Slicing for AI

An Online Learning Framework for Network Slicing Supporting AI Services

Helmy, M.; Abdellatif, A. A.; Mhaisen, N.; Mohamed, A.; Erbad, A.

DOI
10.1109/TNSM.2025.3603391

Publication date
2025

Document Version
Final published version

Published in
IEEE Transactions on Network and Service Management

Citation (APA)
Helmy, M., Abdellatif, A. A., Mhaisen, N., Mohamed, A., & Erbad, A. (2025). Slicing for AI: An Online Learning Framework for Network Slicing Supporting AI Services. *IEEE Transactions on Network and Service Management*, *22*(6), 5239-5254. https://doi.org/10.1109/TNSM.2025.3603391

Slicing for AI: An Online Learning Framework for Network Slicing Supporting AI Services

Menna Helmy, Alaa Awad Abdellatif, *Senior Member, IEEE*, Naram Mhaisen, *Graduate Student Member, IEEE*, Amr Mohamed, *Senior Member, IEEE*, and Aiman Erbad, *Senior Member, IEEE*

***Abstract*—The forthcoming 6G networks will embrace a new realm of AI-driven services that requires innovative network slicing strategies, namely *slicing for AI*, which involves the creation of customized network slices to meet Quality of Service (QoS) requirements of diverse AI services. This poses challenges due to time-varying dynamics of users' behavior and mobile networks. Thus, this paper proposes an online learning framework to determine the allocation of computational and communication resources to AI services, to optimize their accuracy as one of their unique key performance indicators (KPIs), while abiding by resources, learning latency, and cost constraints. We define a problem of optimizing the total accuracy while balancing conflicting KPIs, prove its NP-hardness, and propose an online learning framework for solving it in dynamic environments. We present a basic online solution and two variations employing a pre-learning elimination method for reducing the decision space to expedite the learning. Furthermore, we propose a biased decision space subset selection by incorporating prior knowledge to enhance the learning speed without compromising performance and present two alternatives of handling the selected subset. Our results depict the efficiency of the proposed solutions in converging to the optimal decisions, while reducing decision space and improving time complexity. Additionally, our solution outperforms State-of-the-Art techniques in adapting to diverse environmental dynamics and excels under varying levels of resource availability.**

***Index Terms*—Network slicing, online learning, resource allocation, 6G networks, optimization.**

I. Introduction

IT IS anticipated that the 6G networks will have the capability to cater to an array of services, each with distinct Quality of Service (QoS) specifications. Such services may include multisensory extended reality, autonomous driving, and hologram video streaming. To ensure diversity in services, similar to the 5G networks, network slicing is employed to create multiple slices for various services over a shared physical network infrastructure. An economical management strategy for network slicing can enable the fulfillment of QoS requirements throughout the different phases of the lifecycle, such as preparation, planning, and scheduling [1]. Additionally, newly emerging technologies for 6G networks such as the Open Radio Access Network (O-RAN) architecture will allow the integration of native Artificial intelligence (AI) solutions to accommodate heterogeneous service deployments [2], [3], [4]. In this context, AI will become omnipresent in 6G networks, meaning that it will penetrate every aspect of the network, creating a state of ubiquitous intelligence. Network nodes will possess in-built AI capabilities, not only allowing for intelligent network management but also promoting the growth of AI-based services, such as machine learning, natural language processing, and computer vision.

To fulfil diverse requirements of different AI services, it is important to implement customized network slices, namely *slicing for AI*. This approach allows for the creation of tailored network slices that can cater to the distinct requirements of various AI services (i.e., accuracy, learning speed, etc.), thereby optimizing the network's overall performance [5]. Indeed, slicing for AI refers to the creation of network slices with customized resources to meet QoS requirements of diverse AI services. Hence, it can play a crucial role in the process of dynamically distributing and assigning resources in meta learning systems by allocating resources such as computation power, memory, and data samples efficiently and effectively to improve the learning process and overall performance of the meta learning algorithms. However, this problem is challenging due to the dynamics of diverse learning algorithms and mobile networks. For example, the quality and distribution of acquired data are time-varying and heavily affect the performance of AI services. Hence, novel solutions are needed to adapt to such non-stationary dynamics. Thus, this paper aims to optimize the decision-making process by proposing online learning techniques [6] that continuously observe the system's performance without prior knowledge of the expected behavior.

Several works have addressed the problem of network slicing in 5G and beyond networks to support the heterogeneous requirements of various conventional vertical services, such as ultra-reliable low-latency communication (URLLC), enhanced mobile Broadband (eMBB), and massive

Received 7 July 2023; revised 3 January 2025; accepted 11 August 2025. Date of publication 28 August 2025; date of current version 5 December 2025. Research reported in this publication was supported by the Qatar Research Development and Innovation Council ARG01-0527-230356. The associate editor coordinating the review of this article and approving it for publication was M. J. Khabbaz. *(Corresponding author: Alaa Awad Abdellatif.)*

Menna Helmy, Amr Mohamed, and Aiman Erbad are with the College of Engineering, Qatar University, Doha, Qatar (e-mail: mh1800882@qu.edu.qa; amrm@qu.edu.qa; aerbad@ieee.o).

Alaa Awad Abdellatif is with the Center for Telecommunications and Multimedia, INESC TEC, 4200-465 Porto, Portugal (e-mail: alaa.abdellatif@ieee.org).

Naram Mhaisen is with the College of Electrical Engineering, Mathematics, and Computer Science, TU Delft, 2600 AA Delft, The Netherlands.

Digital Object Identifier 10.1109/TNSM.2025.3603391

machine-type communication (mMTC) [7], [8], [9], [10], [11]. However, few works considered the problem of *slicing for AI*. Most of the proposed solutions in this context leveraged offline classical optimization and reinforcement learning (RL) techniques. Classical optimization approaches assume full knowledge about the behavior of the environment while leveraging mathematical formulations to model different objectives and constraints. However, such an assumption may be impractical with some services and mobile network systems, which are highly dynamic and time-variant. On the contrary, RL approaches aim to learn a policy in a stateful system by mapping between states and actions. These approaches consider an environment that is stationary (i.e., behaving according to a non-varying distribution). Also, classical optimization and the training phase of RL are usually performed in an offline manner on an available dataset/environment. Nonetheless, an offline approach may fail in adapting to dynamic/time-varying systems. Thus, in this paper, we propose an online learning framework to address the problem of *slicing for AI*. The proposed framework can adapt to different system dynamics and uncertainty with regret guarantees. The main contributions of this paper are as follows:

- We formulate the problem of *slicing for AI* services as an optimization problem with the objective of maximizing performance, which turns out to be NP-hard.
- We introduce an online learning framework and propose a basic online solution to solve the formulated problem. We propose two alternatives of the solution where each adopts a pre-learning decision space reduction process to expedite the learning. The first alternative merges similar decisions to obtain a compact structure of the decision space, while the second builds upon the obtained compact structure and identifies candidates of the optimal decision to optimize the size of the action space.
- We propose a subset selection of the original decision space with prior knowledge to accelerate the convergence of the solution. We consider two alternative approaches of manipulating the solution by biasing the selected subset.
- We assess the solution's performance by comparing it to optimal allocation and fixed allocation benchmarks, demonstrating its ability to converge towards optimal resource allocation. We examine the complexity and convergence of the proposed alternative online solutions. Furthermore, we compare between the trade-offs of the two biased subset selection approaches. Additionally, we assess the adaptability of our solution under diverse environmental dynamics, comparing its performance to State-of-the-Art approaches while also accounting for varying levels of resource availability.

The rest of the paper is organized as follows. We present some related work in Section II. Section III introduces the system model and the formulation of the problem as an optimization problem. Section IV presents the proposed solutions and modelling the problem into an online learning framework. Section V evaluates the performance of the solution. Finally, we conclude the paper in Section VI.

II. Related Work

In this section, we review the existing literature on network slicing solutions for 5G and beyond, focusing on diverse approaches such as optimization, RL, online learning, and hybrid methodologies that combine multiple techniques [12].

Optimization-based solutions: Several studies have tackled the issue of network slicing and resource allocation using traditional optimization techniques [9], [13], [14], [15], [16], [17], [18]. For instance, the authors in [9] addressed the problem of allocating base-band resources for diverse slices in an O-RAN system, while considering the diverse requirements of eMBB, URLLC, and mMTC services. They formulated this problem as a mixed-integer non-linear programming problem and proposed a service-aware solution through a two-step iterative heuristic algorithm. Similarly, in [15], the authors modeled their resource allocation problem as a mixed-integer non-linear programming problem and suggested approximation-based methods. The authors of [16] introduced an Integer Linear Programming algorithm to solve their formulated slicing problem. Optimization-based solutions rely on precise mathematical models and, although capable of achieving optimal or near-optimal solutions, they often face practical limitations due to the dynamic nature of mobile networks. Consequently, any changes in the network environment necessitate re-executing the optimization process to obtain new optimal solutions. In some cases, even the underlying mathematical model requires recalibration, resulting in computational overhead and hinders the practicality of the solution.

RL-based solutions: To overcome the limitations of classical optimization solutions, emerging studies have employed classical RL [10], [19], [20], [21] and Deep RL (DRL) techniques for network slicing utilizing deep learning [7], [8], [22], [23], [24], [25], [26], [27], [28], [29]. For instance, the authors in [10] introduced a multi-agent reinforcement learning (MARL) framework called VERA, which leverages SARSA to achieve a fair and Pareto-efficient allocation of computational and network resources among competing users' applications and virtual RAN (vRAN) services at the edge. The work in [19] presented a MARL framework employing Q-Learning for radio resource slicing in 5G networks. Additionally, [20] explored three distinct solutions using the Upper Confidence Bound (UCB) algorithm for network slicing in LoRaWAN networks. Furthermore, [21] utilized Thompson Sampling (TS) to optimize RAN slicing, aiming to minimize energy consumption while enhancing user quality of service. Other works, such as [8], [23], addressed the problem of RAN slicing in 6G networks by proposing a Federated DRL (FDRL) and a DRL based solution respectively. The authors in [25] proposed a DRL solution to the problem of resource allocation of eMBB and URLLC slices, subject to URLLC delay requirements and eMBB data rate requirements. Other network slicing problems such as RAN slice admission and placement, and slice reconfiguration has been addressed by [26], [27] with a DRL-based solution.

Although RL-based solutions have the advantage of adaptability to stationary system dynamics and the ability to

maximize system rewards through policy learning, they often assume a non-varying stochastic environment, which can pose challenges when attempting to adapt to unforeseen changes in the system.

Online learning-based solutions: In contrast to classical optimization and RL methods, online learning approaches offer the advantage of adaptability to non-stationary system dynamics and unforeseen events. However, only a limited number of studies have applied online learning in the context of network slicing [7], [30]. For instance, the work in [7] incorporated online learning in solving its formulated Software-Defined Network (SDN) radio resource allocation problem. The objective of this work was to maximize the total achievable data rate of eMBB and URLLC end-users associated with all gNodeBs while adapting to user dynamics. The work in [30] addressed the problem of resource reservation by service providers, which is closely related to network slicing problems. The authors proposed a framework based on online convex optimization for effective resource reservation while maximizing the services' performance, constrained to a time-average budget constraint, considering a time-varying service demand and slice pricing.

Slicing for AI: While there have been several studies focusing on network slicing for conventional services, the number of works addressing the network slicing problem for AI-based services is comparatively limited [13], [14], [15], [16]. For example, the authors in [13] tackled the challenge of joint bandwidth allocation and user scheduling for federated edge learning. Their objective was to reduce energy consumption while ensuring a guaranteed learning speed. Additionally, in [14], the authors addressed the resource allocation problem for distributed machine learning (ML) application considering both edge and cloud resources in terms of computational performance and cost. In [16], the authors focused on joint AI service placement and resource allocation in mobile edge system, while minimizing computation time and energy consumption. While these works contribute to resource allocation for AI services, none of them consider the joint allocation of resources and tuning of the underlying AI models' variables, such as the learning parameters (e.g., number of epochs) and training data size.

Novelty. Our work is the first to tackle the challenges of *slicing for AI* through the utilization of online learning techniques. The proposed framework aims to optimize the performance of various AI services while effectively adapting to dynamic system conditions and unexpected events. Moreover, unlike other studies, we consider the joint allocation of computational and communication resources for different AI models along with hyper-parameter tuning of these models.

III. System Model and Problem Formulation

In this section, we describe the considered system architecture, highlight the main functions constituting an AI service, discuss the key performance metrics that will be tackled for different AI services, and present the formulated accuracy maximization problem.

TABLE I
KEY NOTATIONS

Notation	Description
$\mathcal{I}$	The set of AI models
$\mathcal{T}$	The set of time slots
$L_i^{(t)}$	The data size in samples required at the t-th slot for training AI model $i \in \mathcal{I}$
$L_{\min_i}$	The Minimum allowable data size for training AI model $i \in \mathcal{I}$
$L_{\max_i}$	The Maximum allowable data size for training AI model $i \in \mathcal{I}$
$l_i^{(t)}$	The data size in percentage required at the t-th slot for training AI model $i \in \mathcal{I}$
$m_i^{(t)}$	The number of epochs required at the t-th slot for training AI model $i \in \mathcal{I}$
$m_{\min_i}$	The Minimum allowable number of epochs for training AI model $i \in \mathcal{I}$
$m_{\max_i}$	The Maximum allowable number of epochs for training AI model $i \in \mathcal{I}$
$\psi_i^{(t)}$	The computing resources allocated at the t-th slot for training AI model $i \in \mathcal{I}$
ϕ_i	The number of CPU cycles required by model $i \in \mathcal{I}$ to compute one sample of data
$\Psi_{\max}$	The maximum available computing resources
$\lambda_i^{(t)}$	The communication resources allocated at the t-th slot for training AI model $i \in \mathcal{I}$
$\Lambda_{\max}$	The maximum available communication resources
$D_{\max_i}$	The maximum learning latency requested by the service corresponding to AI model $i \in \mathcal{I}$
$C_{\max_i}$	The maximum cost set by the service corresponding to AI model $i \in \mathcal{I}$
c_ψ	The cost per computing unit
c_λ	The cost per unit of data rate
$q_i^{(t)}(\cdot)$	A function representing the learning accuracy of AI model $i \in \mathcal{I}$ at the t-th time slot
α_i	The priority of the service corresponding to AI model $i \in \mathcal{I}$
$\mathcal{O}$	The set of all possible hyper-parameter combinations, where $\boldsymbol{o} \in \mathcal{O} = (L_1, m_1, L_2, m_2, ..., L_I, m_I)$
$\mathcal{S}$	The set of all possible resource combinations, where $\boldsymbol{s} \in \mathcal{S} = (\psi_1, \lambda_1, \psi_2, \lambda_2, ..., \psi_I, \lambda_I)$
$\boldsymbol{a}^{(t)}$	Allocation decision vector selected at the t-th slot representing a combination of $\boldsymbol{o} \in \mathcal{O}$ with $\boldsymbol{s} \in \mathcal{S}$
$\mathcal{B}_{\mathrm{SA}_{\boldsymbol{o}}}$	A super action defined as a set of grouped allocation decisions corresponding to the same $\boldsymbol{o} \in \mathcal{O}$ with different $\boldsymbol{s} \in \mathcal{S}$

A. System Architecture

We denote sets with calligraphic capital letters, e.g., $\mathcal{A}$, and vectors with bold small letters, e.g., $\boldsymbol{a}$. We use the superscript (t) to indicate the dependence on a time slot, e.g., $\boldsymbol{a}^{(t)}$, and subscripts to represent the correspondence to an indexed element. For example, to denote the j-th element of the set $\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_J\}$ we write $\boldsymbol{a}_j$, and to denote a parameter that corresponds to the i-th element of $\boldsymbol{a}_j$ we write $L_{i,j}$. To represent a sequence of vectors from slot $t = 1$ to slot T we write $\{\boldsymbol{a}^{(t)}\}_{t=1}^{T}$. We present the key notation in Table I.

The considered system model is depicted in Figure 1, which is adapted from [10] and aligns with the O-RAN framework

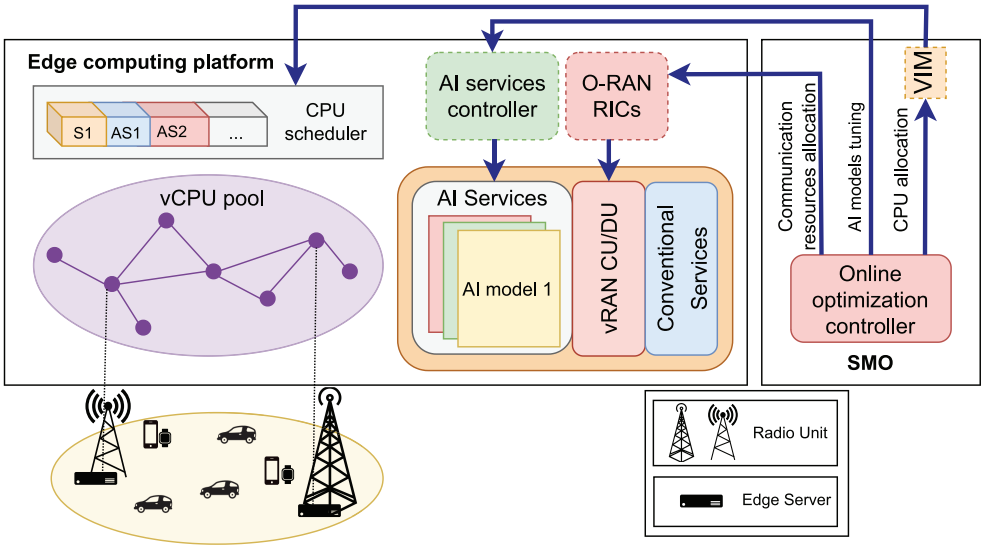


Fig. 1. System model under study.

specifications [2]. The system supports conventional services, such as enhanced mobile broadband (eMBB) and ultra-reliable low-latency communication (URLLC), AI services, such as pandemic detection and object detection for autonomous driving services, and virtualized RAN (vRAN) functions, such as central unit (CU) and distributed unit (DU). Such services and functions coexist within an edge computing platform and compete over a shared pool of virtualized CPU (vCPU) resources (i.e., computing resources) managed by a CPU scheduler, as well as communication resources. In this work, we specifically focus on slicing for AI services, assuming long-term isolation between AI services and other types of services. We consider the joint allocation of virtual resources to AI services represented by a set of underlying AI models, $\mathcal{I} = \{1, 2, 3, \ldots, I\}$, and the tuning of some of the AI models parameters to perform AI model training. The system is time-slotted and consists of T time slots, $t \in \mathcal{T} = \{1, 2, \ldots, T\}$. The online optimization controller running within the Service Management & Orchestration (SMO) platform, represents a learner whose function is to control the allocation of resources and the tuning of the AI models parameters. The learner allocates computing and communication resources to the AI services through interaction with the virtual infrastructure manager (VIM), and the O-RAN intelligent controllers (RICs) respectively. It also tunes the AI models parameters through interacting with the AI services controllers. Specifically, the learner selects a sequence of actions $\{\boldsymbol{a}^{(t)}\}_{t=1}^{T}$ corresponding to allocation decisions from a finite convex set of decisions/actions $\mathcal{A}$ at the beginning of each slot t, which results in a sequence of convex performance functions $\{f^{(t)}(\boldsymbol{a}^{(t)})\}_{t=1}^{T}$ that are only known at the end of t.

B. Key Performance Metrics

AI services, which comprise different functions supporting the continuous training and deployment of underlying AI models, exhibit their unique KPIs, including: learning speed, learning latency, and accuracy. The life cycle of an AI service can be divided into three primary phases or functions [5], which are:

Data aggregation and data pre-processing: It involves collecting the user data necessary for training the learning model. The main resources in demand for this phase are communication resources. It is integral to jointly allocate the communication resources needed to serve this function and the computing resources needed to serve model training of each AI model, while fulfilling an allowable maximum learning latency requirement.

Model training: This phase is usually computationally intensive especially for complex AI models, and it depends on: the size of the training data, and the number of epochs. Tuning these two hyper-parameters for each AI model affects the computational demand as well as the performance of the trained model in terms of inference accuracy.

Model inference: The performance of model inference is a crucial factor in obtaining the ideal allocation of resources and tuning of hyper-parameters for AI model training. If the AI model exhibits a low performance/accuracy, resource re-allocation and re-tuning of hyper-parameters may be necessary. By adjusting the resources allocated and tuning the hyper-parameters to perform AI model training functions, the AI model can be optimized to perform better while still maintaining the coexistence of multiple services/AI models on the same platform.

Learning accuracy and learning latency are two essential performance metrics for any AI model $i \in \mathcal{I}$ that depend on: 1) the allocated computational and communication resources at the t-th time slot denoted by $\psi_i^{(t)}$ and $\lambda_i^{(t)}$ respectively, 2) the tuning of the two AI model training hyper-parameters, data size and number of epochs, at the t-th time slot which we denote by $L_i^{(t)}$ and $m_i^{(t)}$ respectively:

Learning latency: is defined as the total delay to fully train an AI model [31], which consists of: the communication delay of acquiring data from different users/locations d_{comm}, and the processing delay for completing the model training d_{proc}.

Therefore, the learning latency experienced by an AI model $i \in \mathcal{I}$ at the t-th slot is defined as:

$$D(L_i^{(t)}, m_i^{(t)}, \psi_i^{(t)}, \lambda_i^{(t)}) = d_{\text{comm}}(L_i^{(t)}, \lambda_i^{(t)}) + d_{\text{proc}}(L_i^{(t)}, m_i^{(t)}, \psi_i^{(t)}), \quad (1)$$

The communication delay at the t-th slot is defined as the time taken to transmit the acquired data of size $L_i^{(t)}$ from different locations to the designated server with the allocated data rate $\lambda_i^{(t)}$. It is expressed as:

$$d_{\text{comm}}(L_i^{(t)}, \lambda_i^{(t)}) = \frac{L_i^{(t)}}{\lambda_i^{(t)}} + \epsilon, \quad (2)$$

where ϵ is the channel access delay [32]. The processing delay of training an AI model i at the t-th slot is given by [31]:

$$d_{\text{proc}}(L_i^{(t)}, m_i^{(t)}, \psi_i^{(t)}) = m_i^{(t)} \times \left(\frac{\phi_i L_i^{(t)}}{\psi_i^{(t)}}\right), \quad (3)$$

where $m_i^{(t)}$ is the number of epochs, ϕ_i is the CPU cycles required to compute one sample of data, representing the model's computational complexity, and $\psi_i^{(t)}$ is the CPU frequency allocated to service i (i.e., computing resources). We remark that the allocated CPU, $\psi_i^{(t)}$, and data rate, $\lambda_i^{(t)}$, allow a degree of freedom with respect to the trade-off between the training latency and cost. In other words, allocating more resources would reduce the training latency, however, it would entail a higher cost.

Learning accuracy: is defined as the inference (or prediction) result on input data compared against the true values, which directly reflects the quality of the training. We denote the inference accuracy of a model $i \in \mathcal{I}$ that has completed the training phase at slot t as $q_i^{(t)}(L_i^{(t)}, m_i^{(t)}) \in [0, 1]$. This accuracy depends on: 1) the size of the acquired data $L_i^{(t)}$, 2) the quality of the data, 3) the distribution of the training and inference data, 4) the number of epochs $m_i^{(t)}$ adopted to train the model, 5) and the service provider's decision regarding the method of training their AI model. Some methods of training AI models are explained in [33]: 1) *full retraining*: involves periodic retraining of the model from scratch whenever a certain size of new data is available, 2) *online training*: the model is initially trained and a mini-batch iteration is triggered once a new batch is available, 3) *proactive training*: a training iteration over an initially trained model is triggered when a specified number of new elements is available.

C. Problem Formulation

The objective of our *slicing for AI* problem is to continuously train multiple AI models, $i \in \mathcal{I}$, corresponding to different AI services. The goal is to determine the resource allocation that maximizes the collective performance of these models while satisfying the diverse requirements of the services at each time slot t. To streamline presentation, we introduce the vector $\boldsymbol{a}^{(t)} = (L_1^{(t)}, m_1^{(t)}, \psi_1^{(t)}, \lambda_1^{(t)}, \ldots, L_I^{(t)}, m_I^{(t)}, \psi_I^{(t)}, \lambda_I^{(t)})$, that determines the allocation decision at the beginning of the t-th slot, where I is the cardinality of $\mathcal{I}$. We define the performance function of the system as the normalized weighted summation of all AI model accuracies at the end of slot t:

$$f^{(t)}(\boldsymbol{a}^{(t)}) = \frac{\sum_i^I \alpha_i \cdot q_i^{(t)}(L_i^{(t)}, m_i^{(t)})}{\sum_{i=1}^I \alpha_i}, \quad (4)$$

where each AI model is associated with a fixed weighting coefficient α_i that determines the relative priority of the service. We remark that this type of problems is considered a bandit problem where the observed system performance feedback is only limited to the selected decision $\boldsymbol{a}^{(t)}$.

The challenge in the formulated problem is that $f^{(t)}(\boldsymbol{a})$ for any $\boldsymbol{a} \in \mathcal{A}$ depends on $q_i^{(t)}$ for each $i \in \mathcal{I}$, which is initially unknown when the allocation decision is determined at beginning of t. Additionally, the accuracy $q_i^{(t)}$ of an AI model i could adversarially change for the same pair of L_i and m_i according to some of the various previously mentioned factors as well as the availability of the training data (i.e., the observed system performance can unpredictably change from one time slot to another [34]). In such an online system, the learner selects the allocation decision, $\boldsymbol{a}^{(t)}$, at the beginning of slot t based on the preceding sequence of performance functions $\{f^{(\tau)}(\boldsymbol{a}^{(\tau)})\}_{\tau=1}^{t-1}$ (i.e., based on available information up to slot $t-1$). Hence, in order to evaluate the performance of the online policy, we need to compare it against the system performance corresponding to the optimal allocation decision $\boldsymbol{a}^\star = (L_1^\star, m_1^\star, \psi_1^\star, \lambda_1^\star, \ldots, L_I^\star, m_I^\star, \psi_I^\star, \lambda_I^\star)$, which is the solution to the following optimization problem with the objective to maximize the system performance:

$$\mathbf{P}: \max_{\boldsymbol{a} \in \mathcal{A}} \sum_{t=1}^T f^{(t)}(\boldsymbol{a}) \quad (5)$$

subject to

$$\sum_{i=1}^I \psi_i \leq \Psi_{\max} \quad (6)$$

$$\sum_{i=1}^I \lambda_i \leq \Lambda_{\max} \quad (7)$$

$$D(L_i, m_i, \psi_i, \lambda_i) \leq D_{\max_i}, \quad \forall i \in \mathcal{I} \quad (8)$$

$$C(\psi_i, \lambda_i) \leq C_{\max_i}, \quad \forall i \in \mathcal{I} \quad (9)$$

$$L_{\min_i} \leq L_i \leq L_{\max_i}, \quad \forall i \in \mathcal{I} \quad (10)$$

$$m_{\min_i} \leq m_i \leq m_{\max_i}, \quad \forall i \in \mathcal{I} \quad (11)$$

The constraints in (6) and (7) ensure that the summation of the allocated resources for all $i \in \mathcal{I}$ does not exceed the maximum available computing resources $\Psi_{\max}$ and communication resources $\Lambda_{\max}$. The Constraints in (8) and (9) respectively represent the maximum allowable learning latency $D_{\max_i}$ and cost $C_{\max_i}$ to train the AI model i. Herein, the learning cost at the t-th slot, $C(\psi_i^{(t)}, \lambda_i^{(t)})$, is defined as the sum of the cost of allocated resources:

$$C(\psi_i^{(t)}, \lambda_i^{(t)}) = c_\psi \cdot \psi_i^{(t)} + c_\lambda \cdot \lambda_i^{(t)}, \quad (12)$$

where c_ψ is the cost per computing unit and c_λ is the cost per unit of data rate. The allowable range of the data size (i.e., $[L_{\min_i}, L_{\max_i}]$) and the number of epochs (i.e., $[m_{\min_i}, m_{\max_i}]$) to train each AI model i is respectively given by constraints (10) and (11).

It is clear that $\boldsymbol{a}^\star = \arg\max_{\boldsymbol{a}\in\mathcal{A}} \sum_{t=1}^{T} f^{(t)}(\boldsymbol{a})$ is a hypothetical solution that can only be determined with hindsight (i.e., to devise such a decision, knowledge about the sequence of all future performance functions $\{f^{(t)}(\boldsymbol{a})\}_{t=1}^{T}$ is required) and we compare it with the online policy according to the selected $\boldsymbol{a}^{(t)}$ at every t using the regret metric:

$$R_T = \sum_{t=1}^{T} f^{(t)}(\boldsymbol{a}^\star) - \sum_{t=1}^{T} f^{(t)}(\boldsymbol{a}^{(t)}), \tag{13}$$

which measures the loss accumulated across time as a result of not choosing the optimal action, $\boldsymbol{a}^\star$, at each slot t. The goal of an online policy is to achieve a sub-linear regret, $R_T = O(\sqrt{T})$, such that the loss decreases as T grows, $\lim_{T\to\infty} \frac{R_T}{T} = 0$. A diminishing regret indicates that the online decisions are achieving a performance identical to that of the optimal action in hindsight.

The formulated problem depends on multiple integer variables (i.e., $(L_i, m_i, \psi_i, \lambda_i)\ \forall i \in \mathcal{I}$) and hence it is a combinatorial optimization problem which can not be easily tackled with conventional optimization techniques. We prove the NP-hardness of the problem in the following Lemma.

Lemma 1: The slicing for AI problem formulated in **P** is NP-hard problem.

Proof: See the Appendix ■

IV. Proposed Solutions of the Slicing for AI Problem

This section presents an online learning framework to solve the problem in **P**, illustrated in Figure 2. At each time slot t, the online optimization controller makes an allocation decision, $\boldsymbol{a}^{(t)}$, for AI model deployment requests. A performance monitoring module tracks model performance and provides feedback, $f^{(t)}(\boldsymbol{a}^{(t)})$, to guide the next decision at time $t+1$

To address problem **P**, we propose Online Learning for Slicing (OLS) as a basic solution, along with two variants to optimize the decision space: OLS with Super Actions (OLS-SA) and OLS with Reduced Super Actions (OLS-RSA). These algorithms consist of two phases: 1) a pre-learning phase for decision space preprocessing, and 2) a learning phase based on the EXP3 algorithm [6], which adapts to dynamic AI model behaviors without assuming a specific distribution. Each algorithm builds upon the pre-learning phase of the others. The details are provided in Section IV-A. To accelerate learning, we propose biased subset selection using prior knowledge, with two methods: Strictly Biased Selection (SBS) and Gaussian Biased Selection (GBS), discussed in Section IV-B.

A. Online Learning Solution

Now we will present the different variants of our online learning solution.

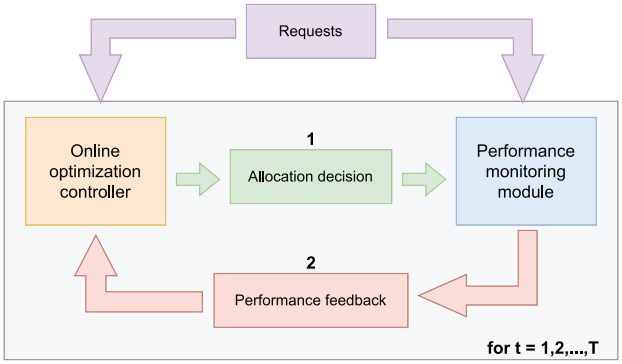


Fig. 2. The online learning framework, representing the interaction between the online optimization controller and a performance monitoring module.

Algorithm 1 Online Learning for Slicing (OLS)

Input: c_ψ, c_λ, $\Psi_{\max}$, $\Lambda_{\max}$, $D_{\max_i}$, $C_{\max_i}$, $L_{\min_i}$, $L_{\max_i}$, $m_{\min_i}$, $m_{\max_i}$
Initialize: $\eta \in (0,1)$, $\boldsymbol{w}^{(1)} = (1/J, ..., 1/J)$, empty set $\tilde{\mathcal{S}}$, empty set of feasible actions $\mathcal{A}_{\text{OLS}}$
Determine a set $\mathcal{O}$ of all possible hyper-parameters combinations for each AI model $i \in \mathcal{I}$
Determine a set $\mathcal{S}$ of all possible resources combinations for each AI model $i \in \mathcal{I}$
for each $\boldsymbol{s} \in \mathcal{S}$ **do**
  if constraints (6), (7), and (9) are satisfied **then**
    store $\boldsymbol{s}$ in feasible combinations set $\tilde{\mathcal{S}}$
  end if
end for
for each $\boldsymbol{o} \in \mathcal{O}$ **do**
  for each $\boldsymbol{s} \in \tilde{\mathcal{S}}$ **do**
    if constraint (8) is satisfied **then**
      store the combination in $\mathcal{A}_{\text{OLS}}$
    end if
  end for
end for
for t = 1,2,...,T **do**
  choose action $\boldsymbol{a}^{(t)}$ from $\mathcal{A}_{\text{OLS}}$ according to $\boldsymbol{w}^{(t)}$
  receive loss, $y^{(t)}_{j^{(t)}} \in [0,1]$, such that $j^{(t)} \widehat{=} \boldsymbol{a}^{(t)}$
  update
    $\tilde{w}_{j^{(t)}} = w^{(t)}_{j^{(t)}} e^{-\eta y^{(t)}_{j^{(t)}} / w^{(t)}_{j^{(t)}}}$
    for $k \in \mathcal{A}_{\text{OLS}}, k \neq j^{(t)}, \tilde{w}_k = w^{(t)}_k$
    $\forall k, w^{(t+1)}_k = \frac{\tilde{w}_k}{\sum_k \tilde{w}_k}$
end for
Output: $L^{(t)}_i, m^{(t)}_i, \psi^{(t)}_i, \lambda^{(t)}_i\ \forall i \in \mathcal{I}$

1) Online Learning for Slicing: OLS sets the base for our online learning solution, with the main steps outlined in Algorithm 1. The pre-learning phase preprocesses the decision space, eliminating infeasible actions that violate constraints. This involves discretizing decision variables for each AI model i such that: $L_i \in [L_{\min_i}, L\max_i]$, $m_i \in [m_{\min_i}, m_{\max_i}]$, $\psi i \in (0, \Psi_{\max}]$, and $\lambda_i \in (0, \Lambda_{\max}]$. A set $\mathcal{O}$ of all possible hyper-parameter combinations for all $i \in \mathcal{I}$ is determined. Each element $\boldsymbol{o} \in \mathcal{O}$ represents a combination

vector $\boldsymbol{o} = (L_1, m_1, L_2, m_2, \ldots, L_I, m_I)$. Similarly, a set $\mathcal{S}$ of all possible resource combinations is formed, where each element $\boldsymbol{s} = (\psi_1, \lambda_1, \psi_2, \lambda_2, \ldots, \psi_I, \lambda_I)$. To tackle infeasible decisions, only feasible combinations $\boldsymbol{s} \in \mathcal{S}$ satisfying constraints (6), (7), and (9) are stored in $\tilde{\mathcal{S}}$. Then, each combination of $\boldsymbol{o} \in \mathcal{O}$ and $\boldsymbol{s} \in \tilde{\mathcal{S}}$ is checked for satisfying the delay constraint (8). Valid combinations form the allocation decision vector $\boldsymbol{a} = (L_1, m_1, \psi_1, \lambda_1, \ldots, L_I, m_I, \psi_I, \lambda_I)$, which is stored in the set of feasible decisions $\mathcal{A}_{\text{OLS}}$ for the learning phase (lines 17 to 24).

In the learning phase, at each t, the learner selects a decision $\boldsymbol{a}^{(t)}$ from $\mathcal{A}_{\text{OLS}}$ according to a probability vector $\boldsymbol{w}^{(t)}$, representing the probability of a decision being close to the optimal decision $\boldsymbol{a}^\star$. Initially, $\boldsymbol{w^{(1)}}$ is a uniform distribution, $\boldsymbol{w}^{(1)} = (1/J, \ldots, 1/J)$, such that $J = |\mathcal{A}_{\text{OLS}}|$. Based on the selected decision the learner observes the loss $y^{(t)}_{j^{(t)}} \in [0,1]^J$ corresponding to system performance and given by:

$$y^{(t)}_{j^{(t)}} = 1 - f^{(t)}(\boldsymbol{a}^{(t)}), \tag{14}$$

such that $j^{(t)}$ corresponds to the index of the selected decision, $\boldsymbol{a}^{(t)}$, where $\widehat{=}$, describes this correspondence.

The loss is integrated into updating $\boldsymbol{w}^{(t)}$, where the learner constructs the updated vector $\boldsymbol{w}^{(t+1)}$ for the next time step. The update depends on the learning rate $\eta \in (0,1)$, which controls the exploration-exploitation trade-off. Over time, the optimization controller learns to increase the probability of favorable actions and reduce the likelihood of less effective ones. As time progresses, the learner converges to the optimal allocation of resources, $\boldsymbol{a}^\star$, using historical performance feedback. Since some decision variables (e.g., λ_i, ψ_i) are constraints rather than objectives, this provides a degree of freedom in terms of the trade-off between cost and learning latency of an AI model. Consequently, multiple actions may yield the same system performance, allowing the learner to identify a subset of optimal allocation decisions of size K, $\{\boldsymbol{a}^\star_1, \boldsymbol{a}^\star_2, \ldots, \boldsymbol{a}^\star_K\} \subset \mathcal{A}_{\text{OLS}}$, sharing the same hyper-parameter combination $\boldsymbol{o} \in \mathcal{O}$, rather than a single optimal decision.

The time complexity of the pre-learning phase involves two steps: 1) checking the feasibility of each $\boldsymbol{s} \in \mathcal{S}$, and 2) checking the feasibility of each $\boldsymbol{o} \in \mathcal{O}$ with each $\boldsymbol{s} \in \tilde{\mathcal{S}}$. Since $|\mathcal{O}||\tilde{\mathcal{S}}| \gg |\mathcal{S}|$, the time complexity of the pre-learning phase is $T_{\text{PL}}(\text{OLS}) = O(|\mathcal{O}||\tilde{\mathcal{S}}|)$. The learning phase involves updating $\boldsymbol{w}^{(t)}$ for each $\boldsymbol{a} \in \mathcal{A}_{\text{OLS}}$ at each time step t, resulting in a complexity of $T\text{L}(\text{OLS}) = O(|\mathcal{A}_{\text{OLS}}|T)$. Therefore, the cumulative time complexity is: $T_{\text{cum}}(\text{OLS}) = T_{\text{PL}}(\text{OLS}) + T_{\text{L}}(\text{OLS})$.

2) Online Learning for Slicing With Super Actions: The OLS-SA algorithm combines decisions with the same objective to create a compact decision space. It extends the OLS pre-learning phase by defining super actions, which group sub-actions that share the same combination $\boldsymbol{o} \in \mathcal{O}$. The steps of OLS-SA are outlined in Algorithm 2.

In Algorithm 2, the pre-learning phase begins by determining the feasible resource set $\tilde{\mathcal{S}}$ based on cost and resource constraints. For each $\boldsymbol{o} \in \mathcal{O}$, a super action set, $\mathcal{B}_{\text{SA}_{\boldsymbol{o}}}$, is created. Each combination of $\boldsymbol{o} \in \mathcal{O}$ and $\boldsymbol{s} \in \tilde{\mathcal{S}}$ is evaluated against the learning latency constraint, and feasible combinations are represented as vectors $\boldsymbol{a}$ within $\mathcal{B}\text{SA}_{\boldsymbol{o}}$. Consequently, all $\boldsymbol{a} \in \mathcal{B}\text{SA}_{\boldsymbol{o}}$ exhibit the same performance while differing in communication and computation resources. If $\mathcal{B}_{\text{SA}_{\boldsymbol{o}}}$ is non-empty, it is added to the set of super actions, $\mathcal{A}_{\text{OLS-SA}}$. During the learning phase, at each slot t, a super action $\mathcal{B}^{(t)}_{\text{SA}_{\boldsymbol{o}}}$ is selected, and a specific allocation decision $a^{(t)} \in \mathcal{B}^{(t)}_{\text{SA}_{\boldsymbol{o}}}$ is chosen randomly. The probability vector $\boldsymbol{w}^{(t)}$ is updated based on the selected super action. Over time, the learner identifies the optimal super action set, $\mathcal{B}^\star_{\text{SA}_{\boldsymbol{o}}} = \{\boldsymbol{a}^\star_1, \boldsymbol{a}^\star_2, \ldots, \boldsymbol{a}^\star_K\}$, which includes the K optimal allocation decisions identified by the OLS algorithm.

The concept of super actions reduces the size of the decision space for the learning phase in OLS-SA, thereby improving cumulative time complexity. The pre-learning phase involves: 1) Identifying feasible decisions, and 2) Grouping similar decisions into super actions. Since combining decisions introduces minimal overhead, the pre-learning phase has a complexity of $T_{\text{PL}}(\text{OLS}-\text{SA}) = O(|\mathcal{O}||\tilde{\mathcal{S}}|)$. The learning phase complexity is $T_{\text{L}}(\text{OLS}-\text{SA}) = O(|\mathcal{A}_{\text{OLS-SA}}|T)$, giving a cumulative complexity of $T_{\text{cum}}(\text{OLS}-\text{SA}) = T_{\text{PL}}(\text{OLS}-\text{SA}) + T_{\text{L}}(\text{OLS}-\text{SA})$. Since $|\mathcal{A}_{\text{OLS-SA}}|$ representing the number of super actions, is smaller than $|\mathcal{A}_{\text{OLS}}|$, it follows that, $T_{\text{cum}}(\text{OLS}-\text{SA}) < T_{\text{cum}}(\text{OLS})$.

3) Online Learning for Slicing With Reduced Super Actions: The OLS-RSA algorithm leverages the ML principle that increasing epochs and data size improves AI model accuracy. It reduces the decision space by identifying candidate super actions with hyper-parameters having the highest epochs and/or training data size. For example, the three candidate

Algorithm 2 Online Learning for Slicing With Super Actions (OLS-SA)

1: **Input:** c_ψ, c_λ, $\Psi_{\max}$, $\Lambda_{\max}$, D_{maxi}, C_{maxi}, L_{mini}, L_{maxi}, m_{mini}, m_{maxi}
2: **Initialize:** an empty set of super actions subsets $\mathcal{A}_{\text{OLS-SA}}$
3: Run algorithm 1 from line 2 to line 9
4: **for** each $\boldsymbol{o} \in \mathcal{O}$ **do**
5: construct an empty super action set $\mathcal{B}_{\text{SA}_{\boldsymbol{o}}}$
6: **for** each $\boldsymbol{s} \in \tilde{\mathcal{S}}$ **do**
7: **if** constraint (8) is satisfied **then**
8: store the combination in $\mathcal{B}_{\text{SA}_{\boldsymbol{o}}}$
9: **end if**
10: **end for**
11: **if** $\mathcal{B}_{\text{SA}_{\boldsymbol{o}}}$ consists feasible sub actions **then**
12: store $\mathcal{B}_{\text{SA}_{\boldsymbol{o}}}$ in $\mathcal{A}_{\text{OLS-SA}}$
13: **end if**
14: **end for**
15: Run the learning phase of Algorithm 1 from line 17 until line 24 to determine the selected super action at the t-th slot $\mathcal{B}^{(t)}_{\text{SA}_{\boldsymbol{o}}}$
16: Choose a sub action $\boldsymbol{a}^{(t)}$ from $\mathcal{B}^{(t)}_{\text{SA}_{\boldsymbol{o}}}$ at random
17: **Output:** $L^{(t)}_i, m^{(t)}_i, \psi^{(t)}_i, \lambda^{(t)}_i \ \forall i \in \mathcal{I}$

Algorithm 3 Online Learning for Slicing With Reduced Super Actions (OLS-RSA)

Input: c_ψ, c_λ, $\Psi_{\max}$, $\Lambda_{\max}$, $D_{\max_i}$, $C_{\max_i}$, $L_{\min_i}$, $L_{\max_i}$, $m_{\min_i}$, $m_{\max_i}$
Initialize an empty set $\mathcal{A}_{\text{OLS}-\text{RSA}}$ to store candidates of the optimal super action
Run algorithm 1 from line 2 to line 9
for each $o \in \mathcal{O}$ **do**
  Run Algorithm 2 form line 8 to line 10
  if $\mathcal{B}_{\text{SA}_o}$ consists feasible sub actions **then**
    if $\mathcal{A}_{\text{OLS}-\text{RSA}}$ is empty **then**
      store $\mathcal{B}_{\text{SA}_o}$ in $\mathcal{A}_{\text{OLS}-\text{RSA}}$
    end if
    copy $\mathcal{A}_{\text{OLS}-\text{RSA}}$ into the set $\tilde{\mathcal{A}}$
    for each $\tilde{\mathcal{B}}_{\text{SA}_o} \in \tilde{\mathcal{A}}$ **do**
      check for candidacy of the super action $\mathcal{B}_{\text{SA}_o}$ over $\tilde{\mathcal{B}}_{\text{SA}_o}$ according to criteria 1
      store the super actions $\tilde{\mathcal{B}}_{\text{SA}_o}$ with overtook candidacy in the set $\mathcal{R}$
      check for candidacy of $\mathcal{B}_{\text{SA}_o}$ according to criteria 2
      if candidacy is confirmed **then**
        Set $\mathcal{B}_{\text{SA}_o}$ *isCandidate* = *True*
      end if
    end for
    if $\mathcal{B}_{\text{SA}_o}$ *isCandidate* **then**
      store in the set $\tilde{\mathcal{A}}$
      update: $\mathcal{A}_{\text{OLS}-\text{RSA}} = \tilde{\mathcal{A}} - \mathcal{R}$
    end if
  end if
end for
Run the learning phase of Algorithm 1 from line 17 until line 24 to determine the selected super action at the *t*-th slot $\mathcal{B}^{(t)}_{\text{SA}_o}$
Choose a sub action $\boldsymbol{a}^{(t)}$ from $\mathcal{B}^{(t)}_{\text{SA}_o}$ at random
Output: $L_i^{(t)}, m_i^{(t)}, \psi_i^{(t)}, \lambda_i^{(t)}\ \forall i \in \mathcal{I}$

optimal super actions $\mathcal{B}_{\text{SA}_1}$, $\mathcal{B}_{\text{SA}_2}$, and $\mathcal{B}_{\text{SA}_3}$ respectively represented as $(L_{1,1}, m_{1,1}, L_{2,1}, m_{2,1})$, $(L_{1,2}, m_{1,2}, L_{2,2}, m_{2,2})$, and $(L_{1,3}, m_{1,3}, L_{2,3}, m_{2,3})$, where $L_{1,1} > L_{1,2} > L_{1,3}$, $m_{1,1} < m_{1,2} < m_{1,3}$, $L_{2,1} = L_{2,2} = L_{2,3}$, and $m_{2,1} = m_{2,2} = m_{2,3}$, exhibit a trade-off between higher epochs against a larger data size for AI model 1. Accordingly, OLS-RSA extends OLS-SA's pre-learning phase to identify the super actions that possesses such a trade-off, eliminating non-candidates to optimize the decision space, hence the name "Reduced Super Actions".

Algorithm 3 begins by identifying the feasible resource combinations $\tilde{\mathcal{S}}$. For each $\boldsymbol{o} \in \mathcal{O}$, feasible combinations with $\boldsymbol{s} \in \tilde{\mathcal{S}}$ are grouped into the super action set $\mathcal{B}\text{SA}\boldsymbol{o}$. The first feasible super action is added to the initially empty decision space $\mathcal{A}_{\text{OLS}-\text{RSA}}$. In subsequent iterations, $\mathcal{A}_{\text{OLS}-\text{RSA}}$ is copied to an intermediary set $\tilde{\mathcal{A}}$ for optimal super action candidacy checking. Given $(L_1, m_1, \ldots, L_I, m_I)$, corresponding to $\mathcal{B}_{\text{SA}_o}$ and, $(\tilde{L_1}, \tilde{m_1}, \ldots, \tilde{L_I}, \tilde{m_I})$, corresponding to a previously stored super action $\tilde{\mathcal{B}}_{\text{SA}_o} \in \tilde{\mathcal{A}}$, candidacy checking is conducted based on two criteria: 1) Criteria 1: $\mathcal{B}\text{SA}\boldsymbol{o}$ overtakes the candidacy of a previously stored super action $\tilde{\mathcal{B}}\text{SA}\boldsymbol{o}$ if $L_i \geq \tilde{L}i$ and $mi \geq \tilde{m}_i$ for all $i \in \mathcal{I}$. Subsequently, $\mathcal{B}_{\text{SA}_o}$ is marked as a candidate by the flag *isCandidate*, while all $\tilde{\mathcal{B}}_{\text{SA}_o} \in \tilde{\mathcal{A}}$ with overtook candidacy are added to the set $\mathcal{R}$ for elimination. 2) Criteria 2: $\mathcal{B}_{\text{SA}_o}$ is a new candidate marked by the flag *isCandidate* if $L_i > \tilde{L}_i$ & $m_i < \tilde{m}_i$ or $L_i \leq \tilde{L}_i$ & $m_i > \tilde{m}_i$, for any $i \in \mathcal{I}$. Super actions marked as candidates are added to $\tilde{\mathcal{A}}$, and overtaken candidates are removed by updating $\mathcal{A}_{\text{OLS}-\text{RSA}} = \tilde{\mathcal{A}} - \mathcal{R}$. Finally, the learning phase is executed as explained in the OLS-SA algorithm to obtain $\mathcal{B}^{(t)}_{\text{SA}_o}$ then $\boldsymbol{a}^{(t)} \in \mathcal{B}^{(t)}_{\text{SA}_o}$. over time, the learner identifies the same optimal super action consisting the optimal allocation decisions identified by the OLS-SA and OLS algorithms, $\mathcal{B}^{\star}_{\text{SA}_o} = \{\boldsymbol{a}^{\star}_1, \boldsymbol{a}^{\star}_2, \ldots, \boldsymbol{a}^{\star}_K\}$.

The time complexity of the pre-learning phase $T_{\text{PL}}(\text{OLS}-\text{RSA})$ involves three main steps: 1) Identifying feasible decisions, 2) Grouping similar decisions into super actions, 3) Determining optimal super action candidates. The third step, candidacy checking, requires comparing each new feasible super action, $\mathcal{B}_{\text{SA}_o}$, with existing candidates and eliminating those with overtaken candidacy. This third step introduces significant initial overhead compared to the pre-learning phases of OLS and OLS-SA. The learning phase complexity is $T_{\text{L}}(\text{OLS}-\text{RSA}) = O(|\mathcal{A}_{OLS-RSA}|T)$, leading to a cumulative complexity of $T_{\text{cum}}(\text{OLS}-\text{RSA}) = T_{\text{PL}}(\text{OLS}-\text{RSA}) + T_L(\text{OLS}-\text{RSA})$. Although the pre-learning phase of OLS-RSA has the highest initial complexity, the reduced size of the decision space ($|\mathcal{A}_{\text{OLS}-\text{RSA}}| < |\mathcal{A}_{\text{OLS}-\text{SA}}|$) ensures that, as $T \to \infty$, the learning phase complexity becomes the lowest. Thus. over time, the overall complexity of OLS-RSA is the smallest: $T_{\text{cum}}(\text{OLS}-\text{RSA}) < T_{\text{cum}}(\text{OLS}-\text{SA}) < T_{\text{cum}}(\text{OLS})$.

B. Biased Action Space Subset Selection

The learning phases of OLS, OLS-SA, and OLS-RSA guarantee an upper regret bound for the accumulated loss over T time slots which is given by [6, Corollary 4.2]:

$$R_T \leq \frac{\log(J)}{\eta} + \eta JT, \tag{15}$$

Ideally, in a highly adversarial environment, we are interested in the optimal (i.e., smallest) upper regret bound. This is achieved at the optimal η given by [6]:

$$\eta_{\text{op}} = \sqrt{\log(J)/(JT)} \tag{16}$$

The learning phase typically initializes $\boldsymbol{w}^{(1)}$ uniformly across all actions or super actions in the decision space. However, if prior experience indicates that the optimal action/super action likely lies within a specific subset, it is beneficial to bias the initial probability distribution toward that subset. This subset can be identified based on previous resource allocation tasks, utilizing knowledge of hyper-parameter combinations that historically maximized system performance. To achieve this, we propose two methods for initializing the probability vector, centering it around a subset

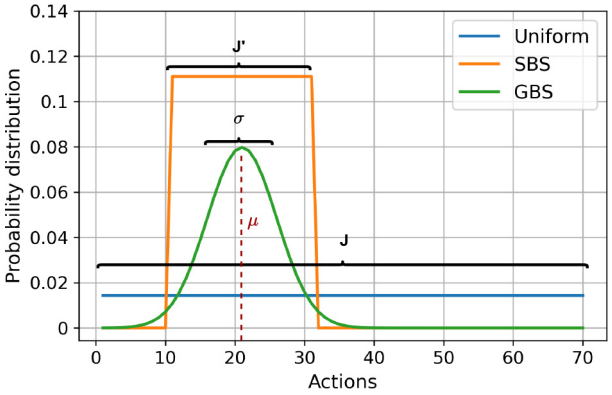


Fig. 3. A representation of different initial probability distribution schemes.

of actions ordered by increasing hyper-parameter combinations for each $i \in \mathcal{I}$.

1) Strictly Biased Subset (SBS): We consider a subset of actions of size $J' < J$ over which the initial probability is uniformly distributed, such that $J = |\mathcal{A}|$. The probability for each action in this subset is $1/J'$, while the probability for the remaining actions is zero, hence the term "Strictly Biased Subset". This leads to a redefined upper regret bound from equation (15) as:

$$R'_T \leq \frac{\log(J')}{\eta} + \eta J' T, \tag{17}$$

therefore, the optimal learning rate η'_{op} for a highly adversarial environment is redefined as:

$$\eta'_{\text{op}} = \sqrt{\log(J')/(J'T)} \tag{18}$$

Thus, a smaller subset of size $J' < J$, results in a lower $R'_T < R_T$ and a larger $\eta'_{\text{op}} > \eta_{\text{op}}$.

2) Gaussian Biased Subset (GBS): Identifying a strictly biased subset from prior knowledge excludes the possibility of selecting actions outside this subset by assigning zero probability to them. However, this approach risks missing the optimal action if it is not included in the subset. Such scenarios can arise due to the distinct learning behaviors of different AI models, where hyper-parameter combinations effective in one resource allocation task may not generalize well to others.

To address this limitation, we propose a Gaussian-based biased selection. Instead of assigning zero probability to actions outside the identified subset, the initial probability distribution is shaped by a Gaussian function, concentrating probabilities around a specific subset while retaining non-zero probabilities for other actions. This ensures flexibility in exploring actions outside the subset. The Gaussian distribution is characterized by its mean μ and standard deviation σ (see Figure 3). The proximity of μ to the optimal action determines the accuracy of subset identification, while σ controls the distribution's spread, influencing the extent of exploration beyond the subset. Notably, the proximity of actions refers to their relative positions in the action space.

TABLE II
COEFFICIENTS OF THE CONSIDERED DL MODELS

Coefficients	$(g_1, g_2, g_3, g_4, g_5, g_6)$
DL model 1	(-60, -0.03109, 96.98, 0.0006553, -120, -0.8355)
DL model 2	(-48, -0.03, 98.5, 0.001, -97, -0.5)
DL model 3	(-40, -0.04, 97, 0.002, -110, -0.6)
DL model 4	(-38, -0.04, 95, 0.0015, -100, -0.64)

V. PERFORMANCE EVALUATION

In this section, we will present the considered experimental setup, the conducted experiments, and results to evaluate the performance of the proposed solutions.

A. Experiment Setup

We perform our experiment considering the Deep Learning (DL) model for a mobile health application and the dataset presented in [35]. For the sake of performance evaluation, we derive an exponential regression model to simulate the system's behavior in terms of inference accuracy of the considered DL model. To obtain a regression model, we train the DL model using different data sizes and number of epochs. We then observe the corresponding inference accuracy on some test data by averaging the results over 10 experiments. The considered DL model consists of 11 layers and is trained on 245,921 samples of the associated dataset, which consists of 13 classes representing different physical activities. Our regression model captures the dependence of the inference accuracy, q_i, on the number of epochs, $m_i^{(t)}$, and the percentage of the training data, $l_i^{(t)}$. The percentage of the data size, $l_i^{(t)}$, is considered with respect to the 245,921 training samples (i.e., $l_i^{(t)} = (L_i^{(t)}/245,921) \times 100$). We use the following exponential model to regress the relation between the three variables:

$$q_i(l_i^{(t)}, m_i^{(t)}) = \frac{1}{100}(g_1 e^{g_2 l_i^{(t)}} + g_3 e^{g_4 m_i^{(t)}} + g_5 e^{g_6 m_i^{(t)}}) \tag{19}$$

The coefficients corresponding to DL model 1 presented in Table II is the result of performing the regression, with a root-mean square error (RMSE) of 5.663% and an R-square measure of 0.9379. The RMSE and the R-square measures indicates that the model captures the relation well, and hence is a good fit. We replicate the obtained exponential model of DL model 1 by adjusting the values of the coefficients and obtain regression models to represent three more DL models (see Table II). We highlight that an adversarial behavior can be represented by a changing exponential model, (i.e., $q_i^{(t)}(l_i^{(t)}, m_i^{(t)})$), or coefficients, (i.e., $g_1^{(t)}$, $g_2^{(t)}$, $g_3^{(t)}$, $g_4^{(t)}$, $g_5^{(t)}$, $g_6^{(t)}$), at each time slot according to the distribution and quality of the data. However, for simplicity and without loss of generality, we consider data with fixed distribution and quality. Hence, we fix the model (19) and its corresponding coefficients for all the considered DL models throughout the entire time horizon.

TABLE III
SIMULATION PARAMETERS

Service Requirements	
Request	$(\alpha, \phi, C_{\max}, D_{\max}, l_{\min}, l_{\max}, m_{\min}, m_{\max})$
DL model 1	(1, 350000, 0.46, 3.70, 25, 100, 2, 10)
DL model 2	(1, 350000, 0.36, 4.50, 25, 100, 2, 10)
DL model 3	(1, 350000, 0.36, 2.43, 25, 100, 2, 10)
DL model 4	(1, 350000, 0.36, 5.3, 20, 100, 3, 10)
Available Resources & Cost	
$(\Psi_{\max}, \Lambda_{\max}, c_\psi, c_\lambda)$	(3.7, 5, 0.2, 0.02)

After obtaining the regression models, we consider the DL model deployment requests coming as a tuple represented by: $(\alpha, \phi, C_{\max}, D_{\max}, l_{\min}, l_{\max}, m_{\min}, m_{\max})$, where $C_{\max}$ is the maximum budget in dollars ($), $D_{\max}$ is the maximum learning latency in minutes (mins), $[l_{\min}, l_{\max}]$ is the range of data size in percentage. The resource allocation problem is solved by the online optimization controller considering the availability and cost of the resources determined by the network operator. The available resources and cost is represented by the tuple: $(\Psi_{\max}, \Lambda_{\max}, c_\psi, c_\lambda)$. $\Psi_{\max}$ is the available units of CPU resources in gigahertz (GHz), and $\Lambda_{\max}$ is the available bandwidth in batches per second (batches/sec), where 10000 samples of data constitute a batch. c_ψ and c_λ represent the cost of a computing unit and a unit rate in dollars respectively. We consider the simulation parameters presented in Table III and assume a negligible channel access delay (i.e., $\epsilon = 0$). We note that, for the purpose of our simulations, a fixed value of ϕ_i is considered for all $i \in \mathcal{I}$. However, in real-world scenarios, this value would vary across different models due to their differing computational complexities.

To assess the performance of our proposed solution, we compare it against two benchmarks. The first benchmark is the Optimal Allocation (OA), which evaluates learning performance relative to the optimal system performance at each time slot, $f^{(t)}(a^\star)$. This benchmark requires knowledge of all future performance functions to determine the optimal decision, $a^\star$, which is derived by solving the formulated problem in **P** through an exhaustive search across the discretized feasible decision space $\mathcal{A}$. This benchmarking approach is commonly used in the literature [36], [37]. The complexity of this benchmark involves computing $f^{(t)}$ for all actions at each time step, making it computationally expensive. However, it remains a theoretical solution, achievable only with prior knowledge of all future performance functions. As detailed in Section IV-A, there may exist multiple optimal allocation decisions, represented as $\{\boldsymbol{a}1^\star, \boldsymbol{a}2^\star, \ldots, \boldsymbol{a}^\star_K\}$. The second benchmark is random Fixed Allocation (FA), where the network operator randomly decides the allocation of computing and communication resources to any admitted slice. These allocations remain fixed for the entire lifetime of the slices and do not guarantee optimality. Random allocation is also a standard strategy for benchmarking [38].

B. Learning Rate Analysis

We study the impact of the learning rate η on the performance of a learner, the best observable η, and the impact of changing the environment on the behavior of η. we run an experiment considering DL model 1, DL model 2 (see Table II), and the available resources presented in Table III. The action space is discretized according to the following:

- $l_i \in \{25, 50, 100\}\ \forall i \in \mathcal{I}$
- $m_i \in \{2, 5, 10\}\ \forall i \in \mathcal{I}$
- $\psi_i \in \{1.5, 1.8, 2.2\}\ \forall i \in \mathcal{I}$
- $\lambda_i \in \{1, 2, 3\}\ \forall i \in \mathcal{I}$

We compare the performance of OLS algorithm, in terms of the cumulative regret (i.e., R_T), using $\eta = \eta_{\text{op}}$ obtained from equation (16), against its performance using different values for η. The results in Figure 4-(a)shows that although the optimal η guarantees the lowest regret bound (i.e., the regret bound at (η_{op})) for a highly adversarial environment, we could observe a lower cumulative regret using another value of η in a less adversarial environment. We observe that $\eta = 0.001$ exhibits the lowest cumulative regret and hence we consider it as the best observable learning rate for the simulated environment setup. We proceed with comparing the performance of OLS, in terms of the average reward (i.e., the average system performance, $\sum_{t=1}^{T} f^{(t)}(\boldsymbol{a}^{(t)})/T$), against the benchmark schemes OA and FA using $\eta = 0.001$ as shown in Figure 4-(b). We consider a fixed allocation (FA) scheme with randomly assigned values: $l_i = 50$, $m_i = 5$, $\Psi_i = 1.5$, and $\lambda_i = 2$, $\forall i \in \mathcal{I}$. Additionally, we evaluate the average performance of multiple random fixed allocations over 10×10^5 trials, labeled as (FA-avg). Unlike the FA and FA-avg, the OLS framework is guaranteed to converge to the optimal system performance defined by OA. It is worth noting that the fluctuations observed at the initial stages are attributed to the exploration of the learner, as it is actively sampling different actions to gather information about their performance.

In Figure 4-(c), we analyze the performance under two setups, each featuring different number of DL model deployment requests, in terms of the average regret (R_T/T). In the first setup, labeled as (2 DL models), we consider DL model 1 and DL model 2 from Table III, with $\Psi_{\max} = 3.7$. The second setup, labeled as (3 DL models), introduces an additional DL model, DL model 3, and scales up the available resources to $\Psi_{\max} = 5.2$ to accommodate the additional deployment. We identify that the best observable learning rate changes according to the environment setup. We particularly note that by accepting another DL model deployment request and up-scaling the available resources, the size of the decision space increases, which demands a lower value for η. It is also observed that learning on a smaller decision space results in a smaller area under the average regret curve (i.e., the cumulative regret, R_T). In other words, the speed of the learning is affected by the size of the decision space.

C. Comparison of Proposed Algorithms

To compare between the performance of the three proposed variants of the online learning solution, we perform an experiment considering DL model 1, DL model 2 and the available

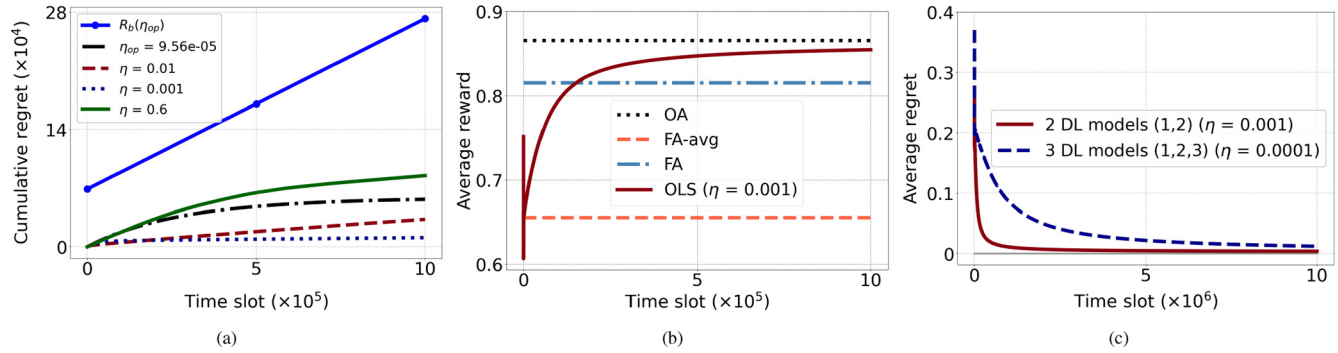


Fig. 4. a) Cumulative regret curve using different values for η compared against the regret bound at η_{op}, b) Average reward curve of OLS against the benchmarks using the best observable η, c) Average regret curve considering different numbers of DL model deployment requests using the best observed η.

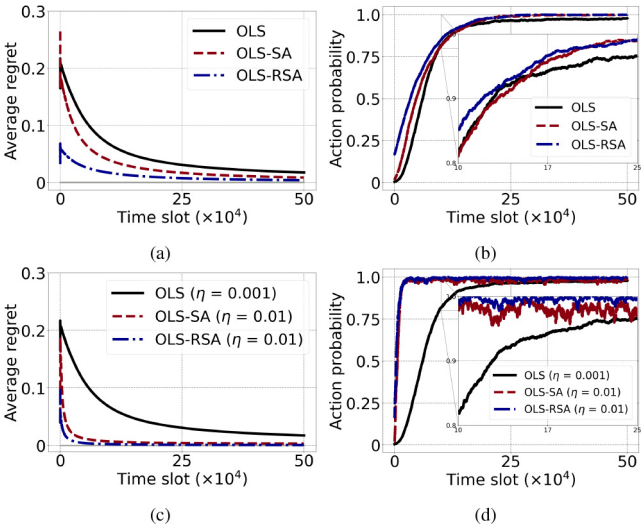


Fig. 5. Comparison between the three elimination methods: a) Average regret curve with $\eta = 0.001$, b) Action probability curve showing the probability given to the optimal action with $\eta = 0.001$, c) Average regret curve using the best observable η, c) Action probability curve showing the probability given to the optimal action using the best observable η.

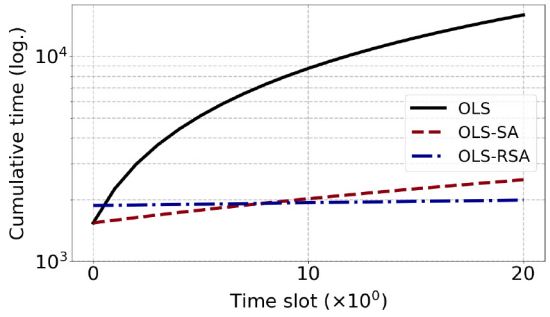


Fig. 6. Comparison of the cumulative time complexity given: $|\mathcal{A}_{\text{OLS}}| = 720$, $|\mathcal{A}_{\text{OLS-SA}}| = 48$, $|\mathcal{A}_{\text{OLS-RSA}}| = 6$, and $T = 20$.

resources presented in Table III. We conduct the comparison in terms of the convergence of the average regret and how fast the probability of the optimal decision approaches a value of 1. Due to the structure of the decision space resulting from the OLS algorithm, as the probability vector is updated at each time slot it will tend to be distributed among multiple optimal decisions rather than a single decision. On the other hand, since OLS-SA and OLS-RSA considers these decisions as sub actions of a single super action, the learning phase will result in a probability that is biased toward a single optimal super action. Therefore, for comparability of the three algorithms, at each time slot we add the probabilities corresponding to the multiple optimal decisions resulting from the OLS algorithm.

In Figure 5, it is observed that in general the OLS-SA algorithm exhibits a lower cumulative regret and hence a faster convergence to a low average regret compared to the OLS algorithm. This is because by combining similar actions together into a super action, they undergo the same probability update, instead of being treated as independent. OLS-SA algorithm reduces the size of the decision space compared to OLS, which increases the speed of the learning. It is also observed that OLS-RSA exhibits the lowest cumulative regret of the three methods. This is because by reducing the size of the decision space, the online optimization controller is able to learn the decisions that maximizes the system performance faster. In addition to this, it is likely that the candidates of the optimal super action will result in a higher system performance compared to other actions, and hence resulting in a faster convergence to a low average regret. It is worth noting that both the OLS-SA and OLS-RSA algorithms exhibit a similar behavior in terms of the probability update of the optimal action (see Figure 5-(b), and Figure 5-(d)). For both algorithms, the probability given to the optimal action nearly converges to a value of 1 as opposed to OLS. We highlight that OLS-SA and OLS-RSA result in a significant reduction in the action space compared to OLS. Therefore, we can achieve a remarkably faster convergence of the probability corresponding to the optimal super action to a value of 1 at the best observable η (see Figure 5-(c), and Figure 5-(d)).

D. Time Complexity Analysis of the Proposed Algorithms

Given the previous experimental setup, we compare between the cumulative time complexity of the three algorithms. To account for the time complexity of the pre-learning phase, we experimentally compute the number of times of executing the major operations described in Section IV-A.

Figure 6 compares the cumulative time complexity of the algorithms over 20 slots on a logarithmic scale. At $t = 0$ (i.e., during the pre-learning phase), the OLS-RSA algorithm incurs

the highest initial overhead due to the optimal super action candidacy checking step. In contrast, the initial overhead for OLS and OLS-SA is identical, stemming from the feasibility checking step. However, both OLS-SA and OLS-RSA reduce the action space size in the pre-learning phase compared to OLS. Time slots $t = \{1, \ldots, 20\}$, show the complexity during the learning phase, which is primarily attributed to the number of operations required to update the probability vector, which heavily depends on the decision space size. Accordingly, the OLS algorithm experiences a significant increase in cumulative time complexity as it requires more operations to update the probability vector for a larger decision space. On the other hand, OLS-RSA achieves the lowest cumulative time complexity over time due to its substantial reduction in the decision space size.

E. Service Performance Evaluation

We use the OLS-RSA algorithm to evaluate the individual performance of several DL models during the process of the online resource allocation. We conduct an experiment considering the four DL models in Table II and the simulation parameters in Table III with $\eta = 0.01$. We set $l_{\min_i} = 20$ $\forall i \in \mathcal{I}$, $m_{\min_i} = 3$ $\forall i \in \mathcal{I}$, $C_{\max_2} = 0.46$, $C_{\max_3} = 0.38$, $D_{\max_1} = 3.07$, $D_{\max_2} = 3.07$, $D_{\max_3} = 4.4$, $\Psi_{\max} = 7$, $\Lambda_{\max} = 7$. The action space is discretized according to the following:

- $l_i \in \{20, 55, 80, 100\}$ $\forall i \in \mathcal{I}$
- $m_i \in \{3, 5, 8, 10\}$ $\forall i \in \mathcal{I}$
- $\psi_i \in \{1.5, 1.8, 2.2\}$ $\forall i \in \mathcal{I}$
- $\lambda_i \in \{1, 2, 3\}$ $\forall i \in \mathcal{I}$

Analyzing the average accuracy of the DL models (see Figure 7), it is observed that each DL model experiences initial fluctuations until the average inference accuracy of each of them converges to a value which corresponds to the maximum of their summation. We observe a trade-off between the allocation of resources and hence the behavior of DL model 1 and DL model 2. This trade-off indicates that the allocation of resources cannot accommodate the combination of data size and number of epochs which leads to the highest possible inference accuracy for both DL models at the same time. It appears that since the objective of the problem is to maximize the summation of accuracies, this trade-off arises as more resources will be allocated to the DL model that weighs higher in terms of accuracy. Hence during the learning phase, the DL model which exhibits a lower accuracy may experience an initial increase in the average performance followed by a decrease as the online optimization controller learns the allocation which corresponds to the collective maximum performance.

F. Biased Subset Selection Evaluation

We have been evaluating the performance considering an initial uniform probability distribution over the entire decision space. Considering the previous experimental setup with four DL models, we extend the analysis to compare between the impact of learning while considering a biased subset of the original decision space (i.e., SBS and GBS) as discussed in Section IV-B, and learning with a uniform distribution over the entire decision space. We analyze the impact of the size of the considered subset J' on the learning. We also study the behavior of the different biased subset selection approaches when it is identified with error and without error. We have identified that the optimal action corresponds to the index $j = 40$ in the decision space. Thus, we consider a strictly biased subset centred at $j = 40$, with $J' = 15$ and $J' = 5$ in the case without error, and another centred at $j = 9$ with $J' = 5$ for the case with error. We also consider a Gaussian biased subset in the case without error to be centred around the optimal action with $\mu = 40$, and another in the case with error with $\mu = 9$. In both cases the standard deviation is set to $\sigma = 3$.

In Figure 8, it is observed that compared to a uniform distribution over the entire decision space, a biased subset selection which includes the optimal action results in a lower cumulative regret (i.e., smaller area under the average regret curve), and a faster convergence of the probability corresponding to the optimal action to a value of 1. It is also observed that the smaller the size of the subset J', the faster the convergence to the optimal action and hence a lower cumulative regret. In the case of identifying the biased subset of actions with error, we observe that the GBS approach allows the online optimization controller to learn the optimal super action in the long time. Accordingly, the average regret will start to decrease as the online optimization controller learns the optimal super action. Meanwhile, using the SBS approach, the learner is not able to identify the optimal super action since the probability assigned to it at initialization is equal to zero. In this case, the online optimization controller will learn a sub-optimal super action within the specified subset. Therefore, it is evident that the GBS approach performs better in comparison to SBS since it is able to account for error.

G. Comparison With State-of-the-Art

In this section, we compare our OLS-RSA solution using the EXP3 algorithm with State-of-the-Art algorithms: Thompson Sampling (TS), Upper Confidence Bound (UCB), Q-Learning (QL), and SARSA. These algorithms were chosen for their similarities to EXP3. TS balances exploration and exploitation by maintaining two parameters, alpha (success rate) and beta (failure rate), which define a beta distribution for action selection. UCB selects actions based on the highest upper confidence bound, reflecting the uncertainty of rewards. QL and SARSA both use an ϵ-greedy policy, exploring with probability ϵ and exploiting the action with the highest Q-value otherwise. QL estimates future rewards based on the highest Q-value, while SARSA uses the ϵ-greedy policy for this estimation. We set $\epsilon = 0.01$, $\alpha = 0.01$ (learning rate), and $\gamma = 0.99$ (discount factor) in our experiments. To ensure a fair comparison, all algorithms undergo the pre-learning action space reduction, and performance is evaluated during the learning phase with varying environment dynamics.

To simulate varying dynamics, we modify the coefficients g_1, g_5, and g_6 in the mathematical models of the DL models. The mean values of these coefficients are shown in

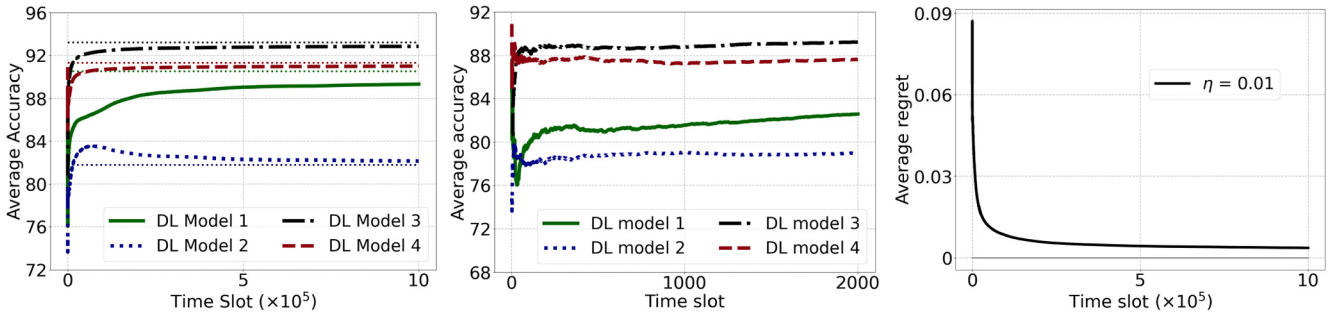


Fig. 7. a) Average accuracy of each DL model, b) Experienced fluctuations during the initial time steps of the learning phase, c)Average regret curve.

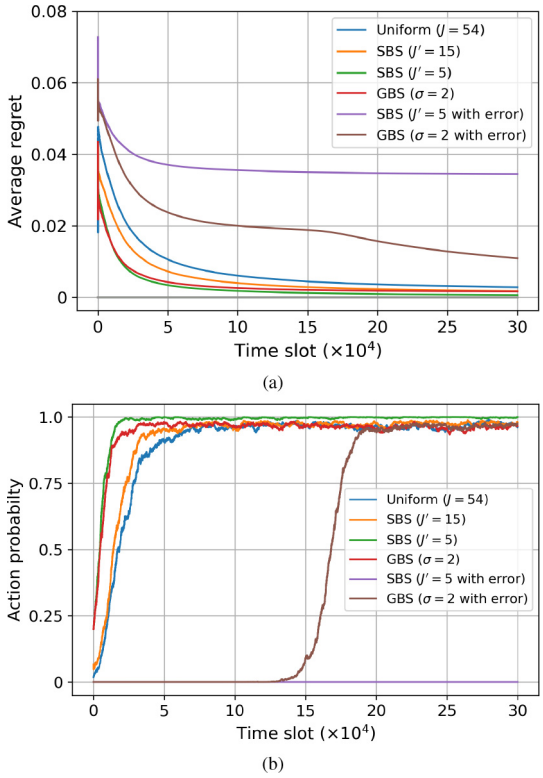


Fig. 8. Comparing the impact of different biased subset selection and initial probability distributions in terms of: a) The average regret b) The probability corresponding to the optimal action at each time step.

TABLE IV
PARAMETERS FOR VARIOUS ENVIRONMENT DYNAMICS

Coefficient		g_1	g_5	g_6
Minimum value		-100	-250	-0.85
Maximum value		-30	-90	-0.2
σ values	Environment 1	0	0	0
	Environment 2	17.5	40	0.1625
	Environment 3	35	80	0.325
	Environment 4	70	160	0.65
	Environment 5	140	320	1.25

Table II, with their standard deviation (σ), maximum, and minimum values provided in Table V. A σ of 0 indicates static dynamics with fixed models, while higher σ values reflect increased dynamics, creating a more adversarial environment. Key parameters are kept constant: $D_{\max}$ at 6.0 minutes, $C_{\max}$ at 0.46, $\Lambda_{\max}$ at 7 batches/sec, and $\Psi_{\max}$ at 8 GHz. The learning phase for each algorithm spans 30×10^4 time steps, with average reward and average regret recorded at the last time slot.

TABLE V
AVERAGE REWARD COMPARISON WITH STATE-OF-THE-ART UNDER VARYING ENVIRONMENT DYNAMICS

env #	Average Reward				
	Optimal	EXP3	TS	UCB	QL/SARSA
1	0.950	0.945	0.940	0.935	0.920
2	0.937	0.929	0.923	0.917	0.909
3	0.919	0.902	0.898	0.890	0.890
4	0.901	0.877	0.876	0.864	0.872
5	0.901	0.878	0.875	0.864	0.872

Table V provides a comparison of the average reward achieved using the EXP3 algorithm against State-of-the-Art learning algorithms and the optimal benchmark. Similarly, Figure 9 illustrates the comparison of the average regret. Since the results for QL and SARSA were identical, they are combined for reporting purposes. The results reveal that the average regret increased for TS and UCB in more adversarial environments, whereas QL and SARSA demonstrated a decrease followed by nearly constant performance in environments 4 and 5. In contrast, EXP3 exhibited an initial increase in average regret, followed by a decrease in environment 5. Furthermore, EXP3 consistently achieved the highest average reward and the lowest average regret across varying environments, showcasing its superior adaptability to diverse environment dynamics.

H. Performance Under Varying Resources Availability

In this section, we assess the performance of OLS-RSA under different levels of computational resource availability and environment dynamics. Building upon the setup from the previous experiment, we vary the value of $\Psi_{\max}$ across the range $[6.0, 8.0]$ in increments of 0.5. Figure 10 illustrates that across all settings, increasing $\Psi_{\max}$ leads to an increase in the achieved average reward. This is attributed to the greater availability of computational resources, enabling more computationally intensive configurations and combinations. These,

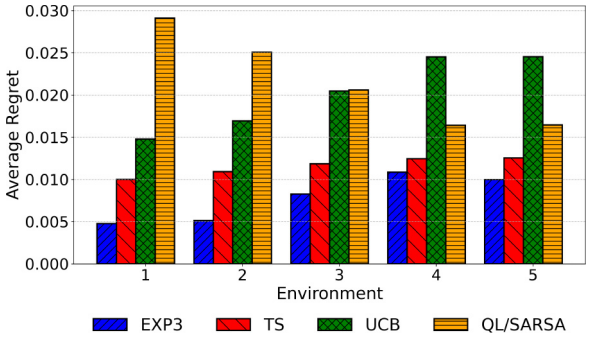


Fig. 9. Average regret comparison with State-of-the-Art under varying environment dynamics.

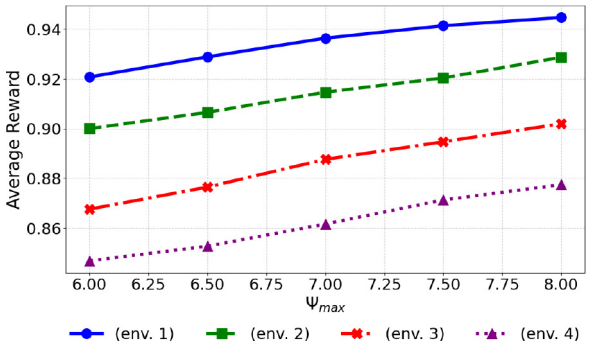


Fig. 10. Average Reward of OLS-RSA achieved in different environments under varying computational resources availability.

in turn, support training models with additional epochs and/or larger data samples, resulting in improved learning accuracy and a corresponding increase in the system's average reward. Moreover, it is evident that under more adversarial settings, the achieved average reward is generally lower, reflecting the challenges posed by these environments.

VI. Conclusion

Towards a new paradigm of 6G network slicing to support AI-based services, *slicing for AI*, we proposed an online learning approach to jointly allocate resources and tune hyper-parameters for training AI models, while maximizing their collective performance. We formulated the problem as an accuracy maximization problem subject to resources, budget, and AI model training latency constraints. We presented an online learning solution with decision space reduction methods and biased decision space subset selection approaches. The simulation results demonstrate the diverse capabilities and trade-offs of the implemented algorithms in achieving an optimal allocation decision while providing sub-linear regret guarantees. Moreover, our findings indicate that optimizing the action space leads to favorable outcomes in terms of faster convergence to the optimal solution, lower loss accumulation, and overall optimization of cumulative time complexity. Additionally, our results highlight the trade-offs in the proposed approaches of biasing a selected subset of the decision space. Lastly, they highlight the superiority of our solution in adapting to various environment dynamics in comparison to other State-of-the-Art solutions and under various resource availability.

Future work could explore a broader range of scenarios with varying computational complexities of AI models. Additionally, an important direction of research could be addressing resource slicing for both AI and conventional services, incorporating dynamic resource allocation where the availability of communication resources for AI services adapts to changing demands from other services.

Appendix

Proof of Lemma 1

The Knapsack problem [39], which is a combinatorial optimization problem that is known to be NP-hard, is reducible to the formulated problem in **P**. The knapsack problem involves selecting a subset of items $\mathcal{S} \subseteq \mathcal{N} = \{1, 2, \ldots, N\}$ from a set of N items, such that each $n \in \mathcal{N}$ is characterized by a value v_n and a weight w_n. The objective of the problem is to select $\mathcal{S}$ such that the total value $\sum_{n=1}^{N} v_n$ is maximized, while satisfying a maximum weight constraint $\sum_{n=1}^{N} w_n \leq W$. Correspondingly, the formulated problem in **P** aims at maximizing the system performance across all t slots by determining the combination $\boldsymbol{a}^\star = (L_1^\star, m_1^\star, \psi_1^\star, \lambda_1^\star, \ldots, L_I^\star, m_I^\star, \psi_I^\star, \lambda_I^\star)$ from the set of combinations $\mathcal{A} = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \}$. Hence, each combination $\boldsymbol{a} \in \mathcal{A}$ representing an allocation decision can be mapped to the considered subset of items, $\mathcal{S}$, in the knapsack problem. The resulting inference accuracy of an allocation decision corresponding to each AI model can be mapped to the value variable in the knapsack problem, and the computing resources, communication resources, learning latency, and learning cost variables corresponding to each AI model can be mapped to the weight variable in the knapsack problem. This shows the formulated problem in **P** is a special case of the knapsack problem proving our Lemma that the formulated problem is NP-hard.

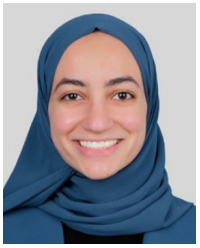
Acknowledgment

The content is solely the responsibility of the authors and does not necessarily represent the official views of Qatar Research Development and Innovation Council.

References

[1] W. Wu et al., "AI-native network slicing for 6G networks," *IEEE Wireless Commun.*, vol. 29, no. 1, pp. 96–103, Feb. 2022.
[2] A. Garcia-Saavedra and X. Costa-Pérez, "O-RAN: Disrupting the virtualized RAN ecosystem," *IEEE Commun. Stand. Mag.*, vol. 5, no. 4, pp. 96–103, Dec. 2021.
[3] W. Azariah, F. A. Bimo, C.-W. Lin, R.-G. Cheng, R. Jana, and N. Nikaein, "A survey on open radio access networks: Challenges, research directions, and open source approaches," *Sensors*, vol. 24, no. 3, p. 1038, 2022.
[4] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 1376–1411, 2nd Quart., 2022.
[5] M. Li, J. Gao, C. Zhou, X. S. Shen, and W. Zhuang, "Slicing-based artificial intelligence service provisioning on the network edge: Balancing AI service performance and resource consumption of data management," *IEEE Veh. Technol. Mag.*, vol. 16, no. 4, pp. 16–26, Dec. 2021.

[6] S. Shalev-Shwartz, *Online Learning and Online Convex Optimization*. Hanover, MA, USA: Now Publ., 2012.
[7] A. Filali, Z. Mlika, S. Cherkaoui, and A. Kobbane, "Dynamic SDN-based radio access network slicing with deep reinforcement learning for URLLC and eMBB services," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 4, pp. 2174–2187, Jul./Aug. 2022.
[8] F. Rezazadeh, L. Zanzi, F. Devoti, H. Chergui, X. Costa-Pérez, and C. Verikoukis, "On the Specialization of FDRL agents for scalable and distributed 6G RAN slicing orchestration," *IEEE Trans. Veh. Technol.*, vol. 72, no. 3, pp. 3473–3487, Mar. 2023.
[9] M. Karbalaee Motalleb, V. Shah-Mansouri, S. Parsaeefard, and O. L. Alcaraz López, "Resource allocation in an open RAN system using network slicing," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 1, pp. 471–485, Mar. 2023.
[10] S. Tripathi, C. Puligheddu, S. Pramanik, A. Garcia-Saavedra, and C. F. Chiasserini, "Fair and scalable orchestration of network and compute resources for virtual edge services," *IEEE Trans. Mobile Comput.*, vol. 23, no. 3, pp. 2202–2218, Mar. 2024.
[11] A. A. Abdellatif, A. Mohamed, A. Erbad, and M. Guizani, "Dynamic network slicing and resource allocation for 5G-and-beyond networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2022, pp. 262–267.
[12] S. Vittal, S. Sarkar, and A. A. Franklin, "Revamping the resilience and high availability of 5G core for 6G ready network slices," *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 2, pp. 2287–2302, Apr. 2024.
[13] Q. Zeng, Y. Du, K. Huang, and K. K. Leung, "Energy-efficient radio resource allocation for federated edge learning," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, 2020, pp. 1–6.
[14] I. Sartzetakis, P. Soumplis, P. Pantazopoulos, K. V. Katsaros, V. Sourlas, and E. M. Varvarigos, "Resource allocation for distributed machine learning at the edge-cloud continuum," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2022, pp. 5017–5022.
[15] M. Chen et al., "Joint data collection and resource allocation for distributed machine learning at the edge," *IEEE Trans. Mobile Comput.*, vol. 21, no. 8, pp. 2876–2894, Aug. 2022.
[16] Z. Lin, S. Bi, and Y.-J. A. Zhang, "Optimizing AI service placement and resource allocation in mobile edge intelligence systems," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7257–7271, Nov. 2021.
[17] A. Papa, A. Jano, S. Ayvaşık, O. Ayan, H. M. Gürsu, and W. Kellerer, "User-based quality of service aware multi-cell radio access network slicing," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 1, pp. 756–768, Mar. 2022.
[18] S. Matoussi, I. Fajjari, N. Aitsaadi, and R. Langar, "User-centric slice allocation scheme in 5G networks and beyond," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 4, pp. 4268–4282, Dec. 2023.
[19] H. Zhou, M. Elsayed, and M. Erol-Kantarci, "RAN resource slicing in 5G using multi-agent correlated Q-learning," 2021, *arXiv:2107.01018*.
[20] F. Z. Mardi, Y. Hadjadj-Aoul, M. Bagaa, and N. Benamar, "Resource allocation for LoRaWAN network slicing: Multi-armed bandit-based approaches," *Internet Things*, vol. 26, Jul. 2024, Art. no. 101195. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660524001367
[21] H. P. Phyu, D. Naboulsi, R. Stanica, and G. Poitau, "Towards energy efficiency in RAN network slicing," in *Proc. IEEE 48th Conf. Local Comput. Netw. (LCN)*, 2023, pp. 1–9.
[22] A. A. Abdellatif, A. Abo-eleneen, A. Mohamed, A. Erbad, N. V. Navkar, and M. Guizani, "Intelligent-slicing: An AI-assisted network slicing framework for 5G-and-beyond networks," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 2, pp. 1024–1039, Jun. 2023.
[23] J. Mei, X. Wang, K. Zheng, G. Boudreau, A. B. Sediq, and H. Abou-Zeid, "Intelligent radio access network slicing for service provisioning in 6G: A hierarchical deep reinforcement learning approach," *IEEE Trans. Commun.*, vol. 69, no. 9, pp. 6063–6078, Sep. 2021.
[24] A. Gharehgoli, A. Nouruzi, N. Mokari, P. Azmi, M. R. Javan, and E. A. Jorswieck, "AI-based resource allocation in end-to-end network slicing under demand and CSI uncertainties," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 3, pp. 3630–3651, Sep. 2023.
[25] M. Alsenwi, N. H. Tran, M. Bennis, S. R. Pandey, A. K. Bairagi, and C. S. Hong, "Intelligent resource slicing for eMBB and URLLC coexistence in 5G and beyond: A deep reinforcement learning based approach," *IEEE Trans. Wireless Commun.*, vol. 20, no. 7, pp. 4585–4600, Jul. 2021.
[26] N. Sen and A. A. Franklin, "Intelligent admission and placement of O-RAN slices using deep reinforcement learning," in *Proc. IEEE 8th Int. Conf. Netw. Softw. (NetSoft)*, 2022, pp. 307–311.
[27] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, and Y.-C. Liang, "Network slice reconfiguration by exploiting deep reinforcement learning with large action space," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 2197–2211, Dec. 2020.
[28] W. Guan, H. Zhang, and V. C. M. Leung, "Customized slicing for 6G: Enforcing artificial intelligence on resource management," *IEEE Netw.*, vol. 35, no. 5, pp. 264–271, Sep./Oct. 2021.
[29] A. Filali, B. Nour, S. Cherkaoui, and A. Kobbane, "Communication and computation O-RAN resource slicing for URLLC services using deep reinforcement learning," *IEEE Commun. Stand. Mag.*, vol. 7, no. 1, pp. 66–73, Mar. 2023.
[30] J.-B. Monteil, G. Iosifidis, and L. A. DaSilva, "Learning-based reservation of virtualized network resources," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 3, pp. 2001–2016, Sep. 2022.
[31] E. Baccour et al., "Pervasive AI for IoT applications: A survey on resource-efficient distributed artificial intelligence," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 4, pp. 2366–2418, 4th Quart., 2022.
[32] A. Awad, O. A. Nasr, and M. M. Khairy, "Energy-aware routing for delay-sensitive applications over wireless multihop mesh networks," in *Proc. 7th Int. Wireless Commun. Mobile Comput. Conf.*, 2011, pp. 1075–1080.
[33] I. Prapas, B. Derakhshan, A. R. Mahdiraji, and V. Markl, "Continuous training and deployment of deep learning models," *Datenbank-Spektrum*, vol. 21, no. 3, pp. 203–212, 2021.
[34] A. Slivkins, "Introduction to multi-armed bandits," 2022.
[35] gaurav2022. "CNN+LSTM+95." Feb. 2021. [Online]. Available: https://www.kaggle.com/code/gaurav2022/cnn-lstm-95/notebook
[36] Y. Cui, X. Yang, P. He, R. Wang, and D. Wu, "Slice freshness guaranteed resource allocation in vehicular networks," in *Proc. 14th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, 2022, pp. 321–325.
[37] S. Nouri, M. K. Motalleb, V. Shah-Mansouri, and S. P. Shariatpanahi, "Semi-supervised learning approach for efficient resource allocation with network slicing in O-RAN," 2024, *arXiv:2401.08861*.
[38] Z. Sasan, M. Shokrnezhad, S. Khorsandi, and T. Taleb, "Joint network slicing, routing, and in-network computing for energy-efficient 6G," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2024, pp. 1–6.
[39] G. J. Woeginger, "Exact algorithms for NP-hard problems: A survey," in *Combinatorial Optimization—Eureka, You Shrink! Papers Dedicated to Jack Edmonds 5th International Workshop Aussois*. Berlin, Germany: Springer, 2003, pp. 185–207.



Menna Helmy received the B.Sc. degree in computer engineering from Qatar University (QU) in 2022, where she is currently pursuing the M.Sc. degree in computing, supported by the Graduate Sponsorship Research Award (GSRA) from the Qatar Research, Development, and Innovation Council. Subsequently, she worked as a Research Assistant with QU from November 2022 to June 2024. Her research focuses on optimizing resource allocation for efficient AI model training supporting AI-based services.

Alaa Awad Abdellatif (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees (with Hons.) in electronics and electrical communications engineering from Cairo University in 2009 and 2012, respectively, and the Ph.D. degree from the Politecnico di Torino, Italy, in 2018. He is currently a Senior Researcher and a Principal Investigator with the Centre for Telecommunications and Multimedia, INESC TEC, Portugal. He has held diverse research and professional roles, including Lecturer, Postdoctoral Researcher, Senior Research Assistant, and Software Engineer, at institutions such as the Politecnico di Torino, Qatar University, Valeo, and Cairo University. He has authored or co-authored over 65 publications, including refereed journal articles, magazine contributions, and conference papers in leading international venues. His research interests span edge computing, blockchain, machine learning, resource optimization for wireless heterogeneous networks, smart healthcare, IoT applications, and vehicular networks. He has received several notable accolades, including the Postdoctoral Research Award and the Graduate Research Award from the Qatar National Research Fund, the Best Paper Award at the 2018 Wireless Telecommunications Symposium, USA, and the Quality Award from Politecnico di Torino in 2018. Also, he has served as a technical reviewer for numerous prestigious journals and magazines.

Naram Mhaisen (Graduate Student Member, IEEE) received the B.Sc. degree in computer engineering from Qatar University, Doha, Qatar, in 2017, with the Order of Excellence award, and the M.Sc. degree (with Distinction) in computing from Qatar University in 2020. He is currently pursuing the Doctoral degree with the Delft University of Technology, The Netherlands. His research focuses on data-driven and optimistic learning techniques for resource management in communication systems. His work has been recognized with the Microsoft Imagine Cup Award in 2017 and the NEC Europe Research Fellowship in 2022. He has authored several journal and conference papers at the intersection of learning frameworks and IoT networks and has delivered talks at conferences and seminars, and in his role as a teaching assistant.

Amr Mohamed (Senior Member, IEEE) received the M.S. and Ph.D. degrees in electrical and computer engineering from The University of British Columbia, Vancouver, Canada, in 2001 and 2006, respectively. He has worked as an Advisory IT Specialist with IBM Innovation Centre, Vancouver, from 1998 to 2007, taking a leadership role in systems development for vertical industries. He is currently a Professor and the Head of the Department of Computer Science and Engineering, Qatar University. He has more than 25 years of experience in wireless networking research and industrial systems development. He has authored or co-authored more than 300 refereed journal and conference papers, textbooks, and book chapters in reputable international journals and conferences and holds six international patents. His research interests include pervasive AI and edge computing for the IoT applications, and open RAN performance optimization and security. He holds three awards from IBM Canada for his achievements and leadership and four best paper awards from IEEE conferences.

Aiman Erbad (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of British Columbia, Canada, in 2012. He is a Professor and a VP Research and Graduate Studies with Qatar University. His research interests span cloud computing, edge intelligence, Internet of Things, private and secure networks, and multimedia systems. In 2013, he received the Platinum award from H.H. The Emir Sheikh Tamim bin Hamad Al Thani at the Education Excellence Day (Ph.D. category). He is an Editor of *KSII Transactions on Internet and Information Systems* and of the *International Journal of Sensor Networks*, and a Guest Editor of IEEE NETWORK. He also served as a Technical Program Committee Member in various IEEE and ACM International Conferences (GlobeCom, NOSSDAV, MMSys, ACMMM, IC2E, and ICNC). He is a Senior Member of ACM.