper-parameter, on the convergence performance of the proposed agent to select adequate triggers within NSM patterns scope. In Fig. 7, the Y-axis represents the average reward values giving different types of learning rates while the X-axis conserved the same representation of the graph in Fig. 5. We study the impact of six different learning rates values in Adam optimizer. We notice the establishment of three distinct categories/classes. By convention, we label the first class as “global-optimal” and it includes learning rates $2 \cdot 10^{-5}$ and $3 \cdot 10^{-5}$ in blue and orange colors, respectively. The second category is the “near-optimal” one; it contains learning rates $1 \cdot 10^{-5}$ in green and $1.25 \cdot 10^{-5}$ in red. Finally, the last class is composed of learning rates $1 \cdot 10^{-4}$ and $5 \cdot 10^{-5}$, i.e., in purple and brown colors, respectively, and dubbed “local-optimum”. We derive that either too small or too large learning rate values may cause local convergence to the designed algorithm resulting in inefficient learning. It is worth noticing that although this graph is presented after comparison tests and graphs, i.e., in Fig. 3, Fig. 4, and Fig. 5, we set the value of the learning rate to $2 \cdot 10^{-5}$ based on this experimentation to ensure suitable hyper-parameter training features.

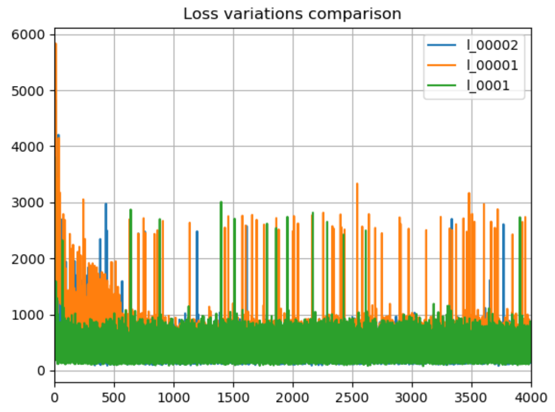

Fig. 8: Training Loss variations comparison.

To further explore the effect of learning rates in convergence characteristics, we plot the training loss of the A2C network, i.e., DNN. In Fig. 8, the blue curve denotes the loss variation when the learning rate is equal to $2 \cdot 10^{-5}$. The loss changes shown in orange color represent the learning rate of $1 \cdot 10^{-5}$, while the loss of $1 \cdot 10^{-4}$ is illustrated using the green color. In other words, we selected one curve from each class defined in Fig. 7 to ensure a fair comparison. The Y-axis portrays the variation of the non-normalized computation rate while the X-axis is similar to previous tests, i.e., Fig. 5 and Fig. 7. Based

0733-8716 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Oulu University. Downloaded on May 31,2021 at 05:44:38 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2021.3078501, IEEE Journal on Selected Areas in Communications

on the obtained results in Fig. 8, we contest that the loss when the learning rate is equal to $2 \cdot 10^{-5}$, i.e., blue curve, gradually decreases and stabilizes at a small value. Besides, the obtained loss is smaller than the remaining ones as it was shadowed from the final training episodes. However, as expected, the loss of the big learning rate, i.e., $1 \cdot 10^{-4}$, was oscillating and not decreasing to learn an optimal behavior. It is noticed that the loss changes of the learning rate equal to $1 \cdot 10^{-5}$ were acceptable with some variations and biases. This enhanced experiment confirmed that the selection of a suitable learning rate helps accelerate the training process, i.e., the blue curve is faster than the orange curve and counter the local optimum optimization trap [56].

A. Discussion

We can observe in Fig. 3, Fig. 4, and Fig. 5 that solution using A2C achieves better cumulative reward while remaining stable during the training phase. Nevertheless, the DQN approach is unstable, with a reduced average reward value, as Fig. 3 displays. Furthermore, the results in Fig. 6 show that when we increase both the number of MEC hosts and MEC apps, it reduces the accuracy of DQN. Meanwhile, A2C results indicate that this ML technique has a lower sensibility to the number of MEC hosts and MEC apps. Thus, the A2C agent is learning a useful policy while satisfying users' QoE and preventing SLA violations.

Our results in Fig. 7 and Fig. 8 showed the effect of hyper-parameters tuning and selection to the performance of an RL algorithm in particular and an ML approach in general. Besides, the learning rate selected for the evaluation directly relates to the convergence of the defined reward function, i.e., in equation 6. Large and small learning rates represented by the class called "local optimum" may either miss the global optimum or drastically slow the learning speed.

Based on those observations and the obtained results, we can assert that the A2C algorithm outperforms the DQN approach when adapted to select efficient triggers for NSM patterns.

VI. Conclusion and Future Work

In this work, we designed, modeled, and evaluated two DRL-based algorithms that allow fine-grained selection of system based triggers regarding the NSM patterns. The work constituted an effort toward making the triggers defined in [20] intelligent while saving their original definition and implementations. The proposed approaches have been fully implemented in the MOSA!C Lab research group [64] and are available as an open-source in Github [65]. We also validated the proposed methods by implementing a simulated environment and testbed. Our numerical results, established by the simulated environment, showed the efficiency of the A2C based approach compared to the DQN solution in terms of training stability, learning time, and scalability.

Nonetheless, our A2C-based agent presented limitations caused by the increase in the size of the action space. Thus, for the authors, a future research direction would be to consider the DRL based on the A2C algorithm approach as a baseline solution and investigate extending the model to cover and benchmark the Deep Deterministic Policy Gradient (DDPG) based on Wolpertinger architecture [66], Trust Region Policy Optimization (TRPO) [67], and Proximal Policy Optimization (PPO) [68]. Besides, we plan to extend the deployment and evaluation to real production environments and cover the remaining triggers related to users' mobility defined in [20].

Acknowledgment

This work is partially supported by the European Union's Horizon 2020 ICT Cloud Computing program under the ACCORDION project with grant agreement No. 871793. The work is also funded by the Academy of Finland Project CSN under Grant Agreement 311654 and the 6Genesis project under grant No. 318927, respectively.

References

[1] 5G PPP, "White Paper: Validating 5G Technology Performance Assessing 5G architecture and Application Scenarios," Tech. Rep., June 2019.

[2] E. Datsika *et al.*, "Software Defined Network Service Chaining for OTT Service Providers in 5G Networks," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 124–131, Nov. 2017.

[3] R. A. Addad *et al.*, "MIRA!: An SDN-Based Framework for Cross-Domain Fast Migration of Ultra-Low Latency 5G Services," in *2018 IEEE Global Communications Conference, IEEE GLOBECOM*, Abu Dhabi, UAE, Dec. 2018.

[4] NTT DOCOMO, INC, "White Paper 5G Evolution and 6G," Tech. Rep., Jan. 2020.

[5] M. Latva-aho and K. Leppänen, "Key Drivers and Research Challenges for 6G Ubiquitous Wireless Intelligence," Tech. Rep., Sep. 2019.

[6] University of Oulu, "6G-Waves Magazine," Tech. Rep., Mar. 2020.

[7] T. Taleb *et al.*, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, Thirdquarter 2017.

[8] I. Afolabi *et al.*, "Network Slicing and Softwarization: A Survey on Principles, Enabling Technologies, and Solutions," *IEEE Communications Surveys Tutorials*, vol. 20, no. 3, pp. 2429–2453, Mar. 2018.

[9] A. Nakao *et al.*, "End-to-end Network Slicing for 5G Mobile Networks," *Journal of Information Processing*, vol. 25, pp. 153–163, Feb. 2017.

[10] T. Taleb *et al.*, "PERMIT: Network Slicing for Personalized 5G Mobile Telecommunications," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 88–93, May 2017.

[11] Z. Shu and T. Taleb, "A Novel QoS Framework for Network Slicing in 5G and Beyond Networks Based on SDN and NFV," *IEEE Network*, vol. 34, no. 3, pp. 256–263, Apr. 2020.

[12] I. Afolabi *et al.*, "Network Slicing-Based Customization of 5G Mobile Services," *IEEE Network*, vol. 33, no. 5, pp. 134–141, Oct. 2019.

[13] European Telecommunications Standards Institute (ETSI), "Network Functions Virtualisation (NFV) Release 3; Evolution and Ecosystem; Report on Network Slicing Support with ETSI NFV Architecture Framework," Tech. Rep., Dec. 2017.

[14] 3GPP, "Study on Management and Orchestration of Network Slicing for Next Generation Network," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 28.801, Jan. 2018.

[15] J. Ordonez-Lucena *et al.*, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, May 2017.

[16] I. Afolabi *et al.*, "Dynamic Resource Provisioning of a Scalable E2E Network Slicing Orchestration System," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2594–2608, July 2020.

[17] T. Taleb, I. Afolabi, K. Samdanis, and F. Z. Yousaf, "On Multi-Domain Network Slicing Orchestration Architecture and Federated Resource Control," *IEEE Network*, vol. 33, no. 5, pp. 242–252, July 2019.

0733-8716 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Oulu University. Downloaded on May 31,2021 at 05:44:38 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSAC.2021.3078501, IEEE Journal on Selected Areas in Communications

[18] R. A. Addad *et al.*, "Towards a Fast Service Migration in 5G," in *2018 IEEE Conference on Standards for Communications and Networking, IEEE CSCN*, Paris, France, Oct. 2018.

[19] ——, "Fast Service Migration in 5G Trends and Scenarios," *IEEE Network*, vol. 34, no. 2, pp. 92–98, Apr. 2020.

[20] ——, "Network Slice Mobility in Next Generation Mobile Systems: Challenges and Potential Solutions," *IEEE Network*, vol. 34, no. 1, pp. 84–93, Jan. 2020.

[21] ——, "Towards studying Service Function Chain Migration Patterns in 5G Networks and beyond," in *2019 IEEE Global Communications Conference, IEEE GLOBECOM*, Waikoloa, HI, USA, Dec. 2019.

[22] T. K. Rodrigues *et al.*, "Machine Learning Meets Computation and Communication Control in Evolving Edge and Cloud: Challenges and Future Perspective," *IEEE Communications Surveys Tutorials*, vol. 22, no. 1, pp. 38–67, Apr. 2020.

[23] D. M. Gutierrez-Estevez *et al.*, "Artificial Intelligence for Elastic Management and Orchestration of 5G Networks," *IEEE Wireless Communications*, vol. 26, no. 5, pp. 134–141, Oct. 2019.

[24] J. Park, S. Samarakoon, M. Bennis, and M. Debbah, "Wireless Network Intelligence at the Edge," *Proceedings of the IEEE*, vol. 107, no. 11, pp. 2204–2239, Nov. 2019.

[25] N. C. Luong *et al.*, "Applications of Deep Reinforcement Learning in Communications and Networking: A Survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3133–3174, Fourthquarter 2019.

[26] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA '17*, Toronto, ON, Canada, June 2017.

[27] M. Wang *et al.*, "Machine Learning for Networking: Workflow, Advances and Opportunities," *IEEE Network*, vol. 32, no. 2, pp. 92–99, Mar. 2018.

[28] M. E. Morocho-Cayamcela *et al.*, "Machine Learning for 5G/B5G Mobile and Wireless Communications: Potential, Limitations, and Future Directions," *IEEE Access*, vol. 7, pp. 137 184–137 206, Sep. 2019.

[29] S. Dargan *et al.*, "A Survey of Deep Learning and Its Applications: A New Paradigm to Machine Learning," *Archives of Computational Methods in Engineering*, June 2019.

[30] European Telecommunications Standards Institute (ETSI), "Experiential Networked Intelligence (ENI); ENI use cases," Tech. Rep., Sep. 2019.

[31] 3rd Generation Partnership Project (3GPP), "System Architecture for the 5G System; Stage 2," Tech. Rep. TS. 23.501, Mar. 2018.

[32] F. Farahnakian *et al.*, "LiRCUP: Linear Regression Based CPU Usage Prediction Algorithm for Live Migration of Virtual Machines in Data Centers," in *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, Santander, Spain, Sep. 2013.

[33] R. Shaw, E. Howley, and E. Barrett, "A Predictive Anti-Correlated Virtual Machine Placement Algorithm for Green Cloud Computing," in *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, Zurich, Switzerland, Dec. 2018.

[34] V. Ravi and H. S. Hamead, "Reinforcement Learning Based Service Provisioning for a Greener Cloud," in *2014 3rd International Conference on Eco-friendly Computing and Communication Systems*, Mangalore, India, Dec. 2014.

[35] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning, Springer*, vol. 8, no. 3-4, pp. 279–292, May 1992.

[36] T. Miyazawa, V. P. Kafle, and H. Harai, "Reinforcement Learning Based Dynamic Resource Migration for Virtual Networks," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, Portugal, May 2017.

[37] M. Duggan *et al.*, "A Reinforcement Learning Approach for Dynamic Selection of Virtual Machines in Cloud Data Centres," in *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, Dublin, Ireland, Aug. 2016.

[38] R. N. Calheiros *et al.*, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software—Practice & Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[39] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," *ArXiv*, vol. abs/1312.5602, 2013.

[40] S. S. Masoumzadeh *et al.*, "An Intelligent and Adaptive Threshold-Based Schema for Energy and Performance Efficient Dynamic VM Consolidation," in *Energy Efficiency in Large Scale Distributed Systems*, Vienna, Austria, Apr. 2013.

[41] European Telecommunications Standards Institute (ETSI), "Mobile Edge Computing (MEC); Deployment of Mobile Edge Computing in an NFV environment," Tech. Rep., Feb. 2018.

[42] ——, "Developing Software for Multi-Access Edge Computing," Tech. Rep., Feb. 2019.

[43] G. Brockman *et al.*, "OpenAI Gym," *ArXiv*, vol. abs/1606.01540, 2016.

[44] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, Sep. 2014.

[45] P. Berde *et al.*, "ONOS: Towards an Open, Distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14*, Chicago, Illinois, USA, Aug. 2014.

[46] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[47] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, May 1996.

[48] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, pp. 26–38, Nov. 2017.

[49] European Telecommunications Standards Institute (ETSI), "Multi-access Edge Computing (MEC); Study on MEC support for alternative virtualization technologies," Tech. Rep., Nov. 2019.

[50] Z. Zainuddin and O. Pauline, "Function Approximation Using Artificial Neural Networks," in *Proceedings of the 12th WSEAS International Conference on Applied Mathematics, MATH'07*, Stevens Point, Wisconsin, USA, Dec. 2007.

[51] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks*, vol. 2, no. 5, p. 359–366, July 1989.

[52] V. Mnih *et al.*, "Human-level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, pp. 529–33, Feb. 2015.

[53] ——, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning, ICML'16*, New York, NY, USA, June 2016.

[54] V. R. Konda and J. N. Tsitsiklis, "On Actor-Critic Algorithms," *SIAM Journal on Control and Optimization*, vol. 42, no. 4, p. 1143–1166, Apr. 2003.

[55] Z. Wang *et al.*, "Sample Efficient Actor-Critic with Experience Replay," *arXiv*, vol. abs/1611.01224, Aug. 2016.

[56] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Advances in neural information processing systems, NIPS'99*, Denver, CO, Nov. 1999.

[57] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *arXiv*, vol. abs/1511.05952, 2015.

[58] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences," *Machine Learning*, vol. 3, no. 1, p. 9–44, Aug. 1988.

[59] W. P. Jones and J. Hoskins, "Back-Propagation," *BYTE*, vol. 12, no. 11, p. 155–162, Oct. 1987.

[60] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv*, vol. abs/1412.6980, 2014.

[61] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *Proc. icml*, vol. 30, no. 1, 2013.

[62] G. E. Hinton and R. R. Salakhutdinov, "Replicated Softmax: an Undirected Topic Model," in *Advances in Neural Information Processing Systems, NIPS'09*, Vancouver, British Columbia, Canada, Dec. 2009.

[63] A. Paszke and al, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32, NIPS'19*, Vancouver, British Columbia, Canada, Dec. 2019.

[64] MOSA!C Lab research group , 2016. [Online]. Available: www.mosaic-lab.org

[65] MOSA!C Lab Research Group , "Triggers Selection in Network Slice Mobility Framework," 2020. [Online]. Available: https://github.com/MOSAIC-LAB-AALTO/drl_based_trigger_selection

[66] G. Dulac-Arnold *et al.*, "Deep Reinforcement Learning in Large Discrete Action Spaces," *arXiv*, vol. abs/1512.07679, 2015.

[67] J. Schulman *et al.*, "Trust Region Policy Optimization," *arXiv*, vol. abs/1502.05477, 2017.

[68] ——, "Proximal Policy Optimization Algorithms," *arXiv*, vol. abs/1707.06347, 2017.

0733-8716 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Oulu University. Downloaded on May 31,2021 at 05:44:38 UTC from IEEE Xplore. Restrictions apply.