Received 31 March 2024; accepted 11 April 2024. Date of publication 17 April 2024; date of current version 3 May 2024.
*Digital Object Identifier 10.1109/OJCOMS.2024.3390591*

# ECP: Error-Aware, Cost-Effective and Proactive Network Slicing Framework

**AMR E. ABOELENEEN[1] (Member, IEEE), ALAA A. ABDELLATIF[2] (Member, IEEE), AIMAN M. ERBAD[1] (Senior Member, IEEE), AND AMR M. SALEM[2] (Senior Member, IEEE)**

[1]College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar
[2]College of Engineering, Qatar University, Doha, Qatar

CORRESPONDING AUTHOR: A. E. ABOELENEEN (e-mail: a.aboeleneen@ieee.org)

This work was supported in part by the NPRP-Standard (NPRP-S) Thirteen (13th) Cycle under Grant NPRP13S-0205-200265.
The work of Amr E. Aboeleneen was supported by the GSRA through the Qatar National Research Fund (a Member of Qatar Foundation) under Grant GSRA9-L-1-0518-22022.
Open Access funding provided by the Qatar National Library.

**ABSTRACT** Recent advancements in Software Defined Networks (SDN), Open Radio Access Network (O-RAN), and 5G technology have significantly expanded the capabilities of wireless networks, extending beyond mere data transmission. This progression has led to the emergence of Virtual Networks (VN) and Network Slicing, enabling industries to enhance their services and applications by establishing virtual networks that utilize shared physical infrastructure. Many works in the literature have considered optimizing the allocation of on-demand slices, assuming the absolute availability of resources and their accurate load. However, accurately allocating future network slices remains challenging due to the error in load prediction, diverse Key Performance Indicators (KPIs), resource price variations, and the potential for over- or under-provisioning. This study presents a two-phase intelligent approach to address these challenges. The framework proactively predicts different slice loads while considering prediction errors in optimizing future slices with varied KPIs in a cost-efficient manner. Specifically, our method utilizes historical load data per service and employs AI-based forecasts for service load prediction. Subsequently, it employs a Deep Reinforcement Learning (DRL) agent on O-RAN's virtual Control Unit (vCU) and virtual Distributed unit (vDU) to correct errors in prediction and optimize the cost of slice allocation based on service KPI requirements, ultimately pre-allocating future network slices at reduced costs. Through experimental validation against various baselines and state-of-the-art solutions, we demonstrate the efficacy of our proposed solution, achieving a notable reduction (37-51%) in the average cost of allocated slices while inquiring about (1.5-7%) of additional resources compared to the state-of-the-art.

**INDEX TERMS** Reinforcement learning, network slicing, load prediction, smart health, error-correction.

## I. INTRODUCTION

OVER the past two decades, wireless technology has become the dominant force in networking. The development of new-generation networks, like 5G and 6G, aimed at meeting the growing demands for communication and computation capabilities in various industries and sectors [1]. These networks have opened up new possibilities for applications with strict requirements, such as Ultra Reliable Low Latency Communication (URLLC), Massive Machine Type Communication (MMTC), and Enhanced Mobile Broadband Communication (eMBBC). The requirements for these applications/services vary and may include a focus on low latency, high reliability, low cost, minimal energy consumption or higher data rates. These varying performance requirements are defined by a set of Key Performance Indicators (KPIs) that need to be met for different services [2].

To meet the KPIs for diverse services, there is an increasing need for timely, efficient, and flexible decision-making in managing network infrastructure and available resources, as well as determining the best traffic routing. In order to improve network management, we no longer perceive networks as mere collections of hardware (i.e., Devices and cables). Instead, the concept of Software-Defined Networking (SDN) [3] was introduced to replace complex network devices with software running on commodity machines. SDN gained popularity with the introduction of Network Function Virtualization (NFV) [4], where SDN provides centralized control and facilitates easy configuration updates, while NFV emulates network functions such as routing and firewalls. Furthermore, Open Radio Access Network (O-RAN) promotes flexible interfaces and structured decomposition of RAN elements (e.g., Central unit (CU) and Radio intelligent Controller (RIC)), allowing efficient resource allocation and enhanced network programmability [5].

Today, various services request their needs as a graph of Virtual Network Functions (VNFs), specifying the necessary network functions and data routing between VNFs. To accommodate this, Network Slicing [6] was proposed to support multiple services with varying demands and requirements, such as latency and reliability. These slices are characterized by mutual isolation and can be independently controlled, managed, and created based on the varying demands of different services, effectively transforming a physical network into a set of virtual network slices.

However, designing accurate network slices for diverse services is a complex problem, since it involves allocating both computational and network resources while considering diverse KPIs, which turns the problem to be an NP-hard problem [7]. This calls for proposing many heuristic approaches that reserve fixed network slices for different services, which results in over- or under-provisioning of resources. Additionally, most of the current work relies on optimizing slices through various techniques or heuristics that lack support for different KPIs or assume the availability of on-demand resources, as in the collaboration between AT&T (as tenant) with Microsoft Azure (as the provider) [8], and others like Verizon and Amazon Web Services [9]. Moreover, it was previously demonstrated in [10] that prior resource acquisition using predictions will help reduce the tenant's overall costs than on-demand resource acquisition.

In contrast to prior studies outlined in Section II, which were limited to conventional traffic load forecasting for network slicing, static non-adaptive network slicing, solely resource-optimized slicing without routing (predicting optimized resources only e.g., [11]), and fixed resource pricing frameworks, this paper employs Artificial Intelligence (AI) techniques on multiple levels to devise a precise, economical, and adaptive network slicing strategy to provision the requirements of various services' future loads proactively. In the first phase of the framework, our solution begins by collecting per-service historical load records along with AI-based forecasted resource and KPI requirements of each slice. Then, in the second phase, an error-correcting and slice optimization Deep Reinforcement Learning (DRL) agent is deployed, which learns the prediction error distribution and services KPI requirements. After that, the agent intelligently pre-allocates various future network slices with a lower pricing, aiming to minimize the cost of creating end-to-end network slices for each service while adhering to KPI constraints to ensure quality of service (QoS).

Thus, in this paper, our contributions can be summarised as follows:

1) We first formulate the problem of network slicing as an optimization problem to minimize the cost of different slices through optimal routing and resource assignment given different KPIs and constraints. Then, we present a two-phase AI-based framework which is named Error-aware, Cost-effective, and Proactive Network Slicing Framework (ECP). ECP proactively provisions minimal-cost future network slices with a high degree of accuracy. Indeed, ECP proactively creates network slices in advance at a minimal cost, ensuring these slices are closely in sync with what would be considered optimal (assuming full knowledge of future loads and optimized using convex programming). The effectiveness of this alignment is evaluated based on three critical aspects: 1) the overall cost efficiency of the slices, 2) the precision in estimating and reserving the necessary resources, and 3) the extent to which the slices meet predefined KPIs. The proposed framework takes into account the dynamic changes in network demand and pricing for each service, addresses future resource scarcity by allocating resources in advance, and considers various services' KPIs.
2) In the first phase of ECP, we develop a deep predictive model that forecasts hourly per-service load for two example healthcare services: Remote Surgery (RS) and Remote Monitoring (RM). This predictive model aims to provide accurate load predictions for these specific services. In the second phase, we introduce an error-correcting Deep Reinforcement Learning (DRL) model. This DRL model leverages the per-service hourly predicted load and prediction errors to create dynamic, cost-efficient, KPI-supported, and pre-allocated network slices. Notably, our DRL model has a unique capability: it learns from errors generated by the predictive model, enabling it to correct inaccuracies and optimize load predictions effectively. To identify the best-performing models for ECP, we conducted an extensive evaluation encompassing both the predictive and DRL phases. This comprehensive evaluation involved exploring multiple alternatives and conducting thorough comparisons against various benchmarks and baselines. Our goal was to assess the performance of these models in terms of total cost, the number of reserved resources, and adaptability to network changes.
3) Given that ECP can accurately estimate and allocate various network slices in advance, we have introduced the concept of dynamic resource pricing. This approach allows our system to take advantage of lower resource prices as the allocation moves further away from the utilization phase. However, a tradeoff is involved: as

we prioritize cost reduction, we may sacrifice some accuracy in our resource estimation, a tradeoff we aim to optimize.

4) Finally, we propose four variants of ECP, each incorporating a different technique to enhance prediction accuracy for precise optimization of network slices. These variants are then compared to the state-of-the-art approach and various baselines, with performance evaluated in terms of total cost, the number of reserved resources, and adaptability to network changes.

Finally, we remark that the proposed framework can benefit tenants (e.g., Hospitals) and network service providers (i.e., Telecom or cloud providers who provide physical or virtual resources to tenants). Indeed, tenants can benefit from leasing optimized slices in advance at discounted prices. Meanwhile, service providers will be able to plan ahead more efficiently, reduce the need for on-demand resources, and support a bigger number of tenants.

Following this introduction section, the rest of the paper is organized as follows: In Section II, we introduce a literature review. Section III covers our proposed system model and the problem formulation. Section IV discusses our proposed solution, whereas Section V presents the performance evaluation of our framework and discussion. Finally, Section VI concludes this paper.

## II. LITERATURE REVIEW

In this section, we discuss the related work regarding load prediction, dynamic pricing, network slicing allocation and optimization.

Over the past years, predicting patient load in healthcare entities has been a fundamental area of research in the medical field. For instance, accurately forecasting the daily number of surgeries in healthcare entities is crucial for optimizing staffing, room allocation, and other hospital-based resources. However, it's worth noting that inaccurate predictions, as observed in scenarios like patient counts and surgery volumes, can substantially escalate a hospital's overall operational costs [12]. Thus, many techniques have been adopted in the literature to tackle this prediction problem, using simple statistical models, such as the auto-regressive integrated moving average (ARIMA) and seasonal ARIMA (SARIMA) models. For example, ARIMA and SARIMA were used in [13] to know the surgical volume, while they were used in [14] to forecast the daily in-patients. Moreover, due to the non-linearity nature of the forecasting dataset, machine and deep learning models, such as Recurrent Neural Network (RNN) and its improved version Long Short Term Memory (LSTM), were also used in many works such as [15] for emergency patients prediction.

Similarly, in networks, forecasting the volume of users manifests under load or traffic estimation. It is also defined formally as the prediction of inbound and outbound traffic at different levels of the network (e.g., Devices and links). The importance of such predictions is that they form the basis of different anomalies and security discovery techniques in networks [16]; they also help the satisfaction of service level agreement (SLA) [17]. In [18], the authors used reinforcement learning to predict the end-to-end real-time network traffic under a dynamically changing environment. Like medical load forecasting, predictive deep learning models (e.g., LSTM) have outperformed statistical models in network load prediction in different scenarios [19], [20]. A recent study by [21] used LSTM and Random Forest to predict network slicing KPIs compliance but did not address automated resource fine-tuning for varying resource needs, which we address. Additionally, many of the literature works, e.g., [22], [23] focused only on optimizing the resources without the inclusion of the routing in their optimization, which we have incorporated.

Building upon the works above, this paper leverages traffic prediction to learn the daily network needs in order to optimize and design pre-provisioned, cost-effective network slices.

In addition to traffic forecasting, many works have focused on the VNF placement problem, which includes finding the location of VNFs and the number of computational resources allocated for any service. For instance, the authors in [24] showed how the placement problem could be modelled as an assignment problem that even a relaxed version of it is considered an NP-hard problem. Hence, a near-optimal solution is presented. Other works (e.g., [25] and [26]) focused on the connection between the placement of VNFs and the associated traffic. Moreover, some related works focused on predicting the network load to ease the challenge of VNF placement, such as [27] and [28]. While the latter focused on identifying the resources based on a forecasting process, the former used the demand's Spatio-temporal variation to reserve the least resources per VNF. In [29], the authors showed the effect of running multiple VNFs on a single node, which can cause a high amount of interference; thus, they created a mechanism to quantify the VNF interference and proposed a way to lazily migrate VNFs from one node to another which helped increasing the overall throughput. On the contrary, different works focused on improving the end-to-end delay and throughput. For instance, [30] focused on VNF placement and routing using network slicing while leveraging graph theory and accounting for different KPIs, showing near-optimal performance. Our work requires no restrictions on VNF placement as we utilize the current existing VNFs to build optimal slices.

While most of the above literature did not focus on using artificial intelligence (AI) to optimize a considerable number of network slices, other works have. This includes [28], which employed a Reinforcement Learning (RL) technique to forecast traffic volume, resulting in almost perfect placement of VNFs with the least expense. Additionally, [31] demonstrated an intelligent traffic management method to make QoS provisioning possible in SDN settings based on multimedia. The authors suggested an RL approach that chooses the best routing algorithm from a selection of centralized routing algorithms to optimize network return and enhance QoS provisioning.

Although many studies in the literature have explored fixed cost models for end-to-end resource provisioning and slicing admission control [32], [33], only a few have investigated the impact of dynamic pricing and how it relates to the timing of resource provisioning on the overall cost. A very recent work [34] was the first to illustrate a dynamic pricing scheme based on a ride-hailing pricing model for the admission of different slice requests. Indeed, that paper aimed to maximize the long-term profit (from the network company's point of view) by optimally accepting or declining different network slicing requests using RL. The paper, however, assumes that the service requester already gives the optimal number of resources for a network slice. On the other hand, our work does not assume that information is beforehand; thus, we create the optimal resource allocation for the services.

Moreover, it is essential to mention that many different resource providers have adopted computational resources dynamic prices, such as Amazon's AWS [35] and Huawei. However, network dynamic pricing was not introduced. We argue that the dynamic price of network resources (e.g., Bandwidth) is essential since a network slice would only function as expected if both computational and network resource requirements are fulfilled. This need is amplified when using network slices for medical services (e.g., Remote surgeries) where latency is critical and thus, reserving and guaranteeing part or all needed resources is essential. Therefore, there is a need to involve a changing price for the network resources like computational resources. It is also worth noting that very recent work by [36] showed another direction of dynamic pricing by engaging the network operators and their usage into a competitive resource pricing game through the usage of game theory and multi-agent reinforcement learning to provide competitive pricing for slicing users.

In light of the above work, our project creates an AI-powered framework for predicting the network demand for different services and customizing optimal network slicing accordingly. Additionally, we incorporate a dynamic pricing model to estimate the cost of the futuristic resources and a prediction error estimation model to account for load prediction errors. To our knowledge, this is the first work to consider load prediction with end-to-end network slice optimization considering dynamic pricing and the prediction error.

## III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, we present the proposed system model. First, we introduce terminologies such as Virtual Network Function (VNF), service, and the considered Key performance indicators (KPIs). Then, we present the problem formulation, combining different aspects altogether.

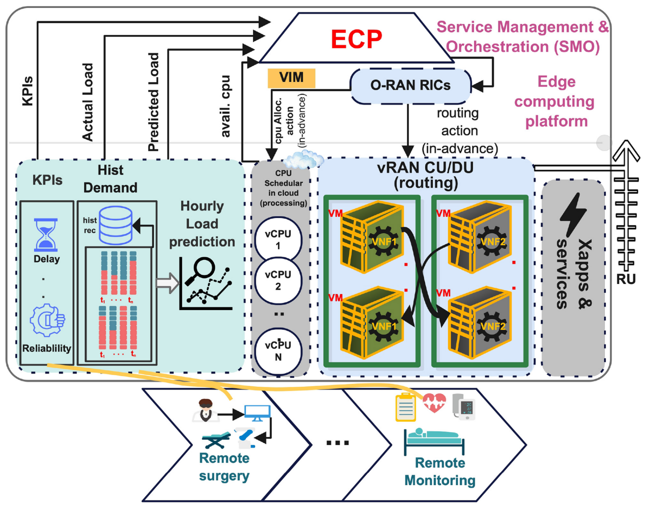


FIGURE 1. Considered System model, framing the solution in the Open RAN architecture.

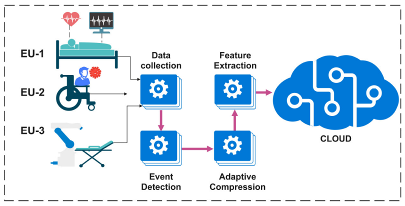
### A. SYSTEM MODEL

Figure 1 shows the considered model that is framed within the O-RAN 5G architecture. Our solution operates within the O-RAN architecture's service management and orchestration (SMO) layer. It takes input from various services/flows, including per-service KPI, actual and AI-predicted historical loads, available CPU resources, and the network's current state, which is provided by the edge computing platform. Then, utilizing the O-RAN intelligent controllers (RIC), we deploy a DRL agent to perform tasks on the virtual Control Unit (vCU) and virtual Distributed Unit (vDU). These tasks include configuring vBS functions, enabling per-service VNF routing, and collaborating with the NFV virtual infrastructure manager (VIM) to configure cloud-based CPU resources schedulers along the routing node according to the O-RAN specification. Without losing generality, we will focus on healthcare services as an example without limiting the kind of services our system can consider. Indeed, our focus will be on two different healthcare services, namely Remote Surgery (RS) and Remote Monitoring (RM) [37]. In what follows, we will explain different model details in our system.

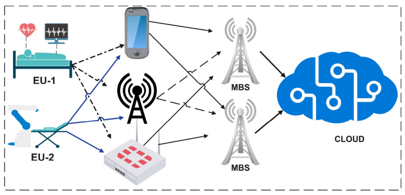
#### 1) SERVICE GRAPH AND VNF

To support the work of various services (such as remote monitoring), each service should be assigned a unique network slice to meet its tight QoS demands. Each service requires a sequential series of VNFs connected in a graph with a particular order (i.e., Service Function Chaining (SFC)) [30]. A VNF may run on single or multiple nodes of different capabilities (i.e., Different resource amounts). As seen in Figure 2a, VNFs may represent a variety of functions such as event filtering and feature extraction [38].

Each service has specific goals to achieve, and its performance can be measured by meeting the associated KPIs. It is also important to mention that according to the service, some KPIs may be prioritized over others depending



(a) Service graph



(b) Physical graph

**FIGURE 2. (a) The service graph for different medical applications, each block represents a VNF. (b) The Corresponding physical graph where the VNFs are running.**

on the service being provided. For example, in a remote surgery service where a patient is being treated remotely, the latency/delay KPI is prioritized over the cost KPI. On the other hand, telemonitoring may care more about cost KPI than latency as the monitoring is done over long periods. We also note that not all KPIs must be fulfilled per service [39]. Additionally, it's assumed that intermediary processing nodes will either reduce the data with a factor $\epsilon$ or keep it as it is. In other words, the data is not increased from one node to another.

### 2) COMPUTING AND NETWORK RESOURCES

By focusing on the vRAN CU/DU segment depicted in Figure 1, we observe the virtual graph alongside its corresponding physical graph illustrated in Figure 2(b). These graphs showcase the entirety of processing and communication resources available across different levels within the system. In Figure 2(b), the graph's vertices represent the different cloud's Extreme Edge Devices [40] (hereafter, referred to as nodes), while the edges indicate the connection between two nodes. Because network nodes $u \in \mathcal{U}$ differ in their computing resources (e.g., CPU and Memory), the max amount of resources of type $k$ in node $u$ is defined as $a_u(k)$. Moreover, a node possesses a processing delay, defined as $D_u$. Similarly, an edge corresponds to a specific link $y \in \mathcal{Y}$, with a transmission delay $D_y$.

Additionally, a collection of interconnected nodes and links, forming a complete route from start to finish, is denoted as a path $g \in \mathcal{G}$. The designation of a specific path chosen by a service $s$ is termed the service flow $f \in \mathcal{F}$. In our system, we define multiple services. Each service has only one choice of end-to-end path, (i.e., one flow per service), therefore the system has multiple flows.

Moreover, each link $y$ has a physical bandwidth limit $W_y$, hence, we define the link capacity constraint across all flows as in Eq. (1), with $r_{f,y}$ reflecting the data amount of a service flow $f$ passing through $y$.

$$\sum_{f \in \mathcal{F}} r_{f,y} \leq W_y, \quad \forall\ y \in \mathcal{Y} \tag{1}$$

**TABLE 1. Table of notations.**

| Notation | Description |
|---|---|
| $u$, $\mathcal{U}$ | Node: computing node that runs a VNF service, Set of all nodes. |
| $g$, $\mathcal{G}$ | Path: a group of nodes and links, Set of all paths. |
| $g^*$ | Denotes the selected path for a certain flow. |
| $y$, $\mathcal{Y}$ | Link: a connection between 2 nodes, Set of all links. |
| $s$, $S$ | Service, Set of all services. |
| $v$, $\mathcal{V}$ | VNF, Set of all VNFs. |
| $f$, $\mathcal{F}$ | Flow, Set of all flows. |
| $B(f, u, u+1)$ | Processing data of $f$ from node $u$ to $u+1$. |
| $N_g$ | Number of links in a path $g$. |
| $\sigma_s$ | Number of patients per service $s$. |
| $a_u(k)$ | Maximum amount of type $k$ resource in $u$. |
| $\eta_u(t)$ | Reliability of $u$ at time $t$. |
| $d_p(u)$, $d_p(f, g)$ | Processing delay at $u$, total processing delay of $f$ for $g$. |
| $D_{u,f}$ | Processing delay at node $u$ for flow $f$. |
| $c_u(t, k)$ | Cost of reserving $k$ resources in $u$ at $t$. |
| $\epsilon_{f,u}$ | Data percentage of flow $f$ that will be processed at node $u$. |
| $B(y, f)$ | The amount of data of flow $f$ passing through link $y$. |
| $D_u$ and $D_y$ | Processing delay caused by $u$, Transmission delay caused by $y$. |
| $D_{y,f}$ | Communication latency of flow $f$ passing through link $y$ . |
| $W_y$ | Link capacity/Bandwidth of $y$. |
| $\eta_y(t)$ | Reliability of link $y$ at time $t$. |
| $r_{f,y}$ | Amount of data in $f$ passing through $y$. |
| $r_{k,f,u}$ | Number of resources reserved of type $k$ at $u$ from flow $f$. |
| $\theta_y$ | Access delay for link $y$. |
| $d_n(f, g)$ | Network delay for $f$ passing through $g$. |
| $c_y(t)$ | Cost of data transmission in link $y$ at $t$. |
| $d_{f,T}$ | Total delay per flow. |
| $R_{T,f}$ | Reliability threshold for a given flow $f$. |
| $D_{T,f}$ | Total delay threshold for given flow $f$. |
| $S_{f,g}$ | Path selection indicator for $f$. |
| $o_{f,g,y}$ | Link selection indicator for flow $f$ passing through $g$. |
| $\vartheta$ and $\zeta$ | Prediction horizon and Input size. |
| $N$ | A general variable to indicate the number of services, inputs, and others. |
| $d_{curr}, h_{curr}$ | Current weekday and hour indicator. |
| $d_{count}, h_{count}$ | Maximum Number of days and hours used for training. |
| $\tau$ | A factor that represents the urgency of needing a resource. |
| $\mathcal{S}$,$\mathcal{A}$,$\mathcal{T}$,$\mathcal{R}$,$\gamma$ | MDP components: state, action, transition probability, reward, and discount factor |

TABLE 2. Table of acronyms.

| Acronym | Definition |
|---|---|
| AI | Artificial Intelligence. |
| ARIMA | Auto-Regressive Integrated Moving Average. |
| CU | Central Unit. |
| CVX | Matlab Software for Disciplined Convex Programming. |
| DRL | Deep Reinforcement Learning. |
| ECP | Error-aware, Cost-effective, and Proactive Network Slicing Framework. |
| KPIs | Key Performance Indicators. |
| LSTM | Long Short-Term Memory. |
| MDP | Markov Decision Process. |
| MBBC | Enhanced Mobile Broadband Communication. |
| MMTC | Massive Machine Type Communication. |
| NFV | Network Function Virtualization. |
| NN | Neural Network. |
| O-RAN | Open Radio Access Network. |
| PPO | Proximal Policy Optimization. |
| QoS | Quality of Service. |
| RAN | Radio Access Network. |
| RIC | RAN Intelligent Controller. |
| RM | Remote Monitoring. |
| RS | Remote Surgery. |
| RNN | Recurrent Neural Network. |
| SFC | Service Function Chaining. |
| SLA | Service Level Agreement. |
| SMO | Service Management and Orchestration. |
| SDN | Software Defined Networking. |
| SARIMA | Seasonal Auto-Regressive Integrated Moving Average. |
| URLLC | Ultra Reliable Low Latency Communication. |
| VN | Virtual Networks. |
| VIM | Virtual Infrastructure Manager. |
| vCU | Virtual Control Unit. |
| vDU | Virtual Distributed Unit. |
| VNF | Virtual Network Function. |

Similarly, the total reservation of resources of type $k$ from all flows at a particular node $u$, denoted as $r_{k,f,u}$ is bounded by the node's maximum capacity $a_u(k)$, therefore Eq. (2):

$$\sum_{f\in\mathcal{F}} r_{k,f,u} \leq a_u(k), \quad \forall\ u \in \mathcal{U} \tag{2}$$

### 3) KEY PERFORMANCE INDICATORS

Various KPIs can assist the requirements and performance of different services. Our system considers two service KPIs: end-to-end delay and end-to-end reliability.

For each service path, the delay is divided into two components: network delay and computation delay. While the network delay is caused by data transmission across multiple network links, the computation delay is generated by the computing edge nodes running different VNFs and processing every flow. As defined in Eq. (3), the sum of all edge/link delays $D_y$ along a certain path $g$ defines the total network delay of a service flow $d_n(f, g)$.

$$d_n(f, g) = \sum_{y\in g} D_{y,f} \tag{3}$$

Here $D_{y,f}$ is defined as $D_{y,f} = (\frac{B(y,f)}{r_{f,y}} + \theta_y)$. Where $\frac{B(y,f)}{r_{f,y}}$ represents the transmission delay for the data of flow $f$ passing through link $y$ and $\theta_y$ signifies the channel access delay that could occur during the data flow.

Moreover, in terms of processing latency, the VNF instances are modelled as queuing models of type M/M/1-PS under the processing paradigm in [30], which simulates how a multi-threaded program operates inside a virtual machine. We also note that different processing models can also be incorporated into our system. Thus, the processing latency per node $D_u$ for a flow $f$, denoted as $D_{u,f}$ is defined in Eq. (4):

$$D_{u,f} = \epsilon_{f,u}\frac{1}{r_{k,f,u} - r_{cpu}(u)\ \epsilon_{f,u}B(f, u)} \tag{4}$$

Such that $\epsilon_{f,u}$ depicts the percentage of data that will be processed at a VNF $v$ hosted on $u$ and $\epsilon_{f,u}B(f, u)$ is the processed traffic at node $u$. Ideally, $\epsilon_{f,u}$ will equal 1 where no compression or data extraction is done to the flow data. During our simulations, $\epsilon_{f,u}$ was set to 1 as the VNF operations included did not perform any data compression to the different flows. $r_{k,f,u}$ is the number of processing resources in $u$ per flow $f$ and $r_{cpu}(u)$ is the rate at which the CPU can execute instructions at node $u$. Summing all edge nodes' processing delay $D_{u,f}$ within a path $g$ gives us $d_p(f, g)$:

$$d_p(f, g) = \sum_{u\in g} D_{u,f} \tag{5}$$

It is worth mentioning that the allocated computing resources play a more prominent role in Eq. (5) than any network and storage resources. Thus, the allocated CPU provides an extra degree of flexibility to the trade-off between cost and performance. A high number of CPUs will reduce processing time but will increase costs, unlike other resource types (e.g., Storage space), where additional quantities would not affect the delay. Thus, the total delay per flow $d_{f,T}$ can be calculated as the summation of both network and computational delay $d_{f,T} = d_n(f, g) + d_p(f, g)$ and therefore each flow should abide by its end-to-end delay constraint $D_{T,f}$ which is given by Eq. (6):

$$d_{f,T} \leq D_{T,f} \tag{6}$$

### 4) RELIABILITY

To ensure high-quality network slices, it is crucial to consider the reliability of the intermediary nodes and links that connect the VNFs. Since network slices typically consist of multiple VNFs hosted on different nodes, the failure or drop in performance of any of these intermediary nodes or links can negatively impact the overall performance of the network slice. Therefore, it is essential to consider reliability as one of the key metrics when designing and deploying network slices.

To simulate a realistic state of links and network nodes, each node $u$ and link $y$ respectively have reliability parameters $\eta_{u(t)}$ and $\eta_{y(t)}$, which indicate the node's or link's ability to operate successfully at time $t$. When calculating the reliability of a single path $g$, which is a group of nodes and links, we need to consider the reliability of each intermediary node $u$ and link $y$. Therefore, the reliability of a path is calculated as the multiplication of the individual reliability of each node and link along that path. The resultant of the multiplication should abide by the system's reliability threshold $R_{T,f}$, which ensures that the overall reliability of the network slice meets the required standard as seen in Eq. (7):

$$\prod_{u\in g}\prod_{y\in g}\eta_u(t)\cdot\eta_y(t)\geq R_{T,f} \tag{7}$$

## B. PROBLEM FORMULATION

Our network slicing system's ultimate purpose is to provide complete and optimal in-advance network slices that fulfil the KPIs required by various services given their estimated network load (from the predictive phase, as will be mentioned later) with minimal cost. This functionality utilizes the intelligence of an intelligent DRL-based framework installed on the O-RAN's RIC that can estimate the service loads and reserve network slices for multiple services (i.e., both computation and network resources). The major steps required by our system can be summarised as follows:

- Collecting the different required KPIs from multiple services (e.g., Delay).
- Estimate the load for different services (based on the predictive models from the first phase).
- Reserve the end-to-end path with the required services' virtual network and computational (which can be reserved from a gateway cloud provider) resources.

Since the main goal is to optimize the cost of all reserved slices, it is essential to note the different costs. Estimating the total per-slice cost can be presented as the summation of two different costs $c_u(t,k)$ and $c_y(t)$ where $c_u(t,k)$ is the price per unit resource $k$ at node $u$ (processing cost) at a particular time $t$. $c_y(t)$ is the fee for transferring data unit per time unit on a link $y$ (network cost) at time $t$.

Following the model described in [34], $c_u(t,k)$ and $c_y(t)$ values can be defined as in Eq. (8):

$$\begin{aligned}c_u(t,k)&=\tau\cdot d_\rho\cdot c_u(k)\\ c_y(t)&=\tau\cdot d_\rho\cdot c_y\end{aligned} \tag{8}$$

Where $\tau$ is the urgency of the resource and is calculated as the logarithm (base 2) of the number of days until reserving the resources $N_d$ (i.e., $\tau=\frac{1}{\log_2(N_d)}$). $d_\rho$ represents the duration for reserving the resources. We simulate a real scenario and consider varying reservation times for each slice based on the service requirements. For example, a service like remote surgery only operates for five hours daily, so the service duration is limited to 5 hours. It is important to note that $N_d$ has a maximum fixed value of 8 to prevent excessive reductions in resource prices when users reserve resources in advance.

Now that we have explained the related KPIs and the primary goal of our optimization problem (cost minimization of slices), we explain the problem formulation in the following. Whenever our system receives the estimated loads from the predictive model (Explained in: Section IV-A) along with each service KPI requirements, it starts solving the following problem **P** allocating the necessary path and virtual computing resources along the path as follows:

$$\text{P:}\quad \min_{S_{f,g},a_u(f,k)} (C) \tag{9}$$

$$\text{s.t.}$$

$$S_{f,g}\big[d_n(f,g)+d_p(f,g)\big]\leq D_{T,f},\quad \forall f\in\mathcal{F},\forall g\in\mathcal{G} \tag{10}$$

$$\prod_{u\in g,y\in g}\eta_u(t)\eta_y(t)\geq S_{f,g}\cdot R_{T,f},\forall f\in\mathcal{F} \tag{11}$$

$$\sum_{f\in\mathcal{F}} o_{f,g,y}\cdot r_{f,y}\leq W_y,\quad \forall y\in\mathcal{Y}, \tag{12}$$

$$B(f,u,u+1)=\epsilon_{f,u}\cdot B(f,u-1,u),\quad \forall u\in\mathcal{U},\forall f\in\mathcal{F}, \tag{13}$$

$$\sum_{y\in g} o_{f,g,y}=N_g\cdot S_{f,g},\quad \forall f\in\mathcal{F},\forall g\in\mathcal{G}, \tag{14}$$

$$\sum_{g\in\mathcal{G}} S_{f,g}=1,\quad \forall f\in\mathcal{F}, \tag{15}$$

$$\sum_{f\in\mathcal{F}} r_{k,f,u}\leq a_u(k),\quad \forall\ u\in\mathcal{U}, \tag{16}$$

$$o_{f,g,y}\in\{0,1\},\quad \forall g\in\mathcal{G},\ \forall f\in\mathcal{F},\forall y\in\mathcal{Y}, \tag{17}$$

$$S_{f,g}\in\{0,1\},\quad \forall g\in\mathcal{G}\ \forall f\in\mathcal{F} \tag{18}$$

where the cost objective $C$ is defined as,

$$C=\sum_f\sum_g S_{f,g}\cdot\left[\sum_u\sum_k c_u(t,k)a_u(f,k)+\sum_{(y\in\mathcal{Y})} c_y(t)B(f,y)\right]$$

Our optimization problem **P** aims to allocate the necessary resources per service flow and traffic route to reduce the overall cost while satisfying different KPIs for all different services. Problem **P** contains two different decision variables, namely $S_{f,g}$ and $a_u(f,k)$. While the former represents a binary selection of a path $g$ for a flow $f$ (i.e., the end-to-end path with all links along that path), the latter illustrates the number of computational resources per each node's VNF along that path. The cost function, $C$, shows these decision variables' impact on the system's overall cost.

A set of constraints has been defined to guarantee the satisfaction of all different KPIs. Firstly, constraints (10) and (11) guarantee the satisfaction of delay and reliability, respectively. Secondly, constraints (12), (16) and (13) ensure that maximum physical link bandwidth, max computing resources per node and flow conservation are not violated. Thirdly, constraints (14) and (15) together dictate the

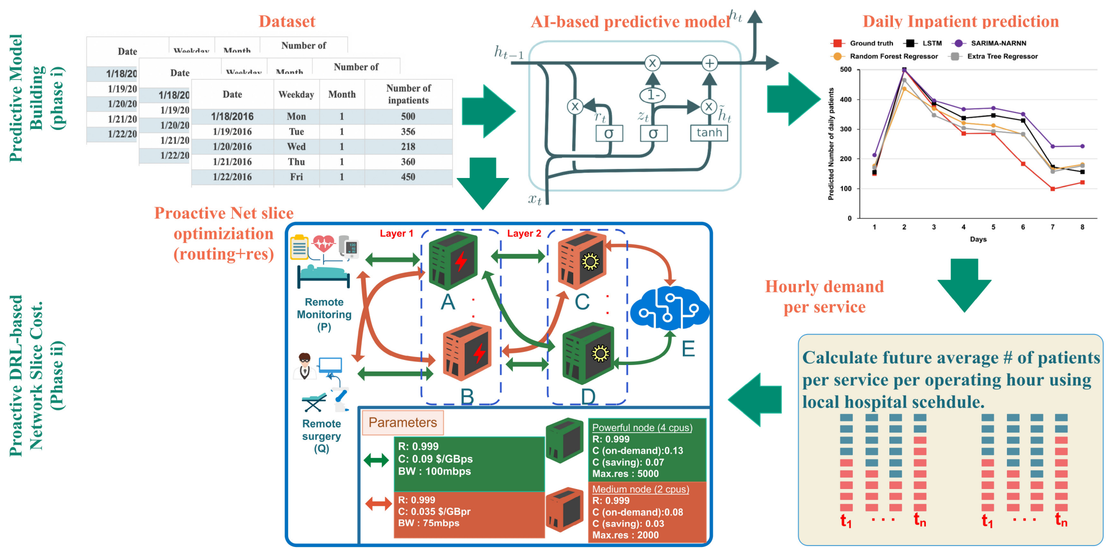


**FIGURE 3.** The proposed solution ECP consists of two phases illustrated on the two rows of the graph.

selection of all links ($N_g$ indicates the number of links of a path) along the selected path $g$ and only choosing a single path per flow apiece. Finally, the constraints (17) indicate the binary choice of all links along the chosen path and (18) narrows the choice of only one path per flow.

## IV. THE PROPOSED SOLUTION (ECP)

In this section, we will discuss the design of our Error-aware, Cost-effective and Proactive Network Slicing Framework (ECP), as illustrated in Figure 3. Our system consists of two main phases. The first phase estimates future load, while the second phase allocates minimized-cost network slices according to the estimated load and respecting different KPIs as presented in **P**.

### A. PREDICTIVE MODEL (PHASE 1)

To have accurate network slicing per service, each service network's demand must be precisely known. Thus, we opt to accurately forecast the number of daily inpatients and assess different techniques as statistical, machine and deep learning techniques.

To test our predictive pipeline, we adopt a sample dataset. The dataset has been collected from [14]. It illustrates the number of daily inpatients for a large hospital in China for about 39 weeks. We adopted and modified the dataset to fit our use, as will be described next.

After acquiring the data, several steps were done to pre-process and fit the data to our needs, which will be discussed next.

Firstly, the dataset shape has been modified to account for different types of techniques/models but with the same amount of records in each (35 weeks of data for training and four weeks for testing). For example, we have used a window size of 1-7 days for the statistical and machine learning models to forecast future loads. A similar technique has been used with deep learning models by using the dataset's sequential characteristics.

Secondly, we perform three sets of predictive models to seek the best model from all of them for predicting the daily load to be used in the next phase (Load optimization, Section IV-B). The sets of models used can be summarized as follows:

- The first set of the predictive model is the window-based statistical and simple machine learning models. This set includes Random Forest regressor [41], Extra Tree regressor [42], ARIMA [43], SARIMA-NARNN [14], and Long short term memory (LSTM)-based model (only 3-day window-based model) [44].
- The second set of models consists of deep single-step sliding-window-based LSTM models, each with a different window size ranging from 1 to 5 days. These models produce a single upcoming value from a sliding window of $N$ values. This set of models is widely used in literature (e.g., [45]) and was selected to evaluate the performance of various sliding window sizes $N$ in the given scenario.
- The third set envisions the set of Deep multi-step forecasting LSTM models, which takes $N$ values to predict $M$ values in the future. In this case, we test models with varying prediction horizons $\vartheta$ (e.g., 7-$\vartheta$, and 21-$\vartheta$) or changing input size $\zeta$ (e.g., $\zeta$-7 and $\zeta$-14) where every configuration is of the form $(\zeta - \vartheta)$. For example, the (7-1) configuration would represent an input of seven days of values to predict the next value. The (35-7) model would indicate the input of 35 days to forecast seven values of the upcoming seven days. This model set was chosen as it is widely used in the literature.

The performance comparison of these models will be presented in Section V.

Thirdly, after estimating the number of daily inpatients from the aforementioned models, we performed the following steps to get the number of hourly patients prepared for the next phase:

1) Using the predicted daily number of inpatients, we assumed a percentage of the daily inpatients to be used in the two healthcare services. For example, in our case, this percentage was set to 15%, divided into 10% and 5% for remote monitoring and remote surgery services, respectively.
2) According to a local hospital working schedule, each service's daily number of patients has been further divided into hourly inpatients (i.e., The 5% of inpatients for remote surgery was divided by the number of daily hours specialized for surgery). In our case, we had 5 hours of daily surgery and 12 hours of remote monitoring.
3) For each projected number of patients per service, the total expected data amount per hour was estimated as the multiplication of the predicted number of patients by each service required bandwidth. For remote surgery, we assumed transmission of a high-definition video of 5 Mbps [46], whereas, for remote monitoring, we assume light traffic of 0.5 Mbps.

## B. NETWORK SLICE CREATION AND OPTIMIZATION (PHASE 2)

After obtaining the predicted hourly network demand per service from the previous phase, this phase is focused on allocating a minimal-cost and adequate network slices for different services, which includes the optimal allocation of communication and computational resources for the different considered services while taking into consideration the different constraints to be respected (As in Eq. (9)) and the nature of the resources price fluctuation. We also consider and correct the prediction error from the previous phase.

Since network demand constantly changes, we opt to use a dynamic optimization approach. Because of its ability to adapt to different service requirements and manage highly complex environments, with low complexity [47], [48], Deep Reinforcement Learning (DRL) was used as the primary method of optimizing network slices. In our case, DRL was used to ensure the optimal choice of paths and resources while the state of the network might be unstable (e.g., When some network links fail). DRL was also used to learn and rectify the errors from our prediction phase.

Unlike predictive machine learning areas, which focus on predicting a value or a class, DRL is a subtype of machine learning which focuses on intelligently interacting in an environment for optimal behaviour. Moreover, DRL does not use a training dataset to train the model. Instead, it generates similar data by interacting with the environment and getting rewards or penalties that illustrate the action's goodness performed by the agent.

Because learning in an environment with many states is complex, DRL models the learning process as a Markov Decision Process (MDP). An MDP consists of the 5-element tuple, which is $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ where the agent is constantly monitoring the state $\mathcal{S}$ of an environment and performing an action $\mathcal{A}$ to obtain a reward $\mathcal{R}$ that has been discounted by $\gamma$ plus the new state $\mathcal{S}'$. When an action is performed, a state is changed with probability $\mathcal{T}$. After successfully training the agent for many episodes, it will be able to successfully map states to actions that result in the largest cumulative reward, referred to as a policy $\pi$, with the assistance of a learning algorithm. But before using DRL to solve our problem, we must first convert it to MDP, in which we will describe the environment, actions, and rewards.

### 1) STATE

To properly represent various states of our system/environment, the state space included three different elements. The first element is the set of the reliability of all links used in the physical graph $\{r_{l_y} : \forall y \in \mathcal{Y}\}$, this will be useful as it will indicate any problems with intermediary nodes, which might disable some end-to-end paths. The second element of the system state is the set of the predicted number of patients per service $\{\sigma_s : \forall s \in S\}$. This information is essential for the decision-making process in the system, as it gives the agent an idea of the amount of resources that need to be allocated per service. Additionally, the state includes a counter that tracks the current weekday $d_{curr}$ and hour $h_{curr}$, allowing the system to take into account the current time and adjust its operations accordingly. Altogether, forming the state space per timestep $\mathcal{S}_t$ as in (19):

$$\mathcal{S}_t = \left(\left\{r_{l_y} : \forall y \in \mathcal{Y}\right\}, \{\sigma_s : \forall s \in S\}, d_{curr}, h_{curr}\right) \tag{19}$$

### 2) ACTION

Prior to any action, an exhaustive list of all possible paths is given to the DRL agent. Then, at each timestep (in our case, an hour), the DRL assigns paths to services based on the path encoding variable $p_l$. For instance, assuming we have only two services, a value of $p_l = 3$ indicates $[0, 3]$, which assigns the first and fourth paths for the first and second services, respectively. This representation of path selection was used to reduce the number of variables in the action space. Moreover, the agent will allocate the necessary intermediary processing resources along each selected path $g^*$ per service flow $f$. Thus, the action space is represented as in (20):

$$\mathcal{A}_t = \left(p_l, \left\{r_{k,f,u} : \forall u \in g^*, \forall f \in \mathcal{F}\right\}\right) \tag{20}$$

### 3) REWARD

We reiterate that our agent undergoes training to choose the optimal slice for each service ahead of time. This slice comprises both the paths and the allocation of resources along those paths. During training, the agent ensures adherence to all previously discussed KPIs, utilizing knowledge of the environment's state. Importantly, this training occurs without access to the actual load data for any of the services. Therefore, our training entails providing the system with load predictions and improving its decision-making through an accurate rewarding function. Within our reward function, we evaluate the agent's performance on how well its combination of paths and resources fulfils the KPIs of the true load while simultaneously minimizing the total cost, relying solely on predicted load data.

Accordingly, the goal of the reward function is twofold: first, it directs the agent into reserving slices that adhere to various KPIs according to the actual demand while only knowing the predicted demand. Indeed, the agent only sees the predicted demand (see $\{\sigma_s : \forall s \in S\}$ in $\mathcal{S}_t$) and is penalized based on the actual demand $\{\sigma_s^* : \forall s \in S\}$ that is available only at training phase. This will enable the agent to understand the error distribution within the forecast model over the prediction period.

Second, out of all slices that adhere to the KPIs, it instructs the agent to find the least-cost set of slices for the set of services.

In light of the above, we have created the reward function $r_t$, which is divided into two parts: The first is punishing the agent for violating certain KPIs constraints for the actual demand data, defined in the first and second rows of $r_t$ as $c1$ and $c2$. Indeed, if the agent's action does not comply with each service's KPI (e.g., Delay $\mathbb{D}$ or reliability $\mathbb{R}$), the difference between the required (subscript *req*) and attained (subscript *att*) KPI multiplied by a scaling factor (e.g., $\alpha$) will be given as a penalty. For example, in delay KPI, the penalty is the difference between the required delay $\mathbb{D}_{req}$ and the attained delay $\mathbb{D}_{att}$ multiplied by a scaling factor $\alpha$.

Finally, if the agent finds the optimal configuration per all slices that minimizes the total cost and abides by all constraints for the actual load, it will be rewarded, as seen in the last row of $r_t$.

Thus, (21) represents the reward function:

$$r_t = \begin{cases} -\beta\Delta\big(\mathbb{R}_{att}, \mathbb{R}_{req}\big) & \textit{if } \mathbb{R}_{att} < \mathbb{R}_{req} \quad (c1) \\ -\alpha\Delta\big(\mathbb{D}_{att}, \mathbb{D}_{req}\big) & \textit{if } \mathbb{D}_{att} > \mathbb{D}_{req} \quad (c2) \\ (1, \ldots, 0) & \mapsto C[C_{min}, C_{max}],\ \textit{if } \neg(c1, c2)\ \forall f \in \mathcal{F} \end{cases} \tag{21}$$

Furthermore, we acknowledge that the minimum total cost of reserving multiple network slices is subject to change over time, given the hourly variations in demand leading to different hourly optimal configurations and costs. Hence, we have chosen to retain the lowest prices per load configurations obtained by the agent each hour, denoted as $C_{min}$. For example, suppose we have two services with hourly load requirements of 1500 Mbps and 2000 Mbps, respectively, and the agent found a minimum price of slices to be 150 USD. In this load configuration, our $C_{min}$ would be saved as $C_{min}[1500, 2000] = 150$ USD. After that, the agent will be rewarded a value between (1 and 0) on how close its current configuration's total cost $C$ is to the same configuration's best cost $C_{min}$ among all hourly loads. We also note that $C_{min}$ is constantly updated if a new least cost is found. This will direct the agent to always converge on finding a lower-cost set of slices. Moreover, if the agent violates any of the given constraints for any of the services, it will not receive the positive reward, hence the condition *if* $\neg(c1, c2)$.

**TABLE 3.** Simulation parameters.

| Parameter | Value |
|---|---|
| PPO learning rate | 0.0003 |
| PPO batch size | 128 |
| gamma | 0.97 |
| Predictor NN struct. | 2 LSTM layers (32 neurons) + 64 neurons |
| PPO Actor and Critique struct. | 2 layers of 64 neurons |
| $N$ | 2 |
| RM rate | 0.5 Mbps |
| RS rate | 5 Mbps |
| $\mathbb{D}_{req}$ | 10 (RM) & 8 (RS) |
| $\mathbb{R}_{req}$ | 0.999 |
| PPO num. of steps | 2048 |
| PPO optimizer | Adam |
| PPO's gae lambda | 0.95 |
| PPO's clip range | 0.2 |
| PPO's entropy coefficient | 0 |
| num. of daily hours for RM and RS | 12 & 5 |
| num. of KPIs used | 2 (Delay & Reliability) |
| num. of service types | 2 (RM & RS) |

#### 4) DRL ALGORITHM

Among different Deep Reinforcement Learning (DRL) algorithms, we chose to address our network slicing allocation problem by leveraging an efficient Deep Reinforcement Learning (DRL) method, specifically the Proximal Policy Optimization (PPO) algorithm [49]. PPO provides three different features, making it the most suitable DRL algorithm in our scenario. First, it inherits the ability of fast training from A2C by getting multiple trajectories from different parallel agents at once. This helps accelerate the training and convergence to the highest reward policy. Second, PPO adopts the trust-region policy update, which indicates that whenever an agent learns a new policy, the new policy will not be completely different from the current policy; this reduces the agent's divergence during the training. Moreover, having a trust region will enable finding a fine-tuned number of resources needed for intermediary nodes, consequently reducing total network slicing costs. Third, PPO supports high-dimensional actions, which supports the scalability needed when scaling up our problem.

### C. ECP ALGORITHM

Combining both phases, the full details of the ECP can be seen in Algorithm 1. In lines 1 and 2, our algorithm starts by initializing the inputs of the two phases of the system. For example, from the predictive phase, these inputs include the actual and predicted hourly load forecasts for different services $\{\sigma_s : \forall s \in S\}$. From the network slicing optimization phase, the maximum number of training episodes $E_{max}$, PPO's buffer size $B_{size}$, the max number of simulation hours

**Algorithm 1** ECP

```
1:  Input: Episode counter $E_{counter}$, Max num. of Episodes $E_{max}$, buffer size $B_{size}$, allocation map $M$, simulation hours per day $h_{count}$, different services hourly forecast $\{\sigma_s : \forall s \in S\}$ and actual loads. $\{\sigma_s^* : \forall s \in S\}$.
2:  Init PPO's Actor & Critiq NN $\theta_{\pi_0}$ and buffer $B$.
3:  while $E_{counter} < E_{max}$ do
4:    Reinitialize ENV.
5:    for each $d_{curr} \in d_{count}$ do
6:      for each $h_{curr} \in h_{count}$ do
7:        Form the per-timestep environment state $S_t$ using $d_{curr}$, $h_{curr}$ and the hourly forecast of all services $\{\sigma_s : \forall s \in S\}$ (using any prediction model or variants).
8:        Feed the environment state to the agent.
9:        Select $a_t$ tuple per service as in (20).
10:       Save chosen path's links and their intermediary nodes in $M$.
11:       if $\sum_{f \in \mathcal{F}} r_{f,y} > W_y, \ \forall y \in \mathcal{Y}$ then
12:         Split link bandwidth equally among demanding services sharing $y$ & update $M$.
13:       end if
14:       if $\sum_{f \in \mathcal{F}} a_u(f,k) > a_u(k), \ \forall u \in \mathcal{U}$ then
15:         Split node resources proportionally among demanding services sharing $u$ & update $M$.
16:       end if
17:       Using $M$, apply action $a_t$ from (20) and transfer data.
18:       Calculate reward $r_{t+1}$ using (21).
19:       Save trajectory $(s_t, a_t, r_{t+1}, s_{t+1})$ in $B$.
20:     end for
21:   end for
22:   for m mini-batch in $B$ do
23:     Compute rewards-to-Go $\hat{R}_t$.
24:     Compute the advantage estimate $\hat{A}_t^{\pi_k}$.
25:     Update Actor & Critiq NN
26:   end for
27:   $E_{counter}$ += 1
28: end while
```

per episode $h_{count}$, PPO's actor and critique neural network (NN) weights are added in addition to other utilities such as allocation map $M$, which records the links bandwidths and the number of resources held by each of the services across all links and nodes.

After ECP initialization, the training phase starts, and it's categorized into data collection (lines 4-20) and PPO's learning stages (lines 21-25). Data collection starts by iterating over a specific number of training hours per episode (e.g., 12 hours from 8 AM to 8 PM); these hours represent a timestep where there is a load for a set of services $\mathcal{S}$ which we need to optimize. Thus, we iterate over each hour $h$, and with the help of our predictive models, we get the hourly load forecast for all services $\{\sigma_s : \forall s \in S\}$ along with actual hourly loads and other variables forming the environment state (19). Presented by this environment state, our PPO agent samples an action $a_t$ (as in (20)) using the actor's policy neural network, which includes the choice of an end-to-end path and number of reserved nodes' resources per that path for each of the services. This information is stored in $M$.

Next, $a_t$ is checked to not to violate any network or computational constraints and rectify any error. For instance, the first test (lines 10-12) checks if the agent has a link $y$ that is allocated by multiple services with a total requesting bandwidth bigger than its limit $W_y$; in that case, the link's bandwidth is divided equally among the requesting services and the record is updated in $M$. The second test (lines 13-15) checks if the summation of allocated resources across each node is higher than its limit; if so, then the resources of the competing services will be divided proportionally among the demanding services (i.e., More resources are given to the service with higher demand) and $M$ is updated. After $a_t$ is checked and $M$ is updated, the agent performs the allocation, the reward is calculated as in (21), and the trajectory is saved in the buffer $B$.

After the data collection finishes, the PPO learning stage starts (lines 21-25), where a mini-batch $m$ is sampled from the buffer, and three main steps follow. The first is computing the reward-to-Go $\hat{R}_t$ per each trajectory in $m$. Next is computing the advantage estimate $\hat{A}_t^{\pi_k}$ that identifies how an action is better than others in a given state and accordingly updates PPO main policy (actor's NN) and value-function (critique NN) networks.

Considering the two phases within the ECP algorithm, we assess its testing computational complexity by identifying the maximum complexity between these phases. First, the initial phase leverages Long Short-Term Memory (LSTM) for prediction, entailing a computational complexity expressed as $O(T \cdot l_N \cdot l)$. Here, $T$ represents the sequence length, with experimental values ranging from 1 to 5. Additionally, $l$ denotes the number of LSTM layers, which is set to 2 in our case, and $l_N$ signifies the number of neurons per layer, set to 32 in our specific configuration. Secondly, in the second phase of testing, we employ the Proximal Policy Optimization (PPO) algorithm, where the inference complexity aligns with that of a typical neural network, characterized by $O(D \cdot X)$. In this context, $D$ represents the data dimension, encompassing 5 state variables. The complexity of the forward pass is defined by $X$, involving 2 layers, each containing 64 neurons. As a result, the overall computational complexity of the ECP algorithm remains constant.

## V. PERFORMANCE EVALUATION & DISCUSSION

In this section, we evaluate the performance of our proposed solution. First, we explain the environmental setup and then assess the system by testing each Phase individually, followed by a test of the entire system.

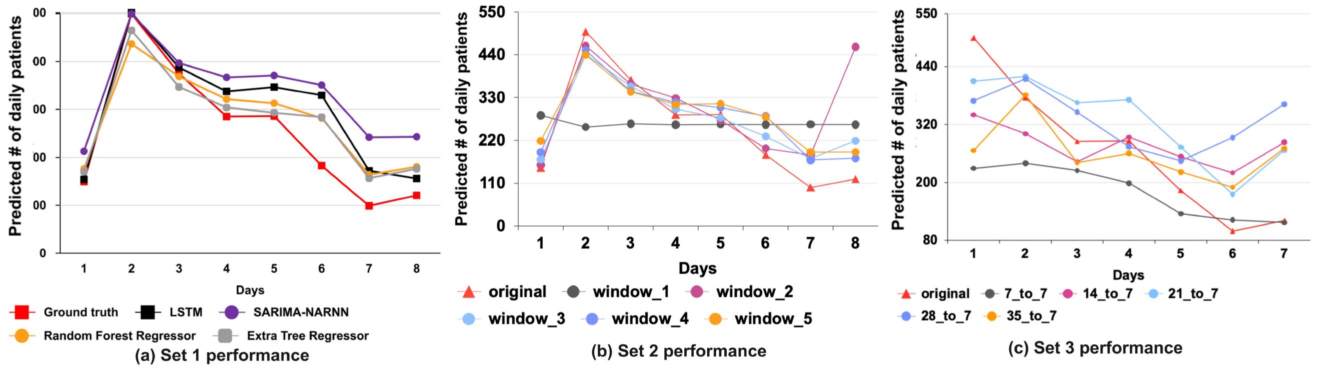


FIGURE 4. The prediction performance of Sets 1, 2 and 3 in predicting the next seven days' load. (a) shows the prediction of classical machine learning algorithms (b) shows the effect of different window sizes of the LSTM model on the prediction. (c) shows that changing input length affects the prediction output.

## A. ENVIRONMENT SETUP

To create a realistic environment, we assumed the presence of two healthcare services: remote monitoring and remote surgery. The RL will be trained in allocating the necessary hourly resources for the entire week for both services. Each service has a unique delay KPI: 8 milliseconds for remote surgery and 10 milliseconds for remote monitoring as in [50]. Furthermore, we have set a maximum required reliability of 0.999, which signifies that each service must ensure the highest level of reliability to prevent service failure. These values were chosen to emulate real-world scenarios.

The second row of Figure 3 shows the simulated scenario with the chosen values of nodes and links. We assume the existence of two layers of nodes (e.g., Nodes A & B are at layer 1). The nodes run the same VNF among the same layer, and each layer simulates different distributed VNF (e.g., One layer can represent AI-based data filtering VNF service). We also note that our model is applicable to any type of VNF. Additionally, each layer includes two nodes, each with a different capability. Two of the four nodes are assumed to be weaker nodes, while the others are assumed to be more robust nodes with more resources; this was done to simulate the real world, where different options for processing nodes might exist, for example, weaker nodes are cheaper but in many cases would not have the sufficient number of processing for the needed task. Similarly, we assume the existence of different bandwidth links with the trade-off of price and bandwidth (e.g., Strong links have higher bandwidth but are costly). Further, since we assume the capability of provisioning network slices ahead of time, the cost per computational resource and transfer of gigabytes of data for the on-demand and saving (ahead of time) options are also considered. Computational node costs were based on Huawei reserved instance costs with 2 and 4 CPUs for one hour of use.

## B. EVALUATING PREDICTIVE MODEL

After training the different sets of models (mentioned in Section IV-A), we evaluated their performance using the determination coefficient $R^2$ for the first simple models and

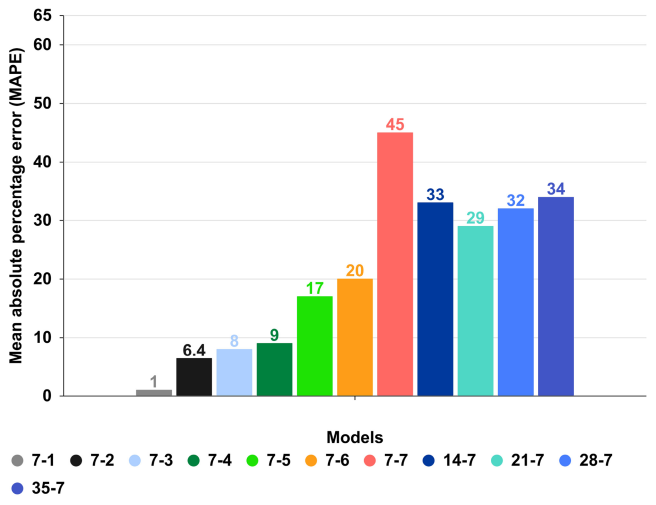
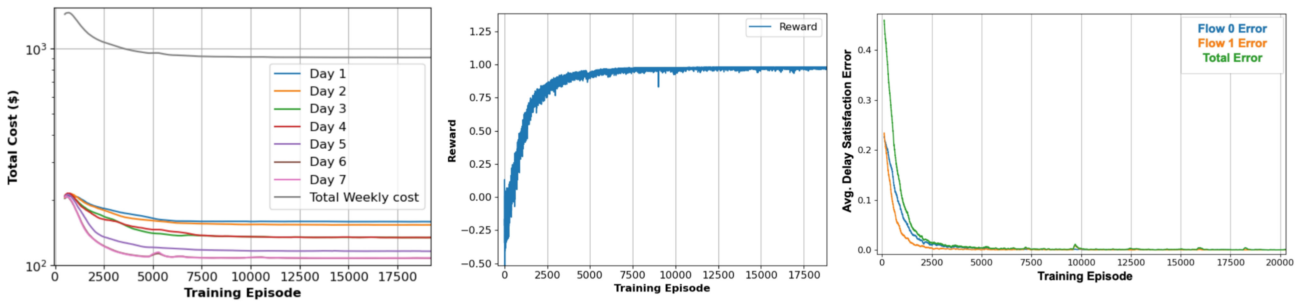


FIGURE 5. MAPE of different prediction input sizes and horizons, indicated by ($\zeta$-$\vartheta$). the graph shows how the prediction error % increases whenever the prediction horizon increases (7-1 to 7-7) and how increasing input sizes (7-7 to 35-7) can slightly improve the accuracy.

Root Mean Square Error (RMSE) for the second and third sets of models, as both were time-series models. The testing prediction was made to predict the load of the last seven days of the dataset.

Starting with Set 1, we present and visualize the results of various models versus the ground truth in Figure 4 (a). The findings indicate that the LSTM-based model had the highest accuracy compared to other statistical, machine learning, and fused models [14], with the highest $R^2$ coefficient. As a result, we evaluated variations of the LSTM model to find the best prediction model for facilitating the training of RL in the next Phase. In Figure 4 (b), the different variations of the window-based LSTM are presented with an optimal window size of 3. The performance of the models in Set 3 can also be seen in Figure 4 (c), where the prediction performance was the worst among all the models. Additionally, we calculated the Mean Absolute Percentage Error (MAPE) for predicting different $\vartheta$ with a fixed input of 7 days of readings (e.g., The performance of 7-1, 7-2, ..., 7-7). The MAPEs of all the models can be seen in Figure 5, which shows how



(a) Total cost convergence of RL for different hourly slices per future days along with their total cost

(b) Reward convergence of RL

(c) The average delay satisfaction error convergence.

**FIGURE 6.** Exp 1: Convergence behaviour of our RL-based solution (a) cost convergence, (b) shows reward convergence, and (c) shows the delay error satisfaction convergence.

the prediction error % increases as the prediction horizon increases. This prediction error will be used in a later variant of our solution.

### C. EVALUATING DRL-BASED NETWORK SLICING

This section aims to evaluate our DRL-based network slicing (second phase only) against other baselines. The input of this Phase is the predicted load (from the first Phase Section IV-A) along with the actual load, which is taken as a guideline for the agent action, not included in the environment state. On the other hand, the output is the list of minimal-cost hourly network slices for seven days of the services that will abide by different KPIs. The created slices are then reserved by each service operator ahead of time.

To test the second Phase of our approach, it was necessary to obtain predictions of network traffic demand from one of the prediction models developed in the first Phase. We used the 7-7 predictive model from Set 3, which takes seven previous inputs to generate seven predicted loads. While this model was not the best performing in prediction accuracy, we chose it also to showcase the ability of deep reinforcement learning (DRL) to correct inaccuracies in the prediction model if given access to ground truth data. Secondly, the output of the predictive model, along with the actual load, was provided to our DRL, which will work on (1) reserving the minimal cost network slice via choosing the best paths and resources along these paths for two services (RS and RM). Moreover, given the actual load of the predicted values (as a constraint), the RL will also work on correcting the prediction values by reserving the needed computational and network resources that abide by the real constraint, only seeing the predicted load as a guideline. This works by adapting to the prediction error distribution seen during the training period.

In this Phase, the list of conducted experiments can be summarised as follows:

- In the first experiment, we assess the agent's learning by viewing the convergence of the total cost for the selected network slices per the prediction duration for.
- In the second experiment, we compare our RL-slicing against the optimal (CVX-based optimizer) and maximum-fixed demand with saving prices and on-demand pricing schemes.
- In the third experiment, we evaluate the adaptability when the number of network service patients suddenly increases twice (simulating a pandemic or an emergency). The exploratory nature of RL will help avoid problems when the network changes unexpectedly. This shows that in situations where the load increases unexpectedly, RL could still adapt to the environmental changes by increasing the number of resources or choosing a different path that satisfies the demand.

#### 1) FIRST EXPERIMENT

After training the agent for about 20 thousand training episodes, we can see that our network-slicing solution learns the minimal-cost network slicing for the hourly demand of a whole week as in Figure 6 (a). The figure shows the aggregated cost of 17 network slices per day for two services (12 for RM, 5 for RS) and the total costs for reserving the total number of slices per week. Moreover, in Figure 6 (b), we show the reward convergence, which illustrates how the RL satisfied all the constraints and optimized the total cost. Finally, Figure 6 (c) is one of the most critical graphs, which shows how our solution was able to correct the prediction error resulting from the low-accuracy model (of phase 1) by following the error distribution between the ground truth and predicted values. It is important to mention that we do not always assume the existence of ground truth; in such cases, we will only follow certain solution prediction variants (e.g., prediction + some error), as discussed in the next section.

#### 2) SECOND EXPERIMENT

To determine the effectiveness of our RL-based method for allocating the least expensive hourly slices, we compare it to two baselines. The first is the on-demand optimal (CVX-based) baseline, inspired by [51], which assumes a perfect knowledge of the demand. This method works by convexifying the problem via enumerating the list of

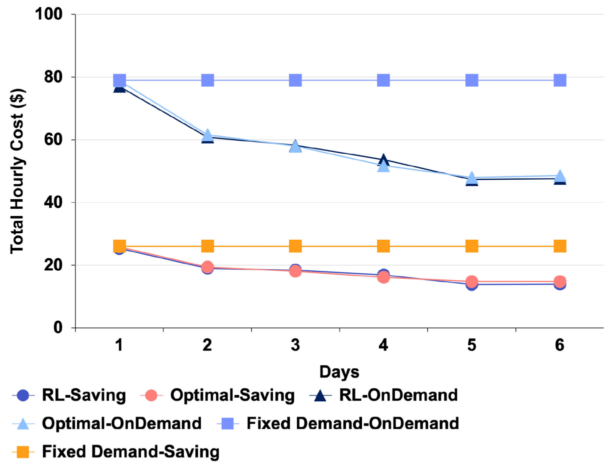


FIGURE 7. The figure demonstrates the effectiveness of our RL-based solution compared to the optimal and fixed-demand baselines under two pricing schemes (on-demand and savings).

paths. Then, for each path, it runs a convex optimizer that chooses the cost resources per the nodes along the paths that satisfy the constraints and saves the solution with its cost in a candidate set. Then, the least cost combination of the candidate set is chosen to be the solution. The second baseline is a maximum-based strategy that assigns the least-cost fixed slice based on the highest hourly demand observed in the past. Additionally, to highlight the effect of dynamic pricing on each solution, we compare two pricing schemes: on-demand and savings, which refer to the cost of reserving resources on-demand and in advance. The results displayed in Figure 7 reveal that the performance of RL and the optimal solution were very similar in allocating the optimal hourly load over a six-day period in both pricing criteria. In some instances, RL resulted in lower prices than the optimal solution. This could be due to the better refinement of resources that RL can achieve compared to the CVX-based solution. Moreover, as seen in the same figure, the fixed-demand policy can result in high hourly costs compared to other solutions.

### 3) THIRD EXPERIMENT

The illustration in Figures 8 demonstrates the flexibility of the RL solution. The experiment involved waiting for the agent to reach a stable state and then doubling the number of patients for both services. The outcome shows that the RL agent can adjust to the changed demand by allocating additional resources and attaining stability after roughly 500 iterations. Figures 8(a) and 8(b) depict the re-stabilization in terms of cost and reward, respectively.

## D. EVALUATING ECP PERFORMANCE

In this section, we assess the overall system's performance, incorporating both predictive and RL-based slicing elements. Unlike the evaluations performed in the prior section, which leveraged actual load data to steer the RL toward precise resource allocation, this assessment assumes that no accurate load information is obtainable and that all predictions are

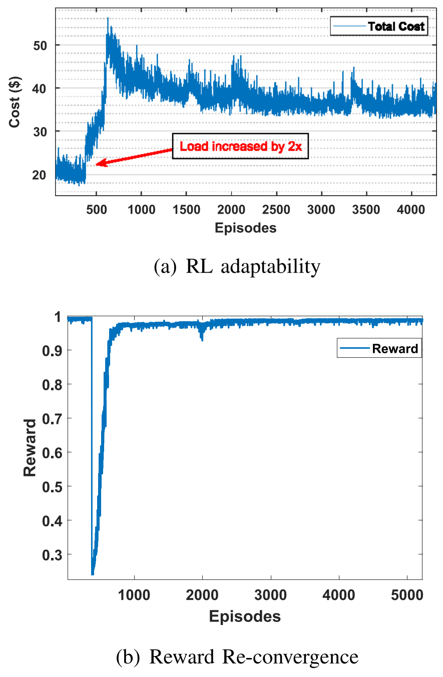


FIGURE 8. Experiment 2: The Adaptability of the ECP (Phase 2) in Response to a Sudden Increase in Service Demand. Fig.(a) illustrates the cost re-convergence, while Fig.(b) shows the reward re-convergence.

focused on forecasting future weeks for which information is yet to be available.

Resource allocation depends on daily inpatient projections. Matching or surpassing these projections ensures sufficient resources for slices, while projecting fewer patients can lead to unmet KPIs and performance degradation. Hence, to better estimate the real load, we have designed multiple variants of our solution, which works on modifying the predicted value from our predictive model and then optimizing the resources using the second Phase. The first and second variants modify the predicted model by adding and subtracting a constant average error percentage from the predicted value. The constant error used in this case was the maximum prediction error given to the DRL agent. The third variant, namely "Predicted + Dynamic Error", incorporates the weekday error % distribution learned by the RL during the training of the agent in Phase (ii) (see Section V-C). i.e., the difference between the actual and predicted load of many weekly weekdays. Averaging that error in addition to the model's prediction gives us the fourth variant (Predicted + Avg. Dynamic Error).

To better understand the system's performance, we test our system with the aforementioned variants for the duration of 4 sequential weeks. We have chosen the best-performing model among the different models tested in phase 1, the LSTM window-based model of 3 days. In Figures 9, we can see the estimations of the different variants for predicting the daily inpatients against the actual load (herein called Original). After getting the weekly load estimations per different variants, we train our DRL-based cost-optimization on it. The total weekly cost convergence of the different

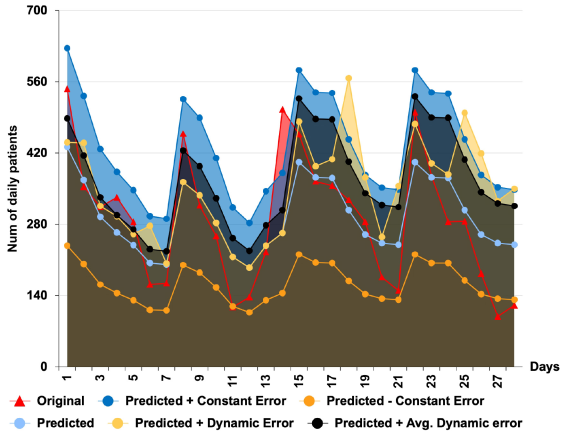


FIGURE 9. The load estimation of different ECP variants vs. the original load.

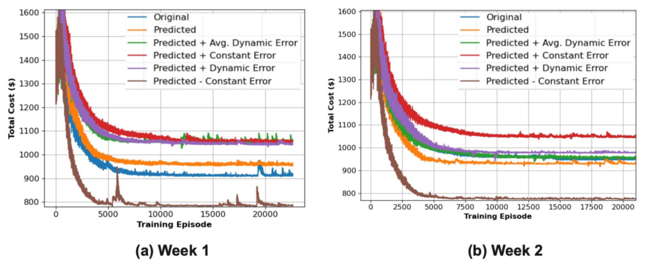
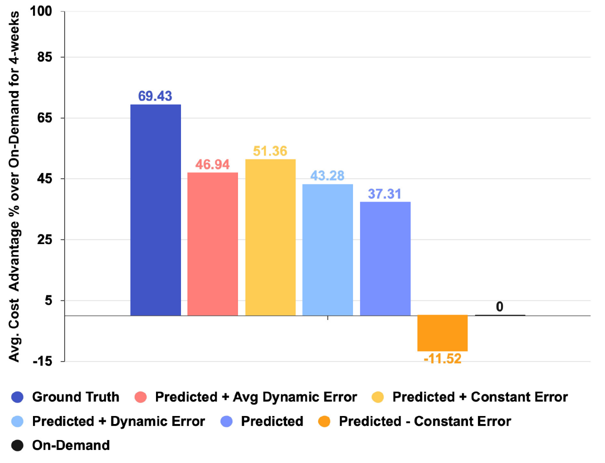
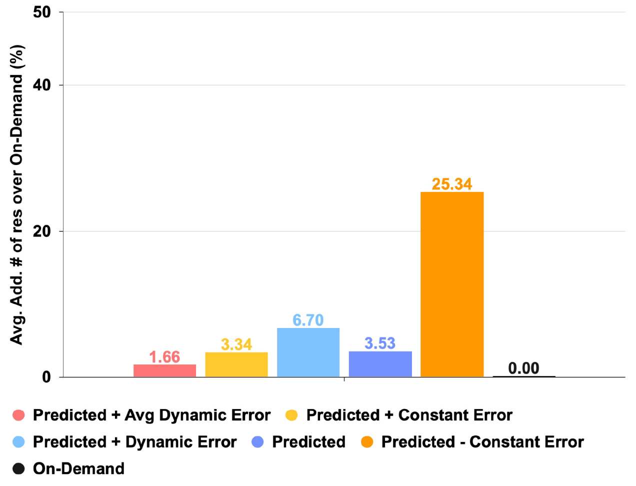


FIGURE 10. The convergence of various ECP variants over four weeks.



(a) The figure illustrates the Avg cost advantage % of ECP's various variants compared and the ground truth against the On-demand baseline, as described in [51].



(b) The average number of hourly resources reserved by ECP's solution variants compared to the ground truth and the On-demand baseline inspired by [51].

FIGURE 11. The convergence of various ECP variants over a period of four weeks.

ECP variants for the first two weeks (first two out of 4 testing weeks) can be seen in Figures 10. We note here that total cost converging on a same or lower cost value than the original demand from different variants does not indicate reserving the actual loads' slices with lower or same cost; it only shows that whatever estimated load has been successfully optimized by RL.

To accurately assess the cost of the different solutions, it is crucial to allocate the necessary resources on-demand, which was achieved by adding additional computational resources to the pre-allocated path of the affected service until all the services are satisfied. We then analyze the cost of each solution by calculating the average cost reduction percentage between the cost of each solution. Each test was repeated three times per week to obtain reliable performance measurements, and the average result is presented. The results in Figures 11 demonstrate that the different variants of our system had a significant cost advantage over reserving the slices on-demand with the variants predicted + constant and dynamic error on the lead. On average, the cost advantage was 37-51%, which is similar to the cost of accurately allocating the slices ahead of time, as shown by the Ground Truth. This remarkable performance is due to the accurate load estimation achieved by combining our predictive model with the error estimation methods. This approach formed an upper bound on the load that triggered our system to reserve almost all resources ahead of time, thereby reducing the cost.

An interesting observation from the same graph is that when the number of patients is vastly underestimated, the cost of rectifying the selected network slices can be higher than on-demand resource allocation. This is because a significant underestimation can result in choosing an inefficient path, demanding a significant amount of resources to correct. In contrast, on-demand allocation can deterministically choose the best route with fewer resources.

Additionally, we compare the average number of resources used by the different variants to the optimal number of resources used in the On-Demand solution. Figures 11(b) reveals that the different variants of our system require a greater number of resources, primarily due to overestimation in prediction. However, this increased resource usage is offset by the lower cost of pre-reserving the slices. Our observations also show that greatly underestimating the load

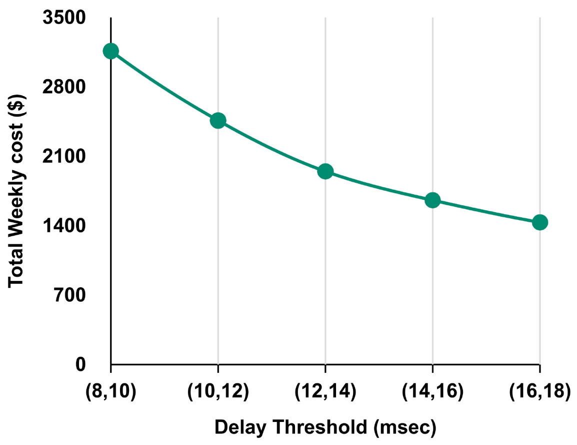


FIGURE 12. Increasing the maximum delay threshold for our two services, as shown on the x-axis, decreases the total cost.

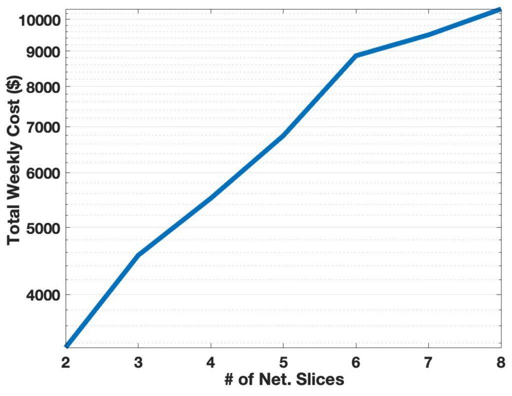


FIGURE 13. Scalability test with multiple edge nodes and varying number of services.

can result in a significantly higher resource reservation than on-demand allocation.

In Figures 12, we complement ECP's cost analysis by showing the tradeoff of choosing relaxed delay thresholds on the total cost for the two adopted services using an instance of pricing scheme (e.g., On-demand pricing). The figure shows that the more stringent the requirement, the more expenditure. Finally, to show the scalability of ECP, we incorporate a more significant scenario consisting of more edge nodes and multiple requesting services. Specifically, we considered three edge nodes per level and various slices with different KPIs. Results shown in Figures 13 shows how the system handled the services with ease and increasing cost due to the saturation of links.

## VI. CONCLUSION AND FUTURE WORK

In this paper, we introduced ECP, an artificial intelligence framework that predicts network demand and pre-allocates optimized network slices for various services with reduced costs. The solution consists of two phases. The first phase involves forecasting the daily demand, while the second phase uses DRL to dynamically optimize the various network slices while taking into account the various KPI constraints, prediction errors, and the dynamic pricing of resources. To accurately evaluate the system, we conducted a series of tests to validate each solution phase. This included testing multiple models for Phase 1 and evaluating convergence, effectiveness vs. optimal, adaptability, and correction for Phase 2. Since there is always uncertainty about future load, we introduced four different variants to enhance the predictive models. With the addition of our optimization reinforcement learning, we compared their performance against the state of the art. Our results showed that our system had a superior ability to allocate lower-cost future slices, with an average improvement of 37-51% compared to the state of the art, using only 1.5-7% additional resources. In forthcoming endeavours, we aim to extend our framework by enhancing its scalability through the implementation of distributed Multi-Agent Reinforcement Learning (MARL) in routing and resource allocation. Using the MARL approach, each network slice will be managed by a dedicated agent tasked with predicting the service's load and optimizing its operations altogether, thus enhancing the overall scalability.

## REFERENCES

[1] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network slicing for 5G with SDN/NFV: Concepts, architectures, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 80–87, May 2017.
[2] S. Kukliński and L. Tomaszewski, "Key performance indicators for 5G network slicing," in *Proc. IEEE Conf. Netw. Softw. (NetSoft)*, 2019, pp. 464–471.
[3] D. Kreutz, F. M. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
[4] N. M. K. Chowdhury and R. Boutaba, "A survey of network virtualization," *Comput. Netw.*, vol. 54, no. 5, pp. 862–876, 2010.
[5] 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., 2023.
[6] P. Rost et al., "Network slicing to enable scalability and flexibility in 5G mobile networks," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 72–79, May 2017.
[7] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the NFV service distribution problem," in *Proc. INFOCOM IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[8] (Microsoft Azure, Redmond, WA, USA). *Improving the Cloud for Telcos: Updates of Microsoft's Acquisition of AT&T's Network Cloud*, Mar. 2023. Accessed: Mar. 19, 2023. [Online]. Available: https://azure.microsoft.com/en-us/blog/improving-the-cloud-for-telcos-updates-of-microsoft-s-acquisition-of-att-s-network-cloud
[9] "Verizon evolves its 5G network—And accelerates its speed and reach—Powered by AWS." Amazon Web services. Nov. 2022. Accessed: Mar. 19, 2023. [Online]. Available: https://aws.amazon.com/blogs/industries/verizon-evolves-its-5g-network-and-accelerates-its-speed-and-reach-powered-by-aws
[10] R. Hu, J. Jiang, G. Liu, and L. Wang, "Efficient resources provisioning based on load forecasting in cloud," *Sci. World J.*, vol. 2014, Feb. 2014, Art. no. 321231. [Online]. Available: https://www.hindawi.com/journals/tswj/2014/321231/
[11] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "DeepCog: Cognitive network management in sliced 5G networks with deep learning," in *Proc. INFOCOM IEEE Conf. Comput. Commun.*, 2019, pp. 280–288.
[12] G. Luo, S. He, B. L. Stone, F. L. Nkoy, and M. D. Johnson, "Developing a model to predict hospital encounters for asthma in asthmatic patients: Secondary analysis," *JMIR Med. Inform.*, vol. 8, no. 1, 2020, Art. no. e16080.
[13] N. Zinouri, K. M. Taaffe, and D. M. Neyens, "Modelling and forecasting daily surgical case volume using time series analysis," *Health Syst.*, vol. 7, no. 2, pp. 111–119, 2018.

[14] L. Zhou, P. Zhao, D. Wu, C. Cheng, and H. Huang, "Time series model for forecasting the number of new admission inpatients," *BMC Med. Inform. Decis. Mak.*, vol. 18, no. 1, pp. 1–11, 2018.

[15] F. Kadri, M. Baraoui, and I. Nouaouri, "An LSTM-based deep learning approach with application to predicting hospital emergency department admissions," in *Proc. Int. Conf. Ind. Eng. Syst. Manag. (IESM)*, 2019, pp. 1–6.

[16] J. Ren, D. Zhang, S. He, Y. Zhang, and T. Li, "A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet," *ACM Comput. Surv.*, vol. 52, no. 6, pp. 1–36, 2019.

[17] A. R. Abdellah, O. A. K. Mahmood, A. Paramonov, and A. Koucheryavy, "IoT traffic prediction using multi-step ahead prediction with neural network," in *Proc. 11th Int. Congr. Ultra Modern Telecommun. Control Syst. Workshops (ICUMT)*, 2019, pp. 1–4.

[18] L. Nie et al., "A reinforcement learning-based network traffic prediction mechanism in Intelligent Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 17, no. 3, pp. 2169–2180, Mar. 2021.

[19] N. Ramakrishnan and T. Soni, "Network traffic prediction using recurrent neural networks," in *Proc. 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, 2018, pp. 187–193.

[20] A. Azzouni and G. Pujolle, "A long short-term memory recurrent neural network framework for network traffic matrix prediction," 2017, *arXiv:1705.05690.*

[21] J. S. Camargo, E. Coronado, B. Gómez, D. Rincón, and S. Siddiqui, "Design of AI-based resource forecasting methods for network slicing," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, 2022, pp. 1064–1069.

[22] C.-N. Nhu and M. Park, "Dynamic network slice scaling assisted by attention-based prediction in 5G core network," *IEEE Access*, vol. 10, pp. 72955–72972, 2022.

[23] J. Zhou, W. Zhao, and S. Chen, "Dynamic network slice scaling assisted by prediction in 5g network," *IEEE Access*, vol. 8, pp. 133700–133712, 2020.

[24] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2015, pp. 1346–1354.

[25] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. INFOCOM IEEE Conf. Comput. Commun.*, 2019, pp. 10–18.

[26] S. Agarwal, F. Malandrino, C. F. Chiasserini, and S. De, "VNF placement and resource allocation for the support of vertical services in 5G networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 433–446, Feb. 2019.

[27] M. Bouet and V. Conan, "Mobile edge computing resources optimization: A geo-clustering approach," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 2, pp. 787–796, Jun. 2018.

[28] V. Sciancalepore, F. Z. Yousaf, and X. Costa-Perez, "z-TORCH: An automated NFV orchestration and monitoring solution," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, pp. 1292–1306, Dec. 2018.

[29] Q. Zhang, F. Liu, and C. Zeng, "Online adaptive interference-aware VNF deployment and migration for 5G network slice," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 2115–2128, Oct. 2021.

[30] J. Martin-Pérez, F. Malandrino, C.-F. Chiasserini, and C. J. Bernardos, "OKpi: All-KPI network slicing through efficient resource allocation," in *Proc. INFOCOM IEEE Conf. Comput. Commun.*, 2020, pp. 804–813.

[31] A. Al-Jawad, I.-S. Comşa, P. Shah, O. Gemikonakli, and R. Trestian, "An innovative reinforcement learning-based framework for quality of service provisioning over multimedia-based SDN environments," *IEEE Trans. Broadcast.*, vol. 67, no. 4, pp. 851–867, Dec. 2021.

[32] N. Van Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Optimal and fast real-time resource slicing with deep dueling neural networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1455–1470, Jun. 2019.

[33] H. Esmat and B. Lorenzo, "Deep reinforcement learning based dynamic edge/fog network slicing," in *Proc. IEEE Global Commun. Conf. GLOBECOM*, 2020, pp. 1–6.

[34] V. C. Ferreira, H. Esmat, B. Lorenzo, S. Kundu, and F. M. G. Felipe, "Reinforcement learning based multi-attribute slice admission control for next-generation networks in a dynamic pricing environment," in *Proc. IEEE 95th Veh. Technol. Conf. (VTC)*, 2022, pp. 1–5.

[35] "Amazon EC2-secure and resizable compute capacity." Amazon Web services. Feb. 2023. Accessed: Feb. 2, 2023. [Online]. Available: https://aws.amazon.com/ec2/pricing

[36] G. Sun, G. O. Boateng, L. Luo, H. Chen, D. A. Mensah, and G. Liu, "Competitive pricing for resource trading in sliced mobile networks: A multi-agent reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 3830–3845, May 2024.

[37] A. A. Abdellatif, A. Mohamed, C. F. Chiasserini, A. Erbad, and M. Guizani, "Edge computing for energy-efficient smart health systems: Data and application-specific approaches," in *Energy Efficiency of Medical Devices and Healthcare Applications*. Amsterdam, The Netherlands: Elsevier, 2020, pp. 53–67.

[38] K. Kamran, E. Yeh, and Q. Ma, "Deco: Joint computation, caching and forwarding in data-centric computing networks," in *Proc. 20th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2019, pp. 111–120.

[39] T. Norp, "5G requirements and key performance indicators," *J. ICT Stand.*, vol. 6, nos. 1–2, pp. 15–30, 2018.

[40] M. S. Allahham, A. Mohamed, A. Erbad, and H. Hassanein, "On the modeling of reliability in extreme edge computing systems," in *Proc. 5th Int. Conf. Commun., Signal Process., Appl. (ICCSPA)*, 2022, pp. 1–6.

[41] M. R. Segal, *Machine Learning Benchmarks and Random Forest Regression*, eScholarship, Oakland, CA, USA, 2004.

[42] Y. Choi, "Tree-structured regression for a loglinear model with an extra-Poisson variation," Ph.D. dissertation, Dept. Math. Statist., State Univ. New York, Stony Brook, NY, USA, 2002.

[43] S. L. Ho and M. Xie, "The use of ARIMA models for reliability forecasting and analysis," *Comput. Ind. Eng.*, vol. 35, nos. 1–2, pp. 213–216, 1998.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[45] F. Haouari, E. Baccour, A. Erbad, A. Mohamed, and M. Guizani, "Transcoding resources forecasting and reservation for crowdsourced live streaming," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–7.

[46] R. Gerardo, P. Lele, K. Sundaram, and T. Ponsky, "Surgical telementoring: Feasibility, applicability, and how to," *J. Surg. Oncol.*, vol. 124, no. 2, pp. 241–245, 2021.

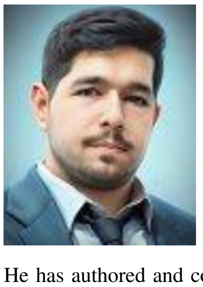
[47] K. Gai and M. Qiu, "Optimal resource allocation using reinforcement learning for IoT content-centric services," *Appl. Soft Comput.*, vol. 70, pp. 12–21, Sep. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494618302540

[48] A. A. Abdellatif, N. Mhaisen, Z. Chkirbene, A. Mohamed, A. Erbad, and M. Guizani, "Reinforcement learning for intelligent healthcare systems: A comprehensive survey," 2021, *arXiv:2108.04087.*

[49] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347.*

[50] 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. Serv. Manag.*, vol. 20, no. 2, pp. 1024–1039, May 2023.

[51] 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.



**AMR E. ABOELENEEN** (Member, IEEE) received the B.Sc. and M.S. degrees in computer science and engineering from Qatar University in 2018 and 2021, respectively. He is currently pursuing the Ph.D. degree with Hamad Bin Khalifa University. He brings over two years of experience as a Research Assistant with Qatar University. His passion lies in applying innovative artificial intelligence methods to enhance networks and the Internet of Things in health-related scenarios, alongside utilizing AI for health-related imaging. He has authored and coauthored a publication record of over nine research papers, He also has both won and participated in various local AI competitions. Additionally, he was the recipient of the Graduate Student Research Award from the Qatar National Research Fund.

**ALAA A. ABDELLATIF** (Member, IEEE) received the B.Sc. and M.Sc. degrees (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 in 2018. He is currently a Postdoctoral Researcher and a Part-Time Lecturer with Qatar University. He also worked as a Senior Research Assistant and a Research Assistant with Qatar University from 2013 to 2015, and with Cairo University from 2009 to 2012, respectively. He has authored or coauthored over 55 refereed journal, magazine, and conference papers in reputable international journals and conferences. He has served as a technical reviewer for many international journals and magazines. His research interests include edge computing, network security, blockchain, machine learning and resources optimization for next-generation wireless networks, smart-health, IoT applications, and vehicular networks. He was the recipient of the Graduate Student Research Award from Qatar National Research Fund, and the Best Paper Award from Wireless Telecommunications Symposium 2018 in USA, in addition to the Quality Award from the Politecnico di Torino in 2018.

**AIMAN M. ERBAD** (Senior Member, IEEE) received the B.Sc. degree in computer engineering from the University of Washington, Seattle, in 2004, the Master of Computer Science degree in embedded systems and robotics from the University of Essex, U.K., in 2005, and the Ph.D. degree in Computer Science from the University of British Columbia, Canada, in 2012. He is an Associate Professor and ICT Division Head with the College of Science and Engineering, Hamad Bin Khalifa University. He published more than 160 papers in reputable international conferences and journals. His research interests span cloud computing, quantum networks, edge intelligence, Internet of Things, and private and secure networks. He received the 2020 Best Research Paper Award from Computer Communications, the IWCMC 2019 Best Paper Award, and the IEEE CCWC 2017 Best Paper Award. He is the General Chair for ISNCC 2023. He also served as the Program Chair of IWCMC 2022 and IWCMC 2019, as the Publicity Chair of ACM MoVid 2015, as the Local Arrangement Chair of NOSSDAV 2011, and as a Technical Program Committee Member in various IEEE and ACM international conferences (GlobeCom, ICC, NOSSDAV, MMSys, ACMMM, IC2E, and ICNC). He is a Senior Member of ACM.

**AMR M. SALEM** (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 over 25 years of experience in wireless networking research and industrial systems development. He has authored or co-authored over 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 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.