MetaSlicing: A Novel Resource Allocation Framework for Metaverse

Nam H. Chu, Dinh Thai Hoang, Member, IEEE, Diep N. Nguyen, Senior Member, IEEE, Khoa T. Phan, Eryk Dutkiewicz, Senior Member, IEEE, Dusit Niyato, Fellow, IEEE, and Tao Shu

Abstract—Creating and maintaining the Metaverse requires enormous resources that have never been seen before, especially computing resources for intensive data processing to support the Extended Reality, enormous storage resources, and massive networking resources for maintaining ultra high-speed and low-latency connections. Therefore, this work aims to propose a novel framework, namely MetaSlicing, that can provide a highly effective and comprehensive solution in managing and allocating different types of resources for Metaverse applications. In particular, by observing that Metaverse applications may have common functions, we first propose grouping applications into clusters, called MetaInstances. In a MetaInstance, common functions can be shared among applications. As such, the same resources can be used by multiple applications simultaneously, thereby enhancing resource utilization dramatically. To address the real-time characteristic and resource demand’s dynamic and uncertainty in the Metaverse, we develop an effective framework based on the semi-Markov decision process and propose an intelligent admission control algorithm that can maximize resource utilization and enhance the Quality-of-Service for end-users. Extensive simulation results show that our proposed solution outperforms the Greedy-based policies by up to 80% and 47% in terms of long-term revenue for Metaverse providers and request acceptance probability, respectively.

Index Terms—Metaverse, MetaSlice, MetaInstance, MetaSlicing, sMDP, deep reinforcement learning, resource allocation.

I. INTRODUCTION

Although the Metaverse’s concept first appeared in 1992 [1], it has just been attracting more attention from academia and industry in the last few years, thanks to the recent advances in technologies (e.g., extended reality, 5G/6G networks, and edge intelligence) along with great efforts of many big corporations such as Facebook [2] and Microsoft [3]. The Metaverse is expected to bring a new revolution to the digital world [4]. Unlike existing virtual worlds (e.g., Second Life and Roblox), where the users’ presentations (e.g., avatars/characters) and assets are limited in specific worlds, the Metaverse can be realized as a seamless integration of multiple virtual worlds [5]. Each virtual world in the Metaverse can be created for a certain application, such as entertainment, education, and healthcare. Similar to our real lives, Metaverse users can bring their assets from one to another virtual world while preserving their values, and vice versa. Moreover, the Metaverse is expected to further integrate digital and physical worlds, e.g., digitizing the physical environment by the digital twin [6]. For example, in the Metaverse, we can create our virtual objects, such as outfits and paintings, and then bring them to any virtual world to share or trade with others. We also can share virtual copies of a real object in different virtual worlds. Thus, the Metaverse will bring total new experiences that can change many aspects of our daily lives, such as entertainment, education, e-commerce, healthcare, and smart industries [7], [8], [9].

However, extremely-high resource demand in the Metaverse is one of the biggest challenges that is impeding the deployment of the Metaverse [4]. To fulfill the Quality-of-Service (QoS) and meet the user experience requirements in the Metaverse, it demands enormous resources that may have never been seen before. First, Metaverse is expected to support millions of users simultaneously since each Metaverse application can host a hundred thousand users simultaneously. For example, the peak number of concurrent players of Counter Strike - Global Offensive is more than one million in 2021 [10]. It is forecasted that

Manuscript received 22 August 2022; revised 17 May 2023; accepted 14 June 2023. Date of publication 21 June 2023; date of current version 4 April 2024. This work was supported in part by the ARC Discovery Projects under Grant DP190102134, in part by the Australian Research Council under the DECRA project under Grant DE210100651, and in part by the National Research Foundation, Singapore, and Infocomm Media Development Authority under its Future Communications Research and Development Programme, DSO National Laboratories under the AI Singapore Programme AISG under Grant AISG2-RP-2020-019, in part by Energy Research Test-Bed and Industry Partnership Funding Initiative, Energy Grid (EG) 2.0 programme, DesCartes and the Campus for Research Excellence and Technological Enterprise (CREATE) programme, and MOE Tier 1 under Grant RGG87/22. The work of Nam Hoai Chu was supported in part by the Vingroup Science and Technology Scholarship Program for Overseas Study for Master’s and Doctoral Degrees. Recommended for acceptance by D. Yang. (Corresponding author: Khoa T. Phan.)

Nam H. Chu is with the School of Electrical and Data Engineering, University of Technology Sydney, Ultimo, NSW 2007, Australia, also with the Department of Computer Science and IT, La Trobe University, Melbourne, VIC 3086, Australia, and also with the Department of Telecommunications Engineering, University of Transport and Communications, Hanoi 78000, Vietnam (e-mail: namhoai.chu@student.uts.edu.au, h.chu@latrobe.edu.au, namch@utc.edu.vn).

Dinh Thai Hoang, Diep N. Nguyen, and Eryk Dutkiewicz are with the School of Electrical and Data Engineering, University of Technology Sydney, Ultimo, NSW 2007, Australia (e-mail: hoang.dinh@uts.edu.au; diep.nguyen@uts.edu.au; eryk.dutkiewicz@uts.edu.au).

Khoa T. Phan is with the School of Computing, Engineering and Mathematical Sciences (SCEMS), Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC 3086, Australia (e-mail: k.phan@latrobe.edu.au).

Dusit Niyato is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798 (e-mail: dniyato@ntu.edu.sg).

Tao Shu is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849 USA (e-mail: tschu@auburn.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TMC.2023.3288085, provided by the authors.

Digital Object Identifier 10.1109/TMC.2023.3288085
data usage on networks can be expanded more than 20 times by the operation of Metaverse [11]. Second, unlike the current online platforms (e.g., massive multiplayer online role-playing games where the uplink throughput can be much lower than that of downlink throughput [12]), the Metaverse requires extremely-high throughput for both uplink and downlink transmission links. The reason is that Metaverse users can create their digital objects and then share/trade them via this innovation platform. Therefore, to maintain the Quality-of-Experience (QoE) for users, the Metaverse’s demand for resources (e.g., computing, networking, and storage) likely exceeds that of any existing massive multiplayer online application [4].

More importantly, the required resource types are highly correlated. In particular, the Extended Reality (XR) technology is believed to be fully integrated into Metaverse’s applications such that users can interact with virtual and physical objects via their digital avatars, e.g., digital twin [4]. Therefore, it requires not only extensive computing to render three-dimensional (3D) objects, a large amount of data collected from perceived networks, e.g., the Internet of Things (IoT), but also an ultra-low delay communication to maintain a seamless user experience. Thus, intensive resources are required not only to contain and operate Metaverse applications but also to support massive data forwarding over networks. Given the above, resource management for Metaverse is a challenging task due to the mountainous size of resources of different types and the correlations between these types.

In this context, although deploying the Metaverse on the cloud is a possible solution, it leads to several challenges. First, the cloud is often located in a physical area (e.g., a data center), making it potentially a point-of-congestion when millions of users connect at once. Second, since users come from around the world, a huge amount of exchanged data puts stress on the communication infrastructure. This results in high delay, which severely impacts the Metaverse since the delay is one of the crucial drivers of user experience [13]. In this context, multi-tier resource allocation architecture, where the computing, storage, networking, and communication capabilities are distributed along the path from end-users to the cloud, is a promising solution for the Metaverse implementation.

In the literature, there are only a few attempts to investigate the Metaverse resource management [14], [15], [16], [17]. Specifically, in [14], the authors consider computing resource allocation for a single-edge computing architecture that has a limited computing resource to allocate for some nearby Metaverse users. Similarly, in [15] and [16], resource allocation at the edge is considered, but more resource types, i.e., computation and communication, are considered. In particular, the work in [15] proposes a pricing model-based resource management to accelerate the trading of Virtual Reality (VR) services between end-users and VR service providers. In [16], the authors address the stochastic demand problem for an application of education in the Metaverse. Specifically, they propose a stochastic optimal resource allocation method to minimize the cost for the virtual service provider. Unlike the above works, in [17], the authors propose an evolutionary game-based resource management for perception networks (e.g., IoT) that are used to collect data for the Metaverse.

It can be observed that none of the above studies considers the multi-tier computing architecture for resource allocation problem. Instead, their approaches are only appropriate for a single-tier edge computing architecture [14], [15], [16], [17]. However, as analyzed above, due to the extremely high resource demands of Metaverse applications, the single-tier computing resource model may not be appropriate and effective. First, relying on a single-tier computing architecture may not guarantee optimal performance for Metaverse applications. The available resources at the edge (i.e., near end-users) are often much lower than those of the cloud and may not satisfy the intensive resource requirements of Metaverse applications [18]. This can lead to high computational latency or even disrupting services due to the lack of resources. Second, all resources in the single-tier model concentrate in one place, possibly resulting in a single point of failure problem, and thus this single architecture has a low ability of scalability and flexibility. Third, the single-tier edge computing tends to initialize Metaverse applications near the user’s location. As such, if the user moves far away from that location, the system cannot guarantee a good Quality of Experience (QoE) [18]. Given the above, the single-tier architecture may not be an optimal solution for deploying Metaverse applications.

Furthermore, it can be observed that in the Metaverse, many Metaverse applications may share some common functions. For example, a digital map is indeed a common function between tourism and navigation applications. Currently, sharing functions among applications has already been made. For instance, Google Map’s Application Programming Interface (API) provides various functions (e.g., digital map, check-in, display live data synching with location [19]) that can be shared among many applications, e.g., Pokemon Go [20], Wooorld [21], and CNN iReport Map [22]. This special feature of the Metaverse indeed can be leveraged to maximize resource utilization for Metaverse applications. However, none of the above works can exploit the similarity among applications to improve resource utilization for the Metaverse. Moreover, in practice, users join and leave the Metaverse at any time, leading to the high uncertainty and dynamic of resource demands. Among the aforementioned works, only the study in [16] addresses the stochastic demands of users. Nevertheless, it only considers resource allocation for a Metaverse education application. In addition, this approach is only appropriate for a single-tier resource allocation and could not leverage the similarities among Metaverse applications’ functions in order to maximize system performance. Thus, there is an urgent need for an effective and comprehensive solution for the Metaverse to handle not only the massive resource usage but also the dynamic and uncertain resource demand.

To address all the aforementioned challenges, we propose a novel framework, namely MetaSlicing, to intelligently allocate diverse types of resources for Metaverse applications by analyzing the incoming requests and allocating appropriate resources, and thereby maximizing the whole system performance. First, we introduce the idea of decomposing an application into...
multiple functions to facilitate the deployment and management of Metaverse, which are highly complex. In particular, each function of an application can be initialized separately and placed at a different tier in the system according to functions’ requirements and tiers’ available resources. For example, functions with low latency requirements can be placed at a low tier (e.g., tier-1), while those with low update frequency can be placed at a higher tier. By doing so, the application decomposition can not only provide a flexible solution for deploying Metaverse applications but also utilize all networks’ resources from different tiers. Second, we propose a novel technique, called MetaInstance, to address extreme-high resource demands of the Metaverse. The MetaInstance aims to improve resource utilization by exploiting the similarities among Metaverse applications. To be more specific, applications with common functions will be grouped into a MetaInstance, and the common functions will be shared among these applications instead of creating one for each application. Therefore, this technique can save more resources. Finally, to address the uncertainty and dynamic of resource demands as well as the real-time response of applications in Metaverse, we develop a highly-effective framework based on the semi-Markov decision process together with a reinforcement learning algorithm that can automatically find the optimal admission control policy for the Metaverse without requiring the complete information about the dynamic and uncertainty of resource demand. We perform extensive simulations not only to explore the resilience of our proposed framework but also to gain insights into the key factors that can affect the system performance.

II. SYSTEM MODEL

In this paper, we consider a system model including three main parties, i.e., (i) End-users, (ii) Metaverse tenants, and (iii) the Metaverse Infrastructure Service Provider (MISP), as illustrated in Fig. 1. First, an end-user subscribes to a Metaverse tenant to request a Metaverse application, namely MetaSlice. Then, the Metaverse tenant will request the MetaSlice from the MISP according to its subscribed users’ demands. If a request is accepted, the MISP will allocate its resources to initiate this MetaSlice. In the following, we explain the main components together with their interactions in our proposed framework in more details.

A. MetaSlicing: Dynamic Resource Allocation Framework for Metaverse

1) Multi-Tier Resource Allocation Architecture-Based Metaverse: As discussed in the previous section, although the centralized cloud can be used for deploying Metaverse’s applications, it poses critical challenges, e.g., a point-of-congestion in the
network and a point of failure. In the literature, most existing works related to the Metaverse resource management consider single-tier edge computing architecture [14], [16], [17]. However, the single-tier edge architecture may not be appropriate and effective since the edge capacity is often limited while Metaverse applications often demand intensive resources and strict requirements. Moreover, in single-tier edge computing, Metaverse applications are likely created near the user’s location so that the user’s QoE requirements may not be satisfied if the user moves far away.

To address these challenges, a multi-tier resource allocation architecture can be leveraged to effectively and dynamically allocate resources for Metaverse applications. First, this architecture alleviates the extensive resource demands of Metaverse applications for both the Metaverse tenants and end-users. Second, this architecture enables the distribution of different types of resources, e.g., computing, storage, networking, and communication capabilities, along the path from end-users to the cloud. By doing so, Metaverse applications can leverage the resources placed near end-users, resulting in a low delay and high QoS for users. Third, when a user moves to a new location, these applications can be migrated to a site near the user’s new location to maintain the QoE. Thus, distributing resources increases the resilience of the system compared to the traditional centralized cloud-based resource allocation architecture.

Although the multi-tier resource architecture offers a promising approach to deploy Metaverse applications, it still faces several challenges. First, recall that Metaverse applications are heavily integrated with XR technologies requiring stringent latency. As a result, they are supposed to be created at the edge servers that are placed near users, i.e., tier-1, thereby possibly causing overload at low tiers. Note that the resource capacity at low tiers is often much lower than those at high tiers, e.g., cloud. As such, it may result in high latency in both computing and transmission or even interrupting services, thus reducing users’ QoE since latency is one of the most important factors to guarantee QoE in Metaverse. Second, a user can physically move while using MetaSlices, and the latency requirement may not be guaranteed if they move too far from the place where ongoing applications are created. This issue can be mitigated by migrating these MetaSlices to a new place near the user’s new location. However, the migration may introduce long delay due to the transmission of user’s data and the application re-initialization. In the following subsection, we will discuss our technique (i.e., MetaSlice decomposition) to alleviate these problems.

2) MetaSlice Decomposition: Recall that the Metaverse can be realized as a seamless integration of multiple virtual worlds, and each virtual world can be regarded as an application for a specific purpose, e.g., entertainment, healthcare, or education [7], [8], [9]. We observe that an application may have several functions that can operate independently. For example, a tourism application may have a recommendation, digital map, and driving assistant functions. Based on the user’s location, the recommendation function can suggest nearby attractive places, and then the user can use the digital map function to get more information about these places. After that, the user can use the driving assistant function to get to the chosen place. Since these functions can be used or unused depending on different users, they can run separately without interfering with other functions. As such, Metaverse applications should be developed in a modular way by which their functions can be initialized and operated independently. By doing so, these independent functions can be connected to each other via application programming interfaces (API), similar to how existing applications connect to current online services, e.g., Google Maps API. Given the above, this paper considers that a MetaSlice, i.e., application, can be decomposed into multiple dependent functions.

Note that this work does not focus on optimal application decomposition, and we assume that applications can be decomposed by using existing methods, e.g., [23], [24]. We consider that each function is allocated dedicated resources due to the strict QoS requirements of Metaverse applications. In particular, a function with the dedicated resource allocation scheme likely executes faster than this function under the dynamic scheme. This is because, under the dynamic resource allocation scheme, a resource allocation procedure is performed whenever a request for processing arrives at a function, leading to an unavoidable additional delay that may degrade the QoS of Metaverse users. In addition, because resources are not reserved for each function in a dynamic allocation scheme, a function may not have sufficient resources to support an application’s request that arrives when a server is overloading with other functions, leading to a high delay or even service disruption. This problem can be alleviated by migrating the function to another server with adequate resources; however, it introduces another additional delay to the function execution. In this context, dedicated resource allocation can avoid this issue by reserving resources for each function. Moreover, the study in [25] points out that under a typical load of business applications, the dedicated resource allocation can achieve a better energy efficiency than that of dynamic resource allocation. In practice, the MetaSlice decomposition can offer numerous benefits for deploying and managing MetaSlices in Metaverse. First, as functions are independent entities and are connected via API, they can be developed and upgraded independently and simultaneously, thereby speeding up MetaSlice’s deployment and evolution. Second, this technique also helps Metaverse developers to concentrate on building their distinctive functions (e.g., a recommendation function in tourism) rather than putting their resources on other functions (e.g., driving assistant or digital map) that can be effectively developed by specialized third parties. Third, the MetaSlice decomposition can help to manage MetaSlices more convenient. For instance, when a function in a MetaSlice fails, users can be quickly redirected to a comparable function to alleviate service disruption. Fourth, the MetaSlice decomposition with a multi-tier resource architecture provides flexibility for Metaverse implementation. In particular, functions can be initiated at different tiers since they are independent entities. For example, a MetaSlice for travel may consist of several major functions, such as a digital map, real-time traffic, real-time weather, and a driving assistant. In this case, these functions can be placed dynamically in the multi-tier resource allocation architecture depending on the functions’ requirements. Specifically, real-time traffic and driving assistant
functions can be placed at tier-1 near end-users since they require low delay, while a digital map (that does not need frequent updates) can be located at a high tier, e.g., at the cloud. Finally, this technique can also alleviate the application migration problem in a multi-tier resource architecture. Particularly, by leveraging application decomposition, instead of migrating a MetaSlice, only functions with stringent delay requirements (e.g., less than 20 ms) will be migrated, thereby significantly decreasing migration delay. Thus, the application decomposition technique offers a flexible and effective implementation of multi-tier-based MetaSlices.

To support the decomposition technique, we consider that a MetaSlice is created based on the MetaBlueprint, i.e., a template describing the workflow, configuration, and structure for initializing and managing this MetaSlice during its life cycle. From a technical standpoint, a MetaSlice (i.e., an application in the Metaverse) can be analogous to the network slice paradigm in the fifth generation of the cellular network (5G) that consists of multiple network functions. It is worth mentioning that slicing is the general term in describing virtualization methods that can partition and organize resources (e.g., computing and networking) of the physical infrastructure to flexibly support diverse requirements. As such, the proposed MetaSlicing mechanism seems similar to the network slicing mechanism in 5G networks due to their names, but they are essentially different, as shown in Table I.

| Approach                  | Aim                                                                 | Novel Features                                                                 |
|---------------------------|----------------------------------------------------------------------|--------------------------------------------------------------------------------|
| Virtualized network function (VNF) allocation | Distributing VNFs under limited resources. | Utilizing the virtualization technologies to virtualize network devices’ functions (such as routing, switching, and load balancing) that are used to create and deliver communication services [27]. |
| Network slicing           | Creating of multiple virtual networks (i.e., network slices) that can coexist on a single physical network infrastructure [28]. | Leveraging VNFs to create various specialized and dedicated network services for different applications, such as autonomous vehicles, manufacturing, or virtual reality applications [28], [29]. |
| Our proposed MetaSlicing   | Creating of multiple virtual worlds or “Metaverses” that can be accessed and interacted with by users through a variety of devices and platforms. | - The application decomposition (with the underlying multi-tier resource allocation architecture) help enhance manageability and accelerate the deployment and evolution of Metaverse applications (i.e., MetaSlice). - The MetaInstance can exploit the functional similarities between MetaSlices to maximize resource utilization. |

3) MetaInstance: To further take advantage of MetaSlice decomposition, we introduce the MetaInstance that can improve system resource utilization. Suppose that Metaverse may consist of different MetaSlice types, e.g., tourism, education, industry, and navigation. In addition, MetaSlices can also be grouped into I classes based on their characteristics, such as occupied resources, technical configurations, and QoS. For example, navigation MetaSlice may have driving assistant functions requiring ultra-low latency and highly-reliable connections, while security and resilience are among the top concerns of e-commerce and industry MetaSlices. Moreover, different types of MetaSlice may share the same functions. For instance, tourism and navigation MetaSlices can use the same underlying digital map and the real-time traffic and weather functions whose data are collected by the same perception network, e.g., IoT. Furthermore, we can observe that different Metaverse tenants can create/manage multiple variants from the same type of MetaSlice (e.g., education, industry, or navigation). Therefore, ongoing MetaSlices may share the same functions. In this case, a lot of resources can be shared, leading to greater resource utilization and higher revenue for the Metaverse Infrastructure Service Provider (MISP).

Based on this fact, Metaverse applications can be classified into groups, namely MetaInstances. Thus, a MetaInstance can be defined by two function types, i.e., (i) shared function and (ii) dedicated function belonging to specific MetaSlices, as
illustrated in Fig. 1. In this case, a MetaInstance can maintain a function configuration consisting of a list of functions and a description of interactions among them. Because the capability of a function is limited, sharing a function for too many MetaSlices may lead to a decrease in user experience (e.g., processing delay) or even service disruption. Therefore, in practice, a function can be only shared by a maximum number of \( N_L \) MetaSlices. From the technical perspective, the implementation of a MetaInstance can be similar to that of the Network Slice Instance (NSI), where network slices can share some Network Functions (NFs) [28].

It is worth mentioning that even though the MetaSlicing may be similar to the network slicing [28], they are actually not the same. In particular, network slicing aims to address the diversity (or even conflict) in communication requirements among various businesses by running multiple logical networks (i.e., Network Slices) over a physical network. For example, one (e.g., automotive customers) may require ultra-low latency connections while others (e.g., manufacturing customers) require ultra-reliable connections. Thus, the network slicing focuses on providing diverse types of communications. In contrast, MetaSlicing is a framework to effectively manage resources in a multi-tier computing architecture by decomposing an application into functions and then optimally distributing them at different tiers.

In addition, as explained in the previous paragraphs, due to different features of Metaverse applications, resource allocation scheme for MetaSlicing is also designed different from that of the network slicing, where dynamic resource allocation approaches are preferred [31].

Based on the aforementioned analysis, we can observe that, on the one hand, our proposed MetaSlicing framework can offer a great solution to the MISP by maximizing the resource utilization and at the same time minimizing the deployment cost and initialization time for Metaverse’s applications. On the other hand, this framework can also benefit end-users by achieving greater user experience, e.g., lower delay and more reliable services. To achieve these results, the Admission Controller in MetaSlicing plays a critical role. For example, accepting requests of MetaSlices that share some functions with the ongoing MetaSlices may help the system to save more resources than accepting those with less or without sharing functions with the MetaSlices running in the system. In addition, Resource Management is another important factor determining the Metaverse system performance. In the following subsection, we explain these components in our proposed MetaSlicing framework.

B. Admission Control and Resource Management

Recall that according to the demands of subscribed end-users, a MetaSlicer tenant sends a MetaSlice request associated with MetaBlueprint to the MISP. Then, the Admission Management and Resource Management are executed as follows. As shown in Fig. 1, the Admission Management block of the MISP consists of a MetaSlice Analyzer and an Admission Controller. Once receiving a MetaSlice request, the MetaSlice Analyzer will analyze the request’s MetaBlueprint to determine similarities between the functions and configuration of the requested MetaSlice and those of the ongoing MetaInstances. In this paper, we consider that a MetaBlueprint consists of at least (i) the function configuration record, including the list of required functions and the description of interactions among them, and (ii) the MetaSlice configuration (e.g., class ID and required resources). Based on the similarity analysis (to be presented in Section IV) obtained from the MetaSlice Analyzer and the currently available resources of the system, the Admission Controller decides whether to accept the request or not according to its admission policy.

Suppose that a MetaSlice request is accepted, the Resource Management allocates the accepted MetaSlice to a MetaInstance with the highest similarity index and updates this MetaInstance accordingly. Specifically, if the accepted MetaSlice has dedicated functions, system resources are allocated to initiate these new functions. If the current MetaInstances do not share any function with the new MetaSlice, a new MetaInstance is created for the accepted MetaSlice. When a MetaSlice departs/complete, its resources will be released, and the MetaInstance will be updated accordingly.

In this paper, we consider \( D \) resource types owned by the MISP, e.g., computing, networking, and storage. Then, the required resources for a MetaSlice \( m \) can be represented by a resource vector, i.e., \( n_m = [n_{m1}, \ldots, n_{md}, \ldots, n_{mD}] \), where \( n_{md} \) is the amount of type \( d \) resources. In this case, the total occupied resources by all MetaSlices cannot exceed the maximum resources of MISP, i.e.,

\[
\sum_{m \in M} n_{md} \leq N^d, \quad \forall d \in \{1, \ldots, D\},
\]

where \( N^d \) is the total amount of type \( d \) resources of the MISP, and \( M \) is the set of all running MetaSlices in the system. In our proposed solution, the system’s available resources and the required resources for the request are two crucial factors for the admission control in MetaSlicing. However, in practice, the future requests’ arrival process and its required resources are likely unknown in advance. In addition, the departure process of MetaSlices (i.e., how long a MetaSlice remains in the system) is also highly dynamic and uncertain. Therefore, in the next section, we will introduce a framework based on the semi-Markov decision process to address these challenges.

III. MetaSlicing Admission Control Formulation

In this paper, we propose a highly-effective semi-Markov Decision Process (sMDP) framework to address the MetaSlice admission control problem due to the following reasons. First, the sMDP can enable the MetaSlicing’s Admission Controller to adaptively make the best decisions (i.e., whether to accept or reject a MetaSlice request) based on the current available system resources (i.e., computing, networking, and storage) and the MetaSlice request’s blueprint (i.e., resource, class and similarity) without requiring complete information about the surrounding environment (e.g., arrival and departure processes of MetaSlices) to maximize the MISP’s long-term revenue. Second, in practice, MetaSlice requests can arrive at any time, so the admission decision needs to be made as soon as possible.
However, the conventional Markov Decision Process (MDP) only takes an action at each time slot with an equal time period, making it unable to capture real-time events, e.g., request arrival [32]. In contrast, the sMDP makes a decision whenever an event occurs so that it can perfectly capture the real time of MetaSlicing. Finally, the MetaSlice’s lifetime is highly uncertain. Upon a MetaSlice departs from the system, its occupied resources are released, and the system state transits to a new state immediately. Again, the conventional MDP is unable to capture this transition as it works in a discrete-time fashion.

To that end, the sMDP will be used in our framework to enable the MISP to make real-time decisions and maximize its long-term revenues. Technically, an sMDP can be defined by a set of five components, including (i) the decision epoch $t$, (ii) the state space $S$, (iii) the action space $A$, (iv) the transition probability $T$, and (v) the immediate reward function $r$. In the following sections, we will explain how this framework can capture all events in the MetaSlice system and make optimal decisions for the MISP.

### A. Decision Epoch

The decision epochs are defined as points of time at which decisions are made [32]. In our real-time MetaSlicing system, the Admission Controller must make a decision once a MetaSlice request arrives. Therefore, we can define the decision epoch as an interval between the occurrence of two consecutive requests.

### B. State Space

Aiming to maximize revenue for the MISP with limited resources, several important factors need to be considered in the system state space. First, the current system’s available resources and the resources required by the current MetaSlice request are the two most important factors for the Admission Controller to decide whether it accepts this current request or not. Second, since the income for leasing each MetaSlice class is different, the class ID $i$ of a requested MetaSlice is another crucial information. Third, the similarity index $j$ of a requested MetaSlice reflects the similarity between a requested MetaSlice and the ongoing MetaInstances. Recall that the higher the value of $j$ is, the more similarity between the requested MetaSlice and the running MetaInstances is, leading to lower occupied resources when deploying this new request. Hence, the similarity index is also an important factor for the MetaSlice admission decision.

We denote the available resources of the system by a vector $n_u = [n_u^1, \ldots, n_u^d, \ldots, n_u^D]$ where $D$ is the total number of resource types, and $n_u^d$ denotes the number of available resources of type $d$. Similarly, the request’s required resources are denoted by $n_m = [n_m^1, \ldots, n_m^d, \ldots, n_m^D]$, where $n_m^d$ is the number of requested resources of type $d$. Given the above, the system state space can be defined as follows:

$$S \triangleq \{ (n_u^1, \ldots, n_u^d, \ldots, n_u^D, n_m^1, \ldots, n_m^d, \ldots, n_m^D, i, j) : n_u^d \text{ and } n_m^d \in \{0, \ldots, N^d\}\forall d \in \{1, \ldots, D\}; i \in \{1, \ldots, I\}; j \in \{0, \ldots, J\}\},$$

where $N^d$ is the maximum resources of type $d$, $I$ is the total number of classes in the system, and $J$ is the maximum similarity index of a MetaSlice derived by the MetaSlice Analyzer. By this design, the system state is presented by a tuple, i.e., $s \triangleq (n_u, n_m, i, j)$, and the system can work continuously without ending at a terminal state, at which the system stops working [33].

Recall that in the sMDP framework, the system only transitions from state $s$ to state $s'$ if and only if an event occurs (e.g., a new MetaSlice request arrival). We define the event as a vector $e = [e_1, \ldots, e_d, \ldots, e_f]$, where $e_i \in \{-1, 0, 1\}$. Specifically, $e_i = -1$ if a MetaSlice class $i$ departs from the system, $e_i = 1$ if a new MetaSlice request class-$i$ arises, and $e_i = 0$ otherwise (i.e., no MetaSlice request class-$i$ arrival or departure).

Thus, the set of all possible events is given as follows:

$$E \triangleq \{ e : e_i \in \{-1, 0, 1\}; \sum_{i=1}^{I} |e_i| \leq 1 \}.$$

Note that there is a trivial event $e^* \triangleq (0, \ldots, 0)$ meaning that no MetaSlice request of any class arrives or departs.

### C. Action Space

If a MetaSlice request arrives at state $s$, the Admission Controller can decide whether to accept or reject this request to maximize the long-term revenue for the MISP. Thus, the action space at state $s$ can be defined by

$$A_s \triangleq \{0, 1\}.$$

In particular, if the requested MetaSlice is accepted, the action at state $s$ is equal to one, i.e., $a_s = 1$. Otherwise, $a_s = 0$.

Recall that a user needs to subscribe to a Metaverse tenant to access a MetaSlice, i.e., a Metaverse application. As such, we assume that users’ requests are queued at the corresponding Metaverse tenant before being forwarded to the MISP. If the MISP rejects a user’s request, it will return to the queue at the Metaverse tenant subscribed by this user until it is served or the user decides to stop the request. Since the lifetime of a MetaSlice is limited, and resources are released whenever a MetaSlice departs, a rejected request is likely served eventually.

It is important to note that this work does not focus on managing the queue, and we assume that the queue management is independently controlled by each Metaverse tenant. In addition, a Metaverse tenant can rent resources from several MISPs to provide alternative access for their subscribed users if the current MISP runs out of resources. Again, this problem is also out of the scope of this work. Thus, this is a potential direction for future works.

### D. State Transition Probabilities

This sub-section analyzes the sMDP’s dynamic by characterizing the underlying Markov chain’s state transition probabilities. Since the sMDP is based on the semi-Markov Process (sMP) that consists of a renewal process and a Continuous-time Markov Chain (CTMC) $\{X(t)\}$, the uniformization method can be used to derive the state transition probabilities $T$ [32], [34], [35]. Specifically, the uniformization transforms the CTMC into
E. Immediate Reward Function

To maximize the MISP’s long-term revenue, the income from leasing resources to Metaverse tenants should be captured by the immediate reward function. Here, we consider that the revenues for leasing resources for different classes are different since different classes may have different requirements such as reliability and delay. Recall that in our proposed Metaverse system, MetaSlices can share some functions with others, leading to differences in resource occupation even between MetaSlices from the same class. As such, even if two MetaSlices give the same revenue, accepting a MetaSlice that requires fewer resources will benefit the provider in the long term. Therefore, the number of resources required by a MetaSlice is another key factor.

To this end, the immediate reward function can be defined as follows:

$$r(s, a) = \begin{cases} r_i - \sum_{d=1}^{D} w_{d} n_{d}^{d}, & \text{if } e_i = 1 \text{ and } a = 1, \\ 0, & \text{otherwise}, \end{cases}$$  \quad (8)$$

where $r_i$ is the revenue from leasing resources for a MetaSlice class $i$, and $n_{d}^{d}$ is the number of type $d$ resources occupied by this requested MetaSlice. The trade-offs between these factors are reflected by weights, i.e., \[ w_{d} \] \forall d. Equation (8) implies that if slices have the same income (i.e., they are in the same class), accepting requests with fewer resource demands can help the provider to maximize the long-term revenue.

Note that a weight $w_{d}$ reflects the price for renting one unit of resources type $d$ (e.g., 1 TB storage) from the Metaverse Infrastructure Service Provider (MISP). In the literature, resource pricing is a well-investigated problem, which can be addressed by various methods such as optimization theory [36] or game theory [37]. Generally, resource prices can be determined by the amount of remaining resources, i.e., the higher amount of remaining resources type $d$ is, the lower its weight is. In practice, the resources are priced based on MISPs’ business strategies, e.g., [38], [39]. Note that optimizing weights for resource types is out of scope in this work. In addition, the simulations in Section V are performed to study the impact of immediate reward on the system performance.

F. Optimization Formulation

Since the statistical characteristics (e.g., arrival rate and departure rate of a MetaSlice) of the proposed system are stationary (i.e., time-invariant), the policy $\pi$ for the meta-controller can be described as the time-invariant mapping from the state space to the action space, i.e., $\pi : S \rightarrow A_{\pi}$. This study aims to find an optimal policy for the MetaSlicing’s Admission Controller that maximizes a long-term average reward function $\mathcal{R}_{\pi}(s)$, which is defined as an average expected reward obtained by starting from state $s$ and following policy $\pi$ as follows:

$$\mathcal{R}_{\pi}(s) = \lim_{G \to \infty} \frac{E \left[ \sum_{g=0}^{G} r(g, \pi(s)) | s_0 = s \right]}{E \left[ \sum_{g=0}^{G} \tau_g | s_0 = s \right]}, \forall s \in S,$$  \quad (9)$$

where $G$ is the total number of decision epochs, $\pi(s_g)$ is the action derived by $\pi$ at decision epoch $g$, and $\tau_g$ is the interval time between two consecutive decision epochs. The existence of the limit in $\mathcal{R}_{\pi}(s)$ is proven in Theorem 1. Thus, given the currently available resources and the MetaSlice request’s information, the optimal policy $\pi^\star$ can give optimal actions to maximize $\mathcal{R}_{\pi}(s)$, thereby maximizing the long-term revenue for the infrastructure provider.

**Theorem 1:** Given that the state space $S$ and the number of decision epochs in a certain period of time are finite, we have

$$\mathcal{R}_{\pi}(s) = \lim_{G \to \infty} \frac{E \left[ \sum_{g=0}^{G} r(g, \pi(s)) | s_0 = s \right]}{E \left[ \sum_{g=0}^{G} \tau_g | s_0 = s \right]}$$

$$= \frac{T_{\pi} \mathbb{E}[r(s, \pi(s))]}{T_{\pi} \mathbb{E}[\tau(s, \pi(s))]}, \quad \forall s \in S,$$  \quad (10)$$

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
The proof of Theorem 1 is presented in Appendix A, available online.

Note that the underlying Markov chain of our sMDP model is irreducible (i.e., the long-term average reward is independent of the starting state), which is proven in Theorem 2.

Theorem 2: The long-term average reward $R_{\pi}(s)$ for any policy $\pi$ is well-defined and independent of the starting state, i.e., $R_{\pi}(s) = R_{\pi}, \forall s \in S$.

Proof: The proof of Theorem 2 is presented in Appendix B, available online.

Since the limiting matrix $T_{\pi}$ exists and the sum of probabilities that the system moves from one state to others is one, we have $\sum_{s' \in S} T_{\pi}(s'|s) = 1$. Given the above, the MetaSlice admission control problem can be formulated as follows:

$$\max_{\pi} R_{\pi} = \frac{T_{\pi}r(s, \pi(s))}{T_{\pi}y(s, \pi(s))},$$

subject to: $\sum_{s' \in S} T_{\pi}(s'|s) = 1, \forall s \in S$. (13)

Note that while the problem studied in this work involves resource management for Metaverse, it may seem at first glance that our problem formulation is similar to conventional resource allocation problems, such as virtualized network function (VNF) resource allocation or network slicing [40], [41], [42]. However, they are fundamentally different. In particular, our proposed resource management framework, MetaSlicing, leverages the similarities in functions between Metaverse applications to improve system resource utilization significantly. In contrast, conventional resource allocation approaches do not consider function similarities between applications when allocating resources [14], [16], [17]. Therefore, it makes our mathematical optimization problems clearly different from conventional resource allocation. We first propose the MetaSlice decomposition to divide a Metaverse application into different functions that can operate and be initialized independently. Then, we introduce the MetaInstance technique to group Metaverse applications into different groups according to their function similarities in which some functions can be shared by multiple Metaverse applications. For that, we propose a method to identify the similarities in functions between MetaSlices. Finally, an admission control problem can be adopted to allocate resources for requests of MetaSlices based on system’s available resources and similarity between the MetaSlice request and the ongoing MetaSlices in the system. Clearly, in addition to optimizing different types of resources as in conventional VNF resource optimization, we need to consider other important factors related to the deployment of Metaverse applications in practice, i.e., decomposition of Metaverse applications (i.e., MetaSlices) and identifying similarities between MetaSlices. Thus, the considered problem requires a fundamentally different approach than conventional VNF resource allocation approaches.

In the following section, we will discuss our proposed solution that can help the Admission Controller to obtain the optimal admission policy $\pi^*$ to maximize the long-term average reward function, i.e., $\pi^* = \arg\max_{\pi} R_{\pi}$.

IV. AI-BASED SOLUTION WITH METASLICE ANALYSIS FOR METASLICE ADMISSION MANAGEMENT

This section presents our proposed approach for the MetaSlice admission management. We first discuss the main steps in the MetaSlice analysis to determine the similarity between a requested MetaSlice and ongoing MetaSlices. Then, we propose a Deep Reinforcement Learning (DRL)-based algorithm for the Admission Controller to address the real-time decision making requirement and the high uncertainty and dynamics of the request’s arrival and MetaSlice departure processes, which are, in practice, often unknown in advance. Thanks to the self-learning ability of DRL, the Admission Controller can gradually obtain an optimal admission policy via interactions with its surrounding environment without requiring complete knowledge of the arrival and departure processes of MetaSlices in advance.

A. MetaSlice Analysis

Recall that the major role of the MetaSlice Analyzer is to analyze the request’s MetaBlueprint to determine the similarity between the requested MetaSlice and the ongoing MetaInstances. Then, the similarity report is used to assist the Admission Controller in deciding whether to accept or reject a request. This paper uses the function configuration to decide the similarity index since it clearly shows the relationship between the requested MetaSlice and the ongoing MetaInstance.

We consider that the proposed framework supports $F$ types of functions. Then, we can denote the function configuration of a MetaSlice $m$ by a set $F_m$ as follows:

$$F_m = \{ f_1^m, \ldots, f_j^m, \ldots, f_F^m \},$$ (14)

where the configuration of function $f$ is represented by a vector with size $K$, i.e., $f = \{ 0, 1 \}^K$. We define a trivial function configuration vector $f^* \triangleq \{ 0, \ldots, 0 \}$ meaning that a function is not required by the MetaSlice. Note that a MetaInstance also maintains a function configuration set and updates it whenever the MetaSlice is initialized or released.

Given two function configuration sets $F_1$ and $F_2$, the similarity index can be given as follows:

$$j(F_1, F_2) = \frac{1}{F} \sum_{f=1}^{F} b(f_1, f_2),$$ (15)

where $b$ can be any similarity function used for vectors such as Jaccard and Cosine similarity functions [43]. Here, we use
Jaccard similarity that is defined as follows:

\[
b_{\text{jaccard}}(f_j^1, f_j^2) = \frac{f_j^1 \cdot f_j^2}{||f_j^1||^2 + ||f_j^2||^2 - f_j^1 \cdot f_j^2},
\]

(16)

where the numerator is the dot product of two vectors and \(|| \cdot ||\) is the euclidean norm of the vector, which is calculated as the square root of the sum of the squared vector’s elements.

Recall that the similarity index \(j\) is one of the four elements of the state, and in reinforcement learning, an agent determines its action based on the observation of the state. Thus, the similarity index plays two important roles in the MetaSlicing framework. First, it supports the Admission Controller based on a reinforcement learning algorithm with precious information for making decisions. Second, based on the accepted request’s similarity index, MetaSlicing’s resource management determines to put the accepted MetaSlice request in an existing MetaInstance or create a new MetaInstance.

B. Deep Dueling Double Q-Learning Based-Admission Controller

In Reinforcement Learning (RL), Q-learning is widely adopted due to its simplicity in implementation and convergence guarantee after the learning phase [44]. Nevertheless, using a table to estimate the optimal values of all state-action pairs \(Q^\ast(s, a)\), i.e., Q-values, hinders the Q-learning from being applied in a high-dimensional state space as the problem considered in this paper with hundred thousand states [45]. In addition, the usage of the Q-table is only feasible when values of states are discrete, but the similarity score in the considered state can be a real number. These challenges are addressed by deep Q-network (DQN) algorithm, in which a Deep Neural Network (DNN), instead of a Q-table, is used to approximate \(Q^\ast(s, a)\) for all state-action pairs [46]. However, both Q-learning and DQN have the same problem of overestimation when estimating Q-values [47]. This issue makes the learning process unstable or even results in a sub-optimal policy if overestimations are not evenly distributed across states [48].

To that end, this work proposes a deep RL algorithm for the MetaSlicing Admission Controller, namely imSAC, that can address these above issues effectively by leveraging three innovative techniques: (i) the memory replay mechanism, (ii) the Dueling neural network architecture, and (iii) the double deep Q-network (DDQN) algorithm.

Note that most recently proposed DRL algorithms are on-policy methods, e.g., Asynchronous Advantage Actor-Critic (A3C) and Proximal Policy Optimization (PPO), and aim to overcome the limitation of DQN in problems with continuous action space [49], [50]. As such, they may not perform well in problems with discrete action space, such as the considered problem in this work. For instance, the study in [51] shows that the Dueling DDQN outperforms the A3C in most of 57 Atari games with discrete action spaces. Similarly, results in [52] show that the performance of PPO and Dueling DDQN are similar in a discrete action problem, and the run time of PPO is almost two times greater than that of Dueling DDQN. Moreover, the authors in [53] demonstrate that PPO’s performance is heavily influenced by its code-level implementation. Hence, the PPO may require significant effort to optimize hyper-parameters, which is not the focus of our work. In contrast, Dueling DDQN can achieve good performance with typical hyper-parameters, as those in [46], [54]. More importantly, since A3C and PPO are on-policy methods, they must use up-to-date experiences collected by \(e\) their current policy to learn an optimal policy [33]. As such, old experiences are dropped after each learning iteration, making the sample efficiencies of these methods low. Here, the sample efficiency presents how much an RL algorithm can get the most use of every experience. In contrast, the Dueling DDQN is an off-policy method that can leverage experiences collected from any other policies [33]. Given the above observations, the Dueling DDQN is the most appropriate learning approach for the considered problem as it can perform well with typical hyper-parameters and has high sample efficiency.

The details of imSAC are presented in Algorithm 1. The rest of this subsection will explain the key components of the imSAC. Given that the learning phase of the proposed algorithm consists of \(T\) time steps, and at time step \(t\), the Admission Controller observes the current state \(s_t\) and takes an action \(a_t\) according to the \(\epsilon\)-policy. After that, it observes a next state \(s_{t+1}\) and gets a reward \(r_t\). This experience data, represented by a tuple \(<s_t, a_t, s_{t+1}, r_t>\), cannot be used directly to train the DNN since the consecutive experiences are highly correlated, which may lead to a slow convergence rate [55]. As such, we adopt the memory replay mechanism where experiences are stored in a buffer \(B\), as illustrated in Fig. 3. Then, at each time step, experiences are sampled uniformly at random to train the DNN. By doing so, correlations among experiences can be removed, thereby accelerating the learning process. Moreover, as one data point can be used multiple times to train the DNN, this mechanism can indeed improve the data usage efficiency.

In the proposed imSAC, since a DNN is used to approximate the Q-values, we define the input and output layers of the DNN according to the state dimensions (i.e., available system resources, requested MetaSlice’s resources, class ID, and similarity score) and action space. Specifically, when feeding a state to the DNN, we obtain the Q-values for all actions at this state, each given by a neuron at the DNN’s output layer. To improve the stability and increase the convergence rate, we propose to use the state-of-the-art dueling neural network architecture [54]...
Algorithm 1: The iMSAC.
1: Initialize $\epsilon$ and buffer B.
2: Create Q-network $Q$ with random parameters $\phi$.
3: Create target Q-network $\hat{Q}$ by cloning the Q-network.
4: for step = 1 to $T$ do
5:   Get action $a_t$ following the $\epsilon$-greedy policy as follows:
6:   Perform $a_t$, then observe reward $r_t$ and next state $s_{t+1}$.
7:   Store experience $(s_t, a_t, r_t, s_{t+1})$ in B.
8:   Create a mini-batch of experiences by sampling randomly from buffer B, i.e., $(s, a, r, s') \sim U(B)$.
9:   Obtain the Q-value and the target Q-value by using (20) and (22), respectively.
10: Update $\phi$ based on Stochastic Gradient Descent (SGD) algorithm.
11: Decrease the value of $\epsilon$.
12: Set $\phi' = \phi$ at every $C$ steps.
13: end for

for the iMSAC’s DNN, as shown in Fig. 2. In particular, the dueling architecture divides the DNN into two streams. The first one estimates the state-value function $V(s)$, which indicates the value of being at a state $s$. The second stream estimates the advantage function $\hat{V}(s, a)$ that demonstrates the importance of action $a$ compared to other actions at state $s$.

Note that the state-action value function $Q(s, a)$ (namely Q-function) expresses the value of taking an action $a$ at state $s$, i.e., Q-value. Thus, the advantage function under policy $\pi$ can be given as $\forall^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ [54]. Then, we can obtain the estimated Q-function for feeding state $s$ to the i-AC’s Deep Dueling Neural Network (DDNN) as follows:

$$Q(s, a; \zeta, \beta) = V(s; \zeta) + \hat{V}(s, a; \beta),$$

where $\zeta$ and $\beta$ are the parameters of the state-value and advantage streams, respectively. It can be observed that given $Q$, $V$ and $\hat{V}$ could not be determined uniquely. For instance, $Q$ is unchanged if $\hat{V}$ decreases the same amount that $V$ increases. As such, using (18) directly may result in a poor performance of the algorithm. Therefore, similar to [54], we propose to use the following output of the advantage stream:

$$Q(s, a; \zeta, \beta) = V(s; \zeta) + \left(\forall^\hat{V}(s, a; \beta) - \max_{a' \in A_s} \forall^\hat{V}(s, a'; \beta)\right).$$

In this way, for the optimal action $a^*$ at state $s$, i.e.,

$$a^* = \argmax_{a \in A_s} Q(s, a; \zeta, \beta) = \argmax_{a \in A_s} \forall^\hat{V}(s, a; \beta),$$

the $Q(s, a^*; \zeta, \beta)$ is forced to equal $V(s, \zeta)$. However, (19) still faces an issue, i.e., the advantage function changes at the same speed at which the advantage of the predicted optimal action changes, making the estimation of Q-values unstable. To address this issue, the max operator is replaced by the mean as follows [54]:

$$Q(s, a; \zeta, \beta) = V(s; \zeta) + \left(\forall^\hat{V}(s, a; \beta) - \frac{1}{|A_s|} \sum_{a' \in A_s} \forall^\hat{V}(s, a'; \beta)\right).$$

The root of the overestimation problem in Q-learning and deep Q-learning comes from the max operation when estimating the target Q-value at time $t$ as follows [47]:

$$H_t = r_t(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}),$$

where $\gamma$ is the discount factor indicating the importance of future rewards. To handle this issue, we adopt the deep double Q-learning algorithm [47] that leverages two identical deep dueling neural networks. One deep neural network is for action selection, namely Q-network $Q$, and the other is for action evaluation, namely target Q-network $\hat{Q}$. Then, the target Q-value is computed as follows:

$$H_t = r_t + \gamma \hat{Q} \left(s_{t+1}, \argmax_a Q(s_{t+1}, a; \phi_t); \phi_t\right),$$

where $\phi$ and $\phi^\prime$ are the parameters of the Q-network and target Q-network, respectively.

As the aim of training the Q-network is to minimize the gap between the target Q-value and the current estimated Q-value, the loss function at time $t$ is given as follows:

$$L_t(\phi_t) = E_{(s, a, r, s')} \left[ (H_t - Q(s, a; \phi_t))^2 \right],$$

where $E[.]$ is the expectation according to a data point $(s, a, r, s')$ in the buffer B. The loss function $L_t(\phi_t)$ can be minimized by the Gradient Descent (GD), which is one of the most popular algorithms for minimizing deep learning loss functions due to its simplicity in implementation [56]. However, GD requires calculating the gradient and cost function for all data points at each time step, yielding a very high processing time for large data.
To that end, we propose to use the Stochastic Gradient Descent (SGD) that only needs to compute gradients and cost function for a mini-batch data, i.e., $\mathcal{D}$, sampled uniformly from the buffer $B$. By doing so, SGD can accelerate the convergence rate while still guaranteeing the convergence of learning [57]. Specifically, the cost function of SGD is defined as follows:

$$J_t(\phi_t) = \frac{1}{|\mathcal{D}|} \sum_{(s,a,r,s') \in \mathcal{D}} \mathcal{L}_t(\phi_t).$$

(24)

At each step, Q-network’s parameters are updated by

$$\phi_{t+1} = \phi_t - \alpha_t \nabla_{\phi_t} J_t(\phi_t),$$

(25)

where $\nabla_{\phi_t}(\cdot)$ is the gradient of the cost function according to the current Q-network’s parameters $\phi_t$ and $\alpha_t$ is the step size controlling the rate of updating the Q-network’s parameters. It is worth mentioning that even though the target Q-value $H_t$ in (23) looks like labels that are used in the supervised learning, $H_t$ is not fixed before starting the learning process. Moreover, it changes at the same rate as that of the target Q-network’s parameters, possibly leading to instability in the learning process. To that end, instead of updating the parameters of $Q$ at every step, $\phi^-$ is only updated by copying from $\phi$ at every $U$ steps.

Note that the value of $\epsilon$ in the iMSAC controls the environment exploration. Specifically, the higher the value of $\epsilon$ is, the more frequently the agent takes a random action. The reason for decreasing $\epsilon$ stems from the fact that an RL agent does not have complete information about its environment in advance. As such, in the beginning, the agent should explore its environment by taking actions randomly. By doing so, it can obtain information about the environment via the feedback of its actions, e.g., corresponding rewards and next states. Then, the agent adjusts its policy according to these experiences, i.e., the state, action, reward, and the next state. Therefore, to converge to an optimal policy, the agent should take action based on its policy more frequently than the random action [33], leading to the decrease of $\epsilon$ at every step.

The computational complexity of the proposed iMSAC algorithm is mainly determined by the training phase of the Q-network. In this DNN, input and hidden layers have $Y$ and $L$ neurons, respectively. The output layer consists of the value stream with $B$ neurons and the advantage stream with $C$ neurons. Since training the Q-network at each iteration is implemented with matrix multiplication [58], the computational complexity of iMSAC for training with a data-point is $O(YL + LB + LC)$. Given the training phase takes $T$ iterations, each with a batch of experiences with size $Z$, the computational complexity of iMSAC is $O(TZ(YL + LB + LC))$.

Note that since an RL agent does not have entire knowledge about its surrounding environment in advance, it may not perform well at the beginning of the learning process. Therefore, to avoid this problem in practice, the agent (i.e., MetaSlicing’s Admission Controller) can be first trained in a simulated environment and then deployed in a practical system [59]. By doing so, the deployed agent in a real-time system starts the learning process with a reasonable policy, thereby improving the system’s performance. As discussed in the previous paragraph, the computational complexity of the proposed iMSAC is mainly determined by the training process of a deep neural network, i.e., Q-network that represents the admission policy. In addition, the Q-network only consists of conventional components, e.g., fully connected layers and tanh activation. Thus, the decision latency is very marginal since decisions can be made mostly instantly by feeding the state to the Q-network. In this work, simulations are performed on a typical laptop with the AMD Ryzen 3550H (4 cores at 2.1 GHz) and 16 GB RAM. We record that the average decision latency is 6.2 $\mu$s. In practice, several low latency-constrained applications have adopted DNN, such as Tesla Autopilot [60] and ALVINN [61], and thus they clearly demonstrate the applicability of DNN for real-time systems.

V. Performance Evaluation

A. Simulation Parameters

The parameters for our simulation are set as follows, unless otherwise stated. We consider that the system supports up to nine types of functions, i.e., $F = 9$. Each MetaSlice consists of three different functions and belongs to one of three classes, i.e., class-1, class-2, and class-3. In the configuration set $\mathcal{F}$, we set $K = 1$. Here, we set $\lambda_1, \lambda_2,$ and $\lambda_3$ to 60, 40, and 25 requests/hour, respectively, and its vector is denoted by $\lambda = [60, 40, 25]$. The average MetaSlice session time is 30 minutes, i.e., $\mu_i = 2, \forall i \in \{1, 2, 3\}$. The immediate reward $r_i$ for accepting a request from class-1, class-2, and class-3 are 1, 2, and 4, respectively. Note that our proposed algorithm, i.e., iMSAC, does not require the above information in advance. It can adjust the admission policy according to the practical requirements and demands (e.g., rental fees, arrival rate, and total resources of the system) to maximize a long-term average reward. Therefore, without loss of generality, the system has three types of resources, i.e., computing, storage, and radio bandwidth. Each MetaSlice is assumed to require a similar amount of resources as those of the Network Slice in 5G Network Slicing [62], e.g., 40 GB for storage, a bandwidth of 40 MHz, and 40 GFLOPS/s for computing.

In our proposed algorithm, i.e., iMSAC, the settings are as follows. For the $\epsilon$-greedy policy, the value of $\epsilon$ is gradually decreased from 1 to 0.01. The discount factor $\gamma$ is set to 0.9. We use Pytorch to build the Q-network and the target Q-network. They have the same architecture as shown in Fig. 2. During the learning process, typical hyperparameters of DNN are selected as those in [46] and [47], e.g., the learning rate of the Q-network is set at $10^{-3}$ and the target Q-network’s parameters are copied from the parameters of Q-network at every $10^4$ steps.

Recall that our proposed solution consists of two important elements, i.e., the intelligent algorithm iMSAC and MetaSlice analysis with the MetaInstance technique (MiT). With MetaInstance, functions can be reused, leading to a significant improvement in resource utilization. Meanwhile, the iMSAC can help the Admission Controller to obtain an optimal policy without requiring the complete information about the arrival and departure of MetaSlice in advance. Therefore, we compare our proposed solution, i.e., iMSAC+MiT, with three counterpart approaches: (i) iMSAC, (ii) Greedy policy [33] where the MetaSlicing’s Admission Controller accepts a request if the system has enough resources for the request, and (iii) Greedy policy with the
MetaInstance technique, i.e., Greedy+MiT. Recall that since the iMSAC approach does not leverage MetaInstance and is a RL-based resource management, it can be straightforwardly adopted to various scenarios, such as network slicing. Thus, performance of iMSAC can provide us insights on how existing admission control approaches perform in Metaverse.

B. Simulation Results

Simulations are conducted to gain insights into our proposed solution, i.e., iMSAC+MiT. First, we will investigate the convergence rate of our proposed algorithm iMSAC. Then, we evaluate the proposed solution in different scenarios to study impacts of important system parameters, e.g., the available system resources, immediate rewards reflecting the revenue of the MISP, and the maximum number of MetaSlices sharing one function that is one of the most important parameters of the MISP.

1) Convergence Rate: Fig. 4 shows the convergence rates of our proposed iMSAC algorithm in two scenarios with and without the MetaInstance technique. In this experiment, we set storage, radio bandwidth, and computing resources to 1200 GB, 1200 MHz, and 1200 GFLOPS/s, respectively. In other words, the system can support up to 30 functions in total. The average rewards obtained by Greedy+MiT and Greedy are also presented for comparisons. Specifically, the learning curves of iMSAC+MiT and iMSAC have a very similar trend. As shown in Fig. 4, both of them gradually converge to the optimal policy after $6 \times 10^4$ iterations. However, the iMSAC+MiT’s average reward is stable at 0.9, which is 2.25 times greater than that of the iMSAC. Similarly, the Greedy+MiT’s average reward is much greater (i.e., 3.5 times) than that of the Greedy. Thus, these results clearly show the benefits of the iMSAC and the MetaInstance technique. In particular, while the MetaInstance can help to maximize the resource utilization for the system, the iMSAC can make the Admission Controller learn the optimal policy to maximize the long term average reward.

2) Performance Evaluation: We now investigate the robustness of our proposed solution, i.e., iMSAC+MiT, in different scenarios. First, we vary the storage, radio, and computing resources from 400 GB, 400 MHz, and 400 GFLOPS/s to 2200 GB, 2200 MHz, and 2200 GFLOPS/s, respectively. In other words, the total number of functions supported by the system is varied from 10 to 55. The policies of iMSAC+MiT and iMSAC are obtained after $3.75 \times 10^5$ learning iterations. In this scenario, two metrics for evaluating the Admission Controller’s performance are average reward and acceptance probability since they clearly show the effectiveness of the admission policy in terms of the income for MISP (i.e., average reward) and the service availability for end-users. Fig. 5(a) clearly shows that as the total amount of system resources increases, the average rewards of all approaches increase. This is due to the fact that the higher the total resource is, the higher number of MetaSlices the system can host, and thus the greater revenue the system can achieve.

It is observed that the proposed algorithm iMSAC+MiT always obtains the highest average reward, up to 80% greater than that of the second-best policy in this scenario, i.e., Greedy+MiT. Similarly, Fig. 5(b) shows that iMSAC+MiT achieves the highest acceptance probability for an arriving MetaSlice request, up to 47% greater than that of the Greedy+MiT, i.e., the second-best policy. In addition, Fig. 5(a) and (b) demonstrate the benefit of MetaInstance. In particular, it helps the system to increase the average rewards and acceptance rates of both iMSAC and Greedy by up to 396% and 222%, respectively.

To gain more insights, we look further at the acceptance probability for each class of MetaSlice. As shown in Fig. 5(c) and (d), for the iMSAC+MiT and iMSAC, the acceptance probabilities of class-3 are always higher than that of other classes when the number of resources increases. Meanwhile, Greedy+MiT and Greedy accept requests from all the classes at almost the same probability, as depicted in Fig. 5(e), and (f). Recall that the arrival rate of class-3 is the lowest value (i.e., $\lambda_3=25$), while the immediate reward for accepting requests class-3 is the greatest value, i.e., $r_3=4$. Thus, the proposed algorithm iMSAC can learn and adjust its policy to achieve the best result. More interestingly, for the iMSAC+MiT results, the acceptance probability of class-3 has significant gaps (up to 50% greater than those of other classes) when the total resources are little (i.e., less than 20), as shown in Fig. 5. This stems from the fact that if the available resources are low, the Admission Controller should reserve resources for future requests from class-3 with the highest reward. In contrast, if the system has more available resources, the Admission Controller should accept requests from all classes more frequently. This observation is also shown in Fig. 5(d), where the MetaInstance is not employed.

Next, we investigate one of the most important factors in the MetaSlicing framework, which is the maximum number of MetaSlices that share the same function, denoted by $N_L$. In this experiment, we set the resources the same as those in Fig. 4, and other settings are set the same as those in Section V-A. Fig. 6(a) shows that as the value of $N_L$ increases from 1 to 15, the average rewards obtained by our proposed solution iMSAC+MiT and Greedy+MiT first increase and then stabilize at around 0.88. Remarkably, when the value of $N_L$ is small (i.e., less than 8), the iMSAC+MiT’s average reward is always greater than that of Greedy+MiT, up to 122%. The reason is that as $N_L$ increases, meaning that more MetaSlices can share the same function, the number of MetaSlices can be deployed.
Fig. 5. Vary the total number of system resources.

Fig. 6. Varying $N_L$.

in the system increases. In other words, the MetaSlicing system capacity increases according to the increase of $N_L$. As such, the Admission Controller can accept more requests to obtain greater rewards when $N_L$ increases. In addition, the thresholds in average rewards for both approaches originate from the fact that the arrival and departure processes of class $i$ follow the Poison and exponential distributions with a fixed mean $\lambda_i$ and $1/\mu_i$.

In terms of the acceptance probability for a MetaSlice request, similar observations can be made in Fig. 6(b). Specifically, our proposed solution maintains higher request acceptance probabilities (up to 34%) than that of the Greedy+MiT when $N_L$ is less than 8. As $N_L$ increases, the acceptance probabilities of both approaches increase, then they are stable at around 0.46 when $N_L > 4$ for iMSAC+MiT and $N_L > 7$ for Greedy+MiT. The reasons are similar as those in Fig. 6(a). In particular, the higher the system capacity is, the higher the request acceptance probability is. Unlike the above metrics, the average numbers of MetaInstances decrease for both approaches as the value of $N_L$ increases from 1 to 15, as shown in Fig. 6(c). The
The above observations in Figs. 5 and 6 show the superiority of our proposed approach compared with others, especially when the system resources are very limited.

We continue evaluating our proposed solution in the case where the immediate reward of class-3, i.e., \( r_3 \), is varied from 1 to 10. In this experiment, we set the storage, radio, and computing resources to 400 GB, 400 MHz, and 400 GFLOPS/s, respectively. The arrival rate vector of MetaSlice is set to \( \lambda = [60, 50, 40] \) to explore the robustness of our proposed solution.

In Fig. 7(a), as \( r_3 \) increases, the average rewards obtained by all approaches increase. In particular, the results demonstrate that our proposed solution, i.e., iMSAC+MiT, consistently achieves the highest average reward, up to 111% greater than that of the second-best, i.e., Greedy+MiT when \( r_3 = 1 \). Interestingly, when \( r_3 \) is small (i.e., less than 4), the iMSAC’s average rewards are lower than those of the Greedy+MiT. However, when \( r_3 \) becomes larger than or equal to 4, the average rewards obtained by iMSAC are higher than those of the Greedy+MiT.

Similarly, Fig. 7(b) and (c) show that our proposed solution always obtains the highest values compared to those of other approaches in terms of the acceptance probability and average number of running MetaSlices when \( r_3 \) increases from 1 to 10. Interestingly, even with a decrease in the acceptance probability and average number of running MetaSlices, the average rewards obtained by the iMSAC+MiT increase as \( r_3 \) is varied from 1 to 10, as shown in Fig. 7. The reason is that when the immediate reward of class-3 is very high (e.g., \( r_3 = 10 \)) compared to those of class-1 and class-2 (i.e., 1 and 2, respectively), the iMSAC+MiT reserves more resources for the future requests of class-3.

We now further investigate the above observations when varying the immediate reward of requests class-3 by looking deeper at the acceptance probability per class for each approach. Fig. 8(a)–(d) illustrate the acceptance rate per class according to the policies obtained by the proposed and counterpart approaches. In Fig. 8(c), the Greedy+MiT’s acceptance probabilities for all classes are almost the same, at around 0.06, when the immediate reward of class-3 increases from 1 to 10. A similar trend is observed for the Greedy but at a lower value, i.e., 0.04, in Fig. 8(d). In contrast to Greedy and Greedy+MiT, iMSAC+MiT’s acceptance probability for class-3 increases while those of other classes decrease as \( r_3 \) increases from one to 10, as shown in Fig. 8(a). More interestingly, when the immediate reward of class-3 requests is small (i.e., \( r_3 < 2 \)), class-3 requests have the lowest acceptance probability compared to those of other classes. However, when the immediate reward of class-3
requests is larger than or equal to 2, class-3 requests will achieve the highest acceptance probability compared with those of other classes. Moreover, when $r_3 > 4$, the acceptance probability for class-3 requests obtained by the iMSAC+MiT is stable at around 0.16.

Similar to the iMSAC+MiT, the iMSAC’s acceptance probability for class-3 requests also increases until reaching a threshold with a lower value, i.e., around 0.14, compared to that of the iMSAC+MiT. Furthermore, the acceptance probability of class-3 requests obtained by the iMSAC-base solutions is up to four-time greater than those of the Greedy-based solutions. Thus, the iMSAC+MiT and iMSAC can obtain a good policy in which the acceptance probability of a class increases if its reward increases compared with the rewards of other classes, and vice versa. Note that our proposed solution does not need complete information about the MetaSlice’s arrival and departure processes in advance. However, as observed, the proposed solution can always achieve the best results in all scenarios when we vary important parameters of the system.

VI. CONCLUSION

In this paper, we have proposed two innovative techniques, i.e., the application decomposition and the MetaInstance, to maximize resource utilization for the Metaverse built on a multi-tier computing architecture. Based on these techniques, we have developed a novel framework for the Metaverse, i.e., MetaSlicing, that can smartly allocate resources for MetaSlices, i.e., Metaverse applications, to maximize the system performance. Moreover, we have proposed a highly effective framework based on sMDP together with an intelligent algorithm, i.e., iMSAC, to find the optimal admission policy for the Admission Controller under the high dynamics and uncertainty of resource demands. The extensive simulation results have clearly demonstrated the robustness and superiority of our proposed solution compared with the counterpart methods as well as revealed key factors determining the system performance. As Metaverse continues to evolve and expand, there will be enormous potential research topics that we can explore. For example, elastic resource allocation approaches can be developed to further enhance the efficiency of using MetaSlices. In particular, allocated resources of a MetaSlice can be scaled dynamically according to the resource demand of this MetaSlice during operation, thus further improving resource utilization. Another topic that we can further consider is methods for ensuring interoperability between different MetaSlices in the Metaverse. This could include developing standards and protocols for communication between different MetaSlices and mechanisms for managing conflicts and ensuring consistency between MetaSlices.

REFERENCES

[1] N. Stephenson, Snow Crash, New York, NY, USA: Bantam Books, 1992.
[2] Introducing meta: A social technology company, 2021. Accessed: May 01, 2022. [Online]. Available: https://about.fb.com/news/2021/10/facebook-company-is-now-meta/
[3] Mesh for Microsoft teams aims to make collaboration in the ‘Metaverse’ personal and fun, 2021. Accessed: May 01, 2022. [Online]. Available: https://news.microsoft.com/innovation-stories/mesh-for-microsoft-teams/
[4] M. Xu et al., “A full dive into realizing the edge-enabled metaverse: Visions, enabling technologies, and challenges,” 2022, arXiv:2203.05471.
[5] MetaQuest. Horizon Worlds, Dec. 10, 2021. Accessed: May 01, 2022. [Online]. Available: https://www.youtube.com/watch?v=02kEUMRwq&d=64
[6] Y. Wu, K. Zhang, and Y. Zhang, “Digital twin networks: A survey,” IEEE Internet Things J., vol. 8, no. 18, pp. 13789–13804, Sep. 2021.
[7] X. Yu, D. Owens, and D. Khazanchi, “Building socioenvironmental environments in metaverses for virtual teams in healthcare: A conceptual exploration,” in Proc. Int. Conf. Health Inf. Sci., 2012, pp. 4–12.
[8] C. Kwon, “Smart city-based metaverse: a study on the solution of urban problems,” J. Chosun Natural Sci., vol. 14, no. 1, pp. 21–26, Mar. 2021.
[9] H. Jeong, Y. Yi, and D. Kim, “An innovative e-commerce platform incorporating metaverse to live commerce,” Int. J. Innov. Comput. Inf. Control, vol. 18, no. 1, pp. 221–229, Feb. 2022.
[10] J. Clement, “Most played games on steam 2021, by peak player count,” 2022. Accessed: May 01, 2022. [Online]. Available: https://www.statista.com/statistics/656278/steam-most-played-games-peak-concurrent-player/
[11] Metaverse: A guide to the next-gen internet. 2022. Accessed: May 01, 2022. [Online]. Available: https://www.credit-suisse.com/media/assets/creative/docs/about-us/media/media-release/2022/03/metaverse-14032022.pdf
[12] X. Wang, T. Kwon, Y. Choi, M. Chen, and Y. Zhang, “Characterizing the gaming traffic of World of Warcraft: From game scenarios to network access technologies,” IEEE Netw., vol. 26, no. 1, pp. 27–34, Jan./Feb. 2012.
[13] J. Zhao, R. S. Allison, M. Vinnikov, and S. Jennings, “Evaluating the motion-to-photon latency in head mounted displays,” in Proc. IEEE Virtual Reality, 2017, pp. 313–314.
[14] Y. Liang, J. Kang, D. Niyato, X. Ge, Z. Xiong, and C. Miao, “Reliable coded distributed computing for metaverse services: Coalition formation and incentive mechanism design,” 2021, arXiv: 2111.10548.
[15] M. Xu, D. Niyato, J. Kang, Z. Xiong, C. Miao, and D. I. Kim, “Wireless edge-empowered metaverse: A learning-based incentive mechanism for virtual reality,” 2021, arXiv: 2111.03776.
[16] W. C. Ng, W. Y. B. Lim, J. S. Ng, Z. Xiong, D. Niyato, and C. Miao, “Unified resource allocation framework for the edge intelligence-enabled metaverse,” 2021, arXiv: 2110.14325.
[17] Y. Han, D. Niyato, C. Leung, C. Miao, and D. I. Kim, “A dynamic resource allocation framework for synchronizing metaverse with IoT service and data,” 2021, arXiv: 2111.00431.
[18] K. Cao, Y. Liu, G. Meng, and Q. Sun, “An overview on edge computing research,” IEEE Access, vol. 8, pp. 85714–85728, May 2020.
[19] Google Maps platform documentation. Accessed: May 01, 2022. [Online]. Available: https://developers.google.com/maps/documentation
[20] I. Singh. “Your next favorite AR game will be built on Google Maps,” 2018. Accessed: May 01, 2022. [Online]. Available: https://geowondermap.com/google-maps-will-now-power-location-aware-augmented-reality-games/
[21] S. Hayden, “‘Woorld’ is a multiplayer version of Google Earth VR’ for quest, releasing soon,” 2022. Accessed: May 01, 2022. [Online]. Available: https://www.roadtovr.com/woorld-google-earth-vr-quest-2/
[22] N. Francis, “Collection of the coolest uses of the Google Maps API,” 2022. Accessed: May 01, 2022. [Online]. Available: https://www.jotform.com/blog/collection-of-the-coolest-uses-of-the-google-maps-api/
[23] S. Sahaf et al., “Network service chaining with optimized network function embedding supporting service compositions,” Comput. Netw., vol. 93, pp. 492–505, Dec. 2015.
[24] B. Atturki, S. Reiff-Marganiec, C. Perera, and S. De, “Exploring the effectiveness of service decomposition in fog computing architecture for the Internet of Things,” IEEE Trans. Sustain. Comput., vol. 7, no. 2, pp. 299–312, Second Quarter 2022.
[25] A. Wolke, M. Bichler, and T. SIES, “Planning vs. dynamic control: Resource allocation in corporate clouds,” IEEE Trans. Cloud Comput., vol. 4, no. 3, pp. 322–335, Third Quarter 2016.
[26] S. Abbassolou, Y. Xu, and H. J. Chao, “2TCP: A flexible cellular TCP to meet stringent delay requirements,” IEEE J. Sel. Areas Commun., vol. 37, no. 4, pp. 918–932, Apr. 2019.
[27] Network functions virtualisation: An introduction, benefits, enablers, challenges & call for action,” ESTI, 2012. Accessed: Apr. 24, 2022. [Online]. Available: https://docbox.etsi.org/isg/nfv/open/Publications_pdf/White_Papers/NFV_White_Paper1_2012.pdf
[28] Description of network slicing concept, NGMN Alliance. 2016. Accessed: Apr. 24, 2022. [Online]. Available: https://www.ngmn.org/wp-content/uploads/Publications/2016/161010_NGMN_Network_Slicing_framework_v1.0.8.pdf

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
[29] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, “Network slicing and softwarization: A survey on principles, enabling technologies, and solutions,” IEEE Commun. Surveys Tut., vol. 20, no. 3, pp. 2429–2453, Third Quarter 2018.

[30] A. Laghribi and T. Taleb, “A survey on the placement of virtual resources and virtual network functions,” IEEE Commun. Surveys Tut., vol. 21, no. 2, pp. 1409–1434, Second Quarter 2019.

[31] H. Zhang et al., “Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges,” IEEE Commun. Mag., vol. 55, no. 8, pp. 138–145, Aug. 2017.

[32] H. C. Tijms, A First Course in Stochastic Models. Hoboken, NJ, USA: Wiley, 2003.

[33] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 2018.

[34] R. G. Gallager, Discrete Stochastic Processes. London, U.K.: Kluwer, 1995.

[35] L. Kallenberg, Markov Decision Processes, University of Leiden, 2016. Accessed: May 01, 2022. [Online]. Available: http://www.math.leidenuniv.nl/%7Ekallenberg/Lecture-notes-MDP.pdf

[36] G. Wang, G. Feng, W. Tan, S. Qin, R. Wen, and S. Sun, “Resource allocation for network slices in 5G with network resource pricing,” in Proc. IEEE Glob. Commun. Conf., 2017, pp. 1–6.

[37] H. Roh, C. Jung, W. Lee, and D.-Z. Du, “Resource pricing game in geo-distributed clouds,” in Proc. IEEE Conf. Comput. Commun., 2013, pp. 1519–1527.

[38] Pricing-cloud services | Microsoft azure. Accessed: Apr. 20, 2023. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/cloud-services/

[39] Google compute engine pricing. Accessed: Apr. 20, 2023. [Online]. Available: https://cloud.google.com/compute/pricing

[40] W. Wu et al., “Dynamic RAN slicing for service-oriented vehicular networks via constrained learning,” IEEE J. Sel. Areas Commun., vol. 39, no. 7, pp. 2076–2089, Jul. 2021.

[41] X. Shen et al., “AI-assisted network-slicing based next-generation wireless networks,” IEEE Open J. Veh. Technol., vol. 1, pp. 45–66, Jan. 2020.

[42] W. Wu et al., “AI-native network slicing for 6G networks,” IEEE Wireless Commun., vol. 29, no. 1, pp. 96–103, Feb. 2022.

[43] P. Jaccard, “The distribution of the flora in the alpine zone,” New Phytologist, vol. 11, no. 2, pp. 37–50, Feb. 1912.

[44] C. J. Watkins and P. Dayan, “Q-learning,” Mach. Learn., vol. 8, no. 3, pp. 279–292, May 1992.

[45] N. C. Luong et al., “Applications of deep reinforcement learning in communications and networking: A survey,” IEEE Commun. Surveys Tut., vol. 21, no. 4, pp. 3133–3174, Fourth Quarter 2019.

[46] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[47] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” in Proc. AAAI Conf. Artif. Intell., 2016, pp. 2094–2100.

[48] S. Thrun and A. Schwartz, “Issues in using function approximation for reinforcement learning,” in Proc. 33rd Int. Conf. Mach. Learn., 2016, pp. 1995–2003.

[49] R. G. Gallager, “The distribution of the flora in the alpine zone,” New Phytologist, vol. 11, no. 2, pp. 37–50, Feb. 1912.

[50] M. Hessel et al., “Rainbow: Combining improvements in deep reinforcement learning,” J. Mach. Learn., vol. 65, pp. 452–468, Oct. 2022.

[51] L. Engstrom et al., “Implementation matters in deep policy gradients: A case study on PPO and TRPO,” in Proc. Int. Conf. Learn. Representations, 2020, pp. 1–14.

[52] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, C. Lanctot, and N. Freitas, “Deep learning.”

[53] S. Halkjær and O. Winther, “The effect of correlated input data on the dynamics of learning,” in Proc. 9th Int. Conf. Neural Inf. Process. Syst., 1996, pp. 169–175.

[54] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, “Gradient descent finds global minima of deep neural networks,” in Proc. Int. Conf. Mach. Learn., 2019, pp. 1675–1685.

[55] H. Robbins and S. Monro, “A stochastic approximation method,” Ann. Math. Statist., vol. 22, pp. 400–407, Sept. 1951.

[56] S. Halkjær and O. Winther, “The effect of correlated input data on the dynamics of learning,” in Proc. 9th Int. Conf. Neural Inf. Process. Syst., 1996, pp. 169–175.

[57] H. Robbins and S. Monro, “A stochastic approximation method,” Ann. Math. Statist., vol. 22, pp. 400–407, Sept. 1951.

[58] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[59] C. T. Nguyen et al., “Transfer learning for wireless networks: A comprehensive survey,” Proc. IEEE, vol. 110, no. 8, pp. 1073–1115, Aug. 2022.

[60] Artificial Intelligence: A Human Affairs. Accessed: Apr. 20, 2023. [Online]. Available: https://www.tesla.com/en/AU/AI

[61] D. A. Pomelraive, “ALVINN: An autonomous land vehicle in a neural network,” in Proc. Adv. Neural Inf. Process. Syst., D. Tourtetzy, Ed., 1988, pp. 305–313.

[62] G. Dandachi, A. De Domenico, D. T. Hoang, and N. Niyato, “An artificial intelligence framework for slice deployment and orchestration in 5G networks,” IEEE Trans. Cogn. Commun. Netw., vol. 6, no. 2, pp. 858–871, Jun. 2020.

[63] M. L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming. Hoboken, NJ, USA: Wiley, 1994.

Nam H. Chu received the BE degree in electronics and telecommunications engineering from the Hanoi University of Science and Technology, Vietnam, in 2009, and the master’s degree in software engineering from the University of Canberra, Australia, in 2014. He is currently working toward the PhD degree with the University of Technology Sydney, Sydney, NSW, Australia. He is a faculty member of the Department of Telecommunications Engineering, University of Transport and Communications, Hanoi, Vietnam, and a research assistant with the School of Engineering and Mathematical Sciences, Department of Computer Science and Information Technology, La Trobe University, Melbourne, Australia. His research interests include applying machine learning and optimization methods for wireless communications.

Diep N. Nguyen (Senior Member, IEEE) received the ME degree in electrical and computer engineering from the University of California at San Diego (UCSD), La Jolla, California, in 2008, and the PhD degree in electrical and computer engineering from The University of Arizona (UA), Tucson, Arizona, in 2013. He is currently a faculty member with the Faculty of Engineering and Information Technology, University of Technology Sydney (UTS), Sydney, NSW, Australia. Before joining UTS, he was a DECRA research fellow with Macquarie University, Macquarie Park, NSW, Australia, and a member of the Technical Staff with Broadcom Corporation, San Jose, California, and ARCON Corporation, Boston, Massachusetts, and consulting the Federal Administration of Aviation, Washington, DC, on turning detection of UAVs and aircraft, and the U.S. Air Force Research Laboratory, Wright-Patterson Air Force Base, Ohio, on anti-jamming. His research interests include computer networking, wireless communications, and machine learning applications, with emphasis on systems’ performance and security/privacy. He received several awards from LG Electronics, UCSD, UA, the U.S. National Science Foundation, and the Australian Research Council. He has served on the editorial boards of IEEE Transactions on Mobile Computing, IEEE Communications Surveys and Tutorials (COMST), IEEE Open Journal of the Communications Society, and Scientific Reports (Nature’s).
Khoa T. Phan received the BEng degree in telecommunications (First Class Hons.) from the University of New South Wales (UNSW), Sydney, NSW, Australia, in 2006, the MSc degree in electrical engineering from the University of Alberta, Edmonton, AB, Canada, in 2008, and California Institute of Technology (Caltech), Pasadena, California, in 2009, respectively, and the PhD degree in electrical engineering from McGill University, Montreal, QC, Canada, in 2017. He is currently a senior lecturer and Australia Research Council (ARC) Discovery Early Career Researcher Award (DECRA) fellow with the Department of Computer Science and Information Technology, La Trobe University, Victoria, Australia. His research interests are broadly design, control, optimization, and security of next-generation communications networks with applications in the Internet of Things (IoT), satellite systems, and cloud computing. He is keen on applying machine learning tools, such as deep learning, federated learning into designing intelligent secure cyber physical systems.

Eryk Dutkiewicz (Senior Member, IEEE) received the BE degree in electrical and electronic engineering from the University of Adelaide, in 1988, the MSc degree in applied mathematics from the University of Adelaide, in 1992, and the PhD degree in telecommunications from the University of Wollongong, in 1996. His industry experience includes management of the Wireless Research Laboratory at Motorola in early 2000’s. He is currently the head of the School of Electrical and Data Engineering, University of Technology Sydney, Australia. His current research interests cover 5G/6G and IoT networks.

Dusit Niyato (Fellow, IEEE) received the BEng degree from the King Mongkuts Institute of Technology Ladkrabang (KMITL), Thailand, in 1999 and the PhD degree in electrical and computer engineering from the University of Manitoba, Canada, in 2008. He is a professor with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests are in the areas of sustainability, edge intelligence, decentralized machine learning, and incentive mechanism design.

Tao Shu received the BS and MS degrees in electronic engineering from the South China University of Technology, Guangzhou, China, in 1996 and 1999, respectively, the PhD degree in communication and information systems from Tsinghua University, Beijing, China, in 2003, and the PhD degree in electrical and computer engineering from The University of Arizona, in December 2010. He is currently an Associate Professor with the Department of Computer Science and Software Engineering, Auburn University. Before joining Auburn, He has worked as an Assistant Professor for 5 years in the Computer Science and Engineering Department of Oakland University in Rochester, Michigan. Prior to Oakland, He was a senior engineer in Qualcomm Atheros Inc. from December 2010 to August 2011.