SafeDRL: Dynamic Microservice Provisioning With Reliability and Latency Guarantees in Edge Environments

Yue Zeng, Zhihao Qu, *Member, IEEE*, Song Guo, *Fellow, IEEE*, Baoliu Ye, *Member, IEEE*, Jie Zhang, *Member, IEEE*, Jing Li, *Member, IEEE*, and Bin Tang *Member, IEEE*

***Abstract*—As a key technology of 5G, network function virtualization enables each monolithic service to be divided into microservices, facilitating their deployment and management in edge environments. One of the most critical issues in 5G is how to support dynamically arriving mission-critical services with low-latency and high-reliability requirements in distributed edge environments. However, most existing works focus on how to provide reliable services without considering latency, and their heuristics struggle to cope with high-dimensional constraints and complex environments with heterogeneous infrastructure and services. In this paper, we propose a SafeDRL algorithm to resource-efficiently support these dynamically arriving services while meeting their reliability and latency requirements. Specifically, we first formulate the problem as an integer nonlinear programming and prove its NP-hardness. To tackle this problem, our SafeDRL algorithm captures delayed rewards in dynamic environments by reinforcement learning, and corrects constraint violations with high-quality feasible solutions based on expert intervention, and prunes unnecessary backup instances for optimality. The algorithm is proved to have a bounded approximation ratio in general cases. Extensive trace-driven simulations show that, compared with the state-of-the-art solution, SafeDRL can save resource costs by up to 49.32% and improve the service acceptance ratio by up to 55% with acceptable execution time.**

***Index Terms*—5G, deep reinforcement learning, network function virtualization, edge computing, mission-critical services.**

Manuscript received 1 February 2023; revised 27 June 2023; accepted 16 October 2023. Date of publication 2 November 2023; date of current version 22 December 2023. This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant No. B210201053, in part by the National Natural Science Foundation of China (Grant Nos. 61832005, 62172204, and 62102131), in part by the Natural Science Foundation of Jiangsu Province, China (Grant Nos. BE2020001-3 and BK20210361), in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization, in part by the Key-Area Research and Development Program of Guangdong Province (No. 2021B0101400003), in part by the Hong Kong RGC Research Impact Fund (No. R5060-19, No. R5034-18), in part by the Areas of Excellence Scheme (AoE/E-601/22-R), in part by the General Research Fund (No. 152203/20E, 152244/21E, 152169/22E, 152228/23E), and in part by the Shenzhen Science and Technology Innovation Commission (JCYJ20200109142008673). Recommended for acceptance by E. Smirni. *(Corresponding authors: Zhihao Qu; Song Guo; Baoliu Ye.)*

Yue Zeng and Baoliu Ye are with the State Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China (e-mail: zengyue@smail.nju.edu.cn; yebl@nju.edu.cn).

Song Guo is with The Hong Kong University of Science and Technology, Hong Kong, China (e-mail: songguo@cse.ust.hk).

Jie Zhang and Jing Li are with The Hong Kong Polytechnic University, Hong Kong, China (e-mail: 18104473r@connect.polyu.hk; jing5.li@polyu.edu.hk).

Zhihao Qu and Bin Tang are with the Key Laboratory of Water Big Data Technology of Ministry of Water Resources, and with the College of Computer Science and Software Engineering, Hohai University, Nanjing 211100, China (e-mail: quzhihao@hhu.edu.cn; cstb@hhu.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TC.2023.3329194, provided by the authors.

Digital Object Identifier 10.1109/TC.2023.3329194

I. Introduction

As a key enabling technology for 5G, network function virtualization (NFV) virtualizes hardware-based service functions (SFs) into software, enabling SFs to be flexibly and elastically deployed to commodity servers [1], [2], [3], [4]. Driven by virtualization technology, the microservice architecture (MSA) separates each monolithic service into multiple independent microservices (called SFs next).[1] Benefiting from the loosely coupled architecture, services can achieve independent deployment, fast iteration, and flexible management.

Recently, many mission-critical applications have urged service providers to support low-latency and high-reliability services. According to 3GPP reports [5], AR/VR requires 0.9999 reliability and 10 ms latency, while medical monitoring demands 0.999999 reliability and 100 ms latency. However, the service deployed on 5G edge sites (ESs) may experience low reliability and high latency due to the following reasons. First, each service consists of multiple SFs; the entire service fails once an SF fails [6]. Second, the ES devices are low-end, poorly maintained, and run in poor operating environments [7]. Finally, the component SFs of each service need to be deployed to highly distributed ESs, which may experience high propagation latency.

Backup is an effective way to improve service reliability. Once an SF fails, the traffic is redirected to its backup. Then, the failure is masked, and the service is maintained. However, more backups will result in higher resource costs. Therefore, many existing works have studied how to back up and deploy services to meet their reliability requirements while minimizing resource costs [6], [8], [9]. In these studies, reliability is defined as the probability that a packet is successfully transmitted

[1]Microservice is a virtualized service function, which is also called VNF. In this paper, we refer to them as service functions.

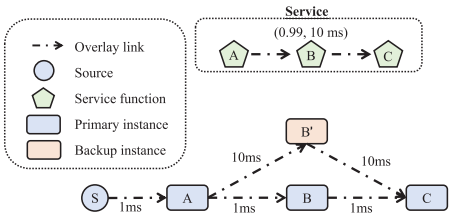


Fig. 1. An example to illustrate the service provisioning with both reliability and latency requirements.

from its source to its destination, where the service latency requirement is ignored. This is reasonable for traditional services with looser performance requirements, but not mission-critical services with strict latency requirements. Considering latency, reliability is defined as the probability that a packet is successfully transmitted from its source to its destination within a time period [10]. For example, as shown in Fig. 1, there is a service consisting of three SFs, which requires 0.99 reliability and 10 ms latency. After they are deployed, traffic is forwarded on path $S \rightarrow A \rightarrow B \rightarrow C$, where A, B, and C are selected as primary instances for forwarding traffic. To improve service reliability to meet its reliability needs, we provide SF B with a backup B'. In this way, once SF B fails, traffic is redirected to SF B' and traverses $S \rightarrow A \rightarrow B' \rightarrow C$ with a latency of 21 ms. This path is considered reliable when the latency is ignored but unreliable when the latency is considered. Therefore, ignoring latency in traditional work may lead to service performance violations, resulting in frequent service request rejections.

However, it is challenging to cost-effectively support dynamically arriving services to meet their reliability and latency requirements on limited and heterogeneous infrastructures. First, service requests arrive dynamically, and current decisions may affect future decisions due to limited resources, while future services are unknown. Second, high-dimensional resource constraints hinder finding feasible solutions in the solution space to satisfy the capacity, reliability, and latency constraints simultaneously. Third, heterogeneous service requests and limited and heterogeneous infrastructure resources make it difficult to extract empirical rules to guide optimal or near-optimal decisions. For example, each ES has limited resources and has different resource charges and hardware reliability. Each service may be composed of different SFs with different reliability and latency requirements, and each SF is heterogeneous in resource consumption and software reliability.

Deep reinforcement learning (DRL) is designed to capture delayed rewards in dynamic environments, while leveraging deep neural networks to extract hidden rules behind complex environments.[2] As a result, several studies have investigated how to leverage DRL for the resource-efficient deployment of dynamically arriving services [12], [13], [14], [15] and to ensure their reliability [9]. However, they fail to support mission-critical services with latency and reliability requirements, while random exploration in DRL may frequently violate constraints. As an important branch of reinforcement learning, safe reinforcement learning aims to learn policies to maximize long-term rewards while respecting constraints. Then, several works [16], [17], [18] have designed safety reinforcement learning algorithms based on human intervention to correct actions that violate constraints. However, frequent human intervention may involve high labor costs and suboptimal solutions.

In this paper, we design a SafeDRL algorithm to resource-efficiently support dynamically arriving services to meet their reliability and latency requirements in heterogeneous infrastructures. To model all the features, we formulate the problem as an integer nonlinear programming and prove its NP-hardness, revealing its challenges. To capture delayed rewards in dynamic environments, we exploit a typical DRL, called DDPG [19], which can effectively handle our problem with continuous state and large-scale discrete action and adapt it to our problem with varying state and behavior sizes in a redundant way. To handle the high-dimensional constraints, we design an expert intervention algorithm based on our insights to correct actions that violate the constraints. Moreover, for optimality, we design a pruning algorithm to prune unnecessary instances in the action that do not violate constraints, which can effectively avoid exploring those low-quality feasible solutions. Finally, we verify the optimality, superiority, and practicality of our algorithm via extensive trace-driven experimental results.

The main contributions of this paper are summarized as follows.

- To our knowledge, we are the first to study how to support dynamically arriving services with high reliability and low latency requirements. To model all features, we formulate this problem as integer nonlinear programming and prove its NP-hardness.
- We design a SafeDRL algorithm to make backup, deployment, and primary instance selection decisions to meet dynamic service requests while respecting constraints. The core idea is integrating expert interventions to correct the action that violates the constraints. Moreover, we propose a pruning algorithm that prunes unnecessary backup instances for optimality. The algorithm is proved to have a bounded approximation ratio in general cases.
- Extensive simulation results driven by Alibaba trace show that, compared with the state-of-the-art solution, SafeDRL can save resource costs by up to 49.32% and improve the service acceptance ratio by up to 55%.

The rest of the paper is organized as follows. Section II briefly introduces the related work. Section III formulates the problem we studied and proves its NP-hardness. SafeDRL is proposed and analyzed in Section IV and evaluated in Section V. Section VI concludes the paper.

II. Related Work

We summarize related work into three categories and discuss their drawbacks and differences compared to our work.

[2] **Delayed reward**. In dynamic environments, the current decision determines the immediate reward and the next state of the environment [11]. The **delayed reward** is a metric to measure the benefits of the next state, which indicates the impact of the current decision on the future, helping to achieve higher long-term returns.

A. Heuristics for Service Provision

There is a series of works [6], [8], [20], [21], [22], [23] that have investigated how to support service provisioning cost-effectively, and efficient heuristics have been devised based on expert insights. As pioneers, Cohen et al. [20] investigated how to deploy SFs for resource-efficient service provisioning. Considering the latency requirement, Jin et al. [21] further investigated how to deploy SFs to meet service latency requirements while minimizing resource cost. Moreover, considering reliability requirements, Martin et al. [22] studied how to deploy SFs resource-efficiently to meet service latency and reliability requirements. However, how to provide backup for SFs is beyond their concern, which fails to support services with high-reliability requirements. To facilitate these services, several works [6], [8], [23] have studied how to back up and deploy SFs to meet their service reliability requirements while minimizing resource costs. However, they fail to meet services with low latency needs. Moreover, the heuristic solutions they designed make one-shot decisions and fail to capture delayed rewards in dynamic environments, leading to suboptimal solutions.

B. DRL-Based Solution for Service Provision

DRL is designed to capture delayed rewards in dynamic environments, and it can utilize neural networks to extract hidden rules from complex environments. Then, several studies [12], [13], [14], [15] have investigated how to leverage DRL for service provisioning. Considering service reliability, Jia et al. [9] further studied how to deploy and backup services resource-efficiently to meet their reliability requirements. However, service latency requirements are beyond their consideration. Moreover, the above DRL-based schemes only add a penalty item in the reward to punish the behavior violating the constraint, which can hardly prevent DRL's random exploration from violating the constraint, especially for our problem with high-dimensional constraints.

C. Safe Reinforcement Learning

Safe reinforcement learning, an important branch of reinforcement learning aiming at maximizing expected returns while respecting security constraints, is critical for real-world applications. Many research works [16], [17], [18], [24], [25] have designed various security reinforcement learning algorithms for different application scenarios. As pioneers, Altman et al. [24] designed safe reinforcement learning based on linear programming, which is applicable to tabular settings where state-action pairs are enumerable. However, this fails to deal with large-scale or even continuous actions and states. Moreover, Lagrangian multiplier methods [25] are used to penalize actions that violate constraints, depending on the degree of constraint violation. This applies when appropriate constraint violations are allowed, but not for mission-critical applications with demanding performance requirements. Finally, several works [16], [17], [18] have designed human-intervention-based safe reinforcement learning algorithms that aim to use human knowledge-based interventions to correct actions that violate constraints. However, frequent human intervention may involve high labor costs and suboptimal solutions.

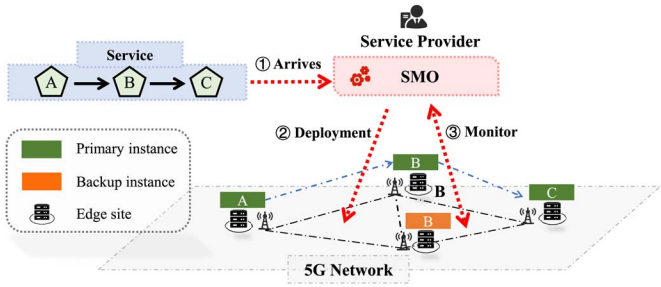


Fig. 2. An example to illustrate the NFV-enabled edge computing system.

The above works design rule-based or DRL-based solutions for highly reliable or low-latency service provisioning. However, none of them considered how to provide low-latency and high-reliability services in dynamic edge environments, where existing work frequently violates service requirements and suffers from resource inefficiency. Inspired by the safety reinforcement learning algorithm based on human intervention, we design an expert intervention-assisted DRL algorithm to capture delayed rewards while holding constraints.

III. System Model and Problem Formulation

This section outlines the system model, formulates the reliable service provision problem, and analyzes its complexity.

A. Primer

1) System Model: As shown in Fig. 2, our system consists of three components [26] called services, infrastructure, and service management and orchestrator (SMO), which can be modeled as

Service. Each service consists of multiple SFs, which can be implemented on commodity servers via virtual machines or containers. In the NFV-enabled edge computing system (NFV-ECS), service requests arrive one by one. Let i denote the i-th service request, and $\mathcal{I}$ denote the set of service requests. Each service i has reliability requirement $\mathbb{R}^i$ and latency requirement $\mathbb{L}^i$. Besides, each service consists of a set of SFs $\mathcal{M}^i$, where j-th SF is denoted as $m^{i,j}$. Each SF $\mathcal{M}^{i,j}$ has software reliability $r^{i,j}$, requires computing resources $c^{i,j}$, and forwards traffic of size $b^{i,j}$ to its next SF.

Infrastructure. In NFV-ECS, there is a set $\mathcal{E}$ of ESs. For each ES $e \in \mathcal{E}$, its computing resource capacity and hardware reliability are denoted by C_e and r_e, respectively. Let τ_e denote the cost per unit computing resource on ES e [27]. These ESs are connected by a set of overlay links $\mathcal{L}$, each link $(e, e') \in \mathcal{L}$ having bandwidth capacity $\mathcal{B}_{e,e'}$ and propagation latency $l_{e,e'}$. Let $\kappa_{e,e'}$ denote the cost per unit bandwidth resource on link (e, e') [28]. Besides, the latency from the source of service i to each ES e is denoted as l_e^i.

SMO. Service management and orchestrator is owned by the service provider and is responsible for managing and monitoring the virtualized physical resources on the underlying infrastructure. When a service request arrives, the SMO needs

TABLE I
SUMMARY OF NOTATIONS

Symbols	Descriptions
$\mathcal{E}, \mathcal{I}, \mathcal{L}$	Set of ESs, services, overlay links
$\mathcal{M}^i$	Set of SFs in service $i \in \mathcal{I}$
$\mathbb{R}^i$	Reliability required by service $i \in \mathcal{I}$
$\mathbb{L}^i$	Latency requirements by service $i \in \mathcal{I}$
$\Upsilon^{i,j}$	Set of ESs with instances deployed for SF $m^{i,j}$
$\mathcal{P}^i$	Set of routing paths for service i
$\overline{\mathcal{P}}^i$	Set of paths in $\mathcal{P}^i$ that violate the latency requirement
C_e	Computing capacity of ES $e \in \mathcal{E}$
$\mathcal{B}_{e,e'}$	Bandwidth capacity of link $(e, e') \in \mathcal{L}$
$p^{i,k}$	k-th routing path in $\mathcal{P}^i$
ρ^i	Primary routing path for service i
$\tilde{p}^{i,k,j}$	j-th ES on path $p^{i,k}$
$\tilde{\rho}^{i,j}$	j-th ES on path ρ^i
τ_e	Cost of unit computing resources on ES $e \in \mathcal{E}$
$\kappa_{e,e'}$	Cost of unit bandwidth resources on link $(e, e') \in \mathcal{L}$
r_e	Hardware reliability of ES $e \in \mathcal{E}$
$m^{i,j}$	j-th SF in service $i \in \mathcal{I}$
$r^{i,j}$	Software reliability of SF $m^{i,j}$
$c^{i,j}$	Computing resources required by SF $m^{i,j}$
$b^{i,j}$	Bandwidth resource required by the traffic from SF $m^{i,j}$
$l_{e,e'}$	Propagation latency on link $(e, e') \in \mathcal{L}$
l_e^i	Propagation latency from the source of service i to ES $e \in \mathcal{E}$
Decisions	**Descriptions**
$x_e^{i,j}$	Binary variable indicating whether to deploy the instance of SF $m^{i,j}$ to ES e
$y_e^{i,j}$	Binary variable indicating whether to set the instance deployed on ES e for SF $m^{i,j}$ as the primary instance
$z_{e,e'}^{i,j}$	Binary variable indicating whether the outgoing traffic from SF $m^{i,j}$ traverses link (e, e')

to implement the service into the underlying infrastructure and monitor it continuously. The SMO is implemented by fault-tolerant software-defined network (SDN) controllers [29] and is considered reliable.

The above system architecture is designed based on the NFV architecture specified by ETSI [26], and it can be well integrated into the 5G system (5GS) to serve mission-critical applications and network services. In 5GS [30], [31], SMO is owned by network service producers, and accepts requests from network function service consumers, and implements network services into the underlying infrastructure in the form of microservices according to their latency and reliability requirements.

2) Highly-Reliable and Low-Latency Service Provisioning: In our dynamic NFV-ECS, the service requests arrive one by one and need to be deployed to the underlying infrastructure immediately. To meet the service reliability, we may need to provide backups for its component SFs. Besides, we also need to select the primary instances to forward the traffic of the service. Therefore, we need to make the following three decisions: i) how many instances are needed for each SF, with one as the primary instance and the other as the backup, ii) which ESs to deploy these SF instances to, iii) which instance is selected as the primary instance to forward traffic. For ease of understanding, we give an example as shown in Fig. 2. There is a service consisting of three SFs A, B, and C. We deploy instances of these three SFs to the underlying infrastructure, where the primary instances are selected to forward traffic. Moreover, we also deploy a backup instance for SF B to improve its reliability. Once the primary instance of SF B fails, its traffic will be redirected to its backup instance to mask the failure and maintain service.

B. Problem Analysis

Our research investigates how to support dynamically arriving services under limited resource capacity to meet their reliability and latency requirements while minimizing resource cost. Next, we analyze and formalize the cost model, along with service completeness, primary instance, capacity, latency, and reliability constraints.

1) Cost Model: Implementing SFs in infrastructure will introduce financial costs, including computational costs and bandwidth costs. Specifically, deploying primary and backup instances for the component SFs of service i incurs computational resource costs, while forwarding its traffic between primary SF instances incurs bandwidth resource costs.[3] Hence, the resource cost of service i can be formalized as

$$\mathcal{C}^i = \sum_{j \in \mathcal{M}^i} \sum_{e \in \mathcal{E}} \tau_e c^{i,j} x_e^{i,j}, + \sum_{j \in \mathcal{M}^i} \sum_{e,e' \in \mathcal{L}} b^{i,j} \kappa_{e,e'} z_{e,e'}^{i,j}, \forall i \in \mathcal{I}, \tag{1}$$

where $x_e^{i,j}$ is a binary variable indicating whether to deploy the instance of SF $m^{i,j}$ to ES e. And, $z_{e,e'}^{i,j}$ is a binary variable indicating whether the traffic of service i traverses link (e, e'). It should be noted that, similar to [8], the off-site backup (rather than on-site backup) strategy is adopted in our research. This backup strategy provides backup instances on geographically separated ESs for the primary VNF instance, which can provide higher reliability and is suitable for applications with high-reliability requirements.

2) Service Completeness Constraints: For each service, at least one instance of its component SF needs to be deployed to ensure its normal operation. This can be formulated as

$$\sum_{e \in \mathcal{E}} x_e^{i,j} \geq 1, \forall i \in \mathcal{I}, j \in \mathcal{M}^i. \tag{2}$$

3) Primary Instance Constraints: Each SF in each service needs to deploy a primary instance for forwarding its traffic (Eq. (3)), and other instances serve as backup instances. The instance on ES e can be served as the primary instance of SF $m^{i,j}$ only if the instance of the SF is deployed on the ES (Eq. (4)). The traffic from SF $m^{i,j}$ traverses link (e, e') only if its primary instance is located at ES e and the primary instance of its successor SF $m^{i,j+1}$ is located at ES e' (Eq. (5)). The above conditions can be formalized as

$$\sum_{e \in \mathcal{E}} y_e^{i,j} = 1, \forall i \in \mathcal{I}, j \in \mathcal{M}^i, \tag{3}$$

[3]It should be noted that the bandwidth on non-primary routing paths is allocated only when failures occur; otherwise it will result in overwhelming bandwidth resource cost for exponential routing paths. Although these paths may not be allocated enough bandwidth due to insufficient link bandwidth, this is negligible, since the network is highly redundant and designed for traffic peaks, its average link utilization is only 30-40% [32]. Besides, these costs are negligible due to infrequent failures. For example, a reliability of 0.99 means that its downtime only accounts for 1% of the total running time.

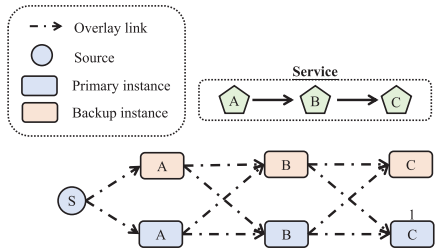


Fig. 3. An example to illustrate the potential paths of the routing traffic of a service.

$$x_e^{i,j} \geq y_e^{i,j}, \forall i \in \mathcal{I}, j \in \mathcal{M}^i, e \in \mathcal{E}, \tag{4}$$

$$z_{e,e'}^{i,j} = y_e^{i,j} y_{e'}^{i,j+1}, \forall i \in \mathcal{I}, j \in \mathcal{M}^i, (e,e') \in \mathcal{L}, \tag{5}$$

where $y_e^{i,j}$ is a binary variable indicating whether to set the instance deployed on ES e for SF $m^{i,j}$ as the primary instance. Obviously, once $y_e^{i,j}$ is determined, $z_{e,e'}^{i,j}$ is determined.

4) Capacity Constraints: The computational load on each ES and the communication load on each link should not exceed their capacity, which can be expressed as

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{M}^i} c^{i,j} x_e^{i,j} \leq C_e, \quad \forall e \in \mathcal{E}. \tag{6}$$

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{M}^i} b^{i,j} z_{e,e'}^{i,j} \leq \mathcal{B}_{e,e'}, \quad \forall (e,e') \in \mathcal{L}. \tag{7}$$

5) Latency Constraints: End-to-end latency (E2E) is defined as the time required for a packet to be transmitted from its source to its destination [33]. We give an example to illustrate the latency constraint, as follows. As shown in Fig. 3, there is a client at s that requests a service, which consists of three SFs A, B, and C. Each of them is deployed with two instances, one as the primary instance and one as the backup. In this case, the client's traffic may traverse any path consisting of these instances when a failure occurs or does not occur. To satisfy the latency constraint, we need to prune the paths that violate the latency requirement. Therefore, based on the above analysis, we model the latency constraint as follows.

Each SF may be deployed with multiple instances, one as the primary instance and the others as backups, as shown in Fig. 3. Then, the set of ESs with instances deployed for SF $m^{i,j}$ can be denoted as

$$\Upsilon^{i,j} = \{e \in \mathcal{E} | x_e^{i,j} = 1\}, \forall i \in \mathcal{I}, j \in \mathcal{M}^i. \tag{8}$$

For each service i, its traffic can be routed along any path consisting of its deployed SF instances. These routing paths can be denoted as

$$\mathcal{P}^i = \{(s_{i,1}, ..., s_{i,j}, ..., s_{i,|\mathcal{M}^i|}), \forall s_{i,j} \in \Upsilon^{i,j}\}, \forall i \in \mathcal{I}, \tag{9}$$

where $s_{i,j}$ denotes any ES in $\Upsilon^{i,j}$.

E2E latency includes transmission, queuing, processing, and propagation latency. First, the queuing latency should be negligible, which can be guaranteed by allocating adequate resources, which is critical for mission-critical applications with demanding delay requirements.[4] Besides, the transmission latency is also negligible because the packets are particularly small in size. Moreover, the processing latency is also negligible. This is because an SF processes a packet in 0.01-0.1 ms [34], whereas typical services require latency of 5-100 ms [5]. Finally, propagation latency is non-negligible because ESs are highly distributed. Propagation latency includes all link latency on the path and the latency from the source to the first ES on the path. Then, the E2E latency on any routing path $p^{i,k} \in \mathcal{P}^i$ can be expressed as

$$L^{i,k} = \sum_{(e,e') \in p^{i,k}} l_{e,e'} + l_{\gamma_i, \delta_{i,k}}, \forall p^{i,k} \in \mathcal{P}^i. \tag{10}$$

where γ_i denotes the source of service i and $\delta_{i,k}$ denotes the first ES on path $p^{i,k}$.

Then, the set of routing paths that violate the latency requirement can be expressed as

$$\overline{\mathcal{P}}^i = \{p^{i,k}, \forall p^{i,k} \in \mathcal{P}^i, L^{i,k} > \mathbb{L}^i\}. \tag{11}$$

6) Reliability: Reliability is defined as the probability that a data packet is successfully transferred within a time period [10]. For example, VR requires 99.99% reliability and 10 ms E2E latency. This means that its packets need to be successfully transmitted within 10 ms with no less than 99.99% probability. Obviously, latency and reliability are coupled. Besides, reliability refers to a probability term that is distributed from 0 to 1, i.e., $\mathbb{R}^i \in [0,1]$.

Next, we first model the service reliability without considering the latency, and then further model the service reliability with considering the latency to eliminate the reliability brought by the path that violates the constraint.

Reliability without latency. For each service $i \in \mathcal{I}$, once a component SF $m^{i,j}$ fails, the entire service fails [6]. Besides, similar to [6], [8], the virtual link connecting SFs is considered reliable.[5] Then, the service reliability can be defined as

$$R^i = \prod_{j \in \mathcal{M}^i} R^{i,j}, \forall i \in \mathcal{I}, \tag{12}$$

where R^i denotes service reliability regardless of latency, and $R^{i,j}$ denotes the reliability of SF $m^{i,j}$.

For each SF, it fails only when its instance fails on all ESs [37]. Then, the reliability of SF $m^{i,j}$ can be characterized as

$$R^{i,j} = 1 - \prod_{e \in \mathcal{E}} (1 - R_e^{i,j}), \forall i \in \mathcal{I}, j \in \mathcal{M}^i, \tag{13}$$

where $R_e^{i,j}$ indicates the reliability of SF $m^{i,j}$ on ES e.

[4]It is important to note that providing physical resources in terms of application demand peaks ensures no queuing latency, which is critical for mission-critical applications such as remote surgery and autonomous driving. Queuing latency with drastic changes in load is unacceptable for these mission-critical applications.

[5]The reliable virtual links can be achieved by SDN-based multi-path routing mechanisms [35]. Although there are also backup mechanisms that can restore failed SFs, such as 1:1 backup [36], it provides a static backup for each container-based SF. However, it is cost-inefficient. In this work, we explore a cost-effective service provisioning mechanism.

An SF fails on an ES only when the SF instance fails, or the ES fails [37]. Therefore, the reliability of SF $m^{i,j}$ on ES e can be denoted as

$$R_e^{i,j} = r^{i,j} r_e x_e^{i,j}, \forall i \in \mathcal{I}, j \in \mathcal{M}^i, e \in \mathcal{E}, \tag{14}$$

Reliability with latency. After considering the latency, the service reliability may be degraded due to latency constraint violation. For example, as shown in Fig. 1, a service's traffic can traverse $A \rightarrow B' \rightarrow C$, which is considered reliable when ignoring latency requirements. However, once traffic traverses this path, its latency requirement is violated. This is considered unreliable when considering latency requirements. Therefore, we need to prune the reliability enhanced by those paths that violate the latency constraint.

The service reliability is enhanced by path $p^{i,k}$ when the service's primary routing path $\tilde{p}^i$ has SF failures, and all other backup instances fail, and path $p^{i,k}$ is reliable. Besides, path $p^{i,k}$ is reliable only when all ES on the path and all components SFs in the service are reliable. Then, the reliability of path $p^{i,k}$ can be expressed as

$$\hat{R}^{i,k} = \prod_{j \in \mathcal{M}^i} r^{i,j} \prod_{e \in p^{i,k}} r_e, \forall p^{i,k} \in \overline{\mathcal{P}}^i, i \in \mathcal{I}. \tag{15}$$

For path $p^{i,k}$, there may be one or several backup instances instead of the primary instance, and they work and enhance service reliability when their primary instances on path $\tilde{p}^i$ fail and the other backup instances fail. Then, the reliability enhanced by path $p^{i,k}$ can be calculated as

$$\tilde{R}^{i,k} = \hat{R}^{i,k} \cdot \prod_{j \in \mathcal{M}^i, \rho^{i,k,j} \neq \tilde{\rho}^{i,j}} \Big(\prod_{e \in \Upsilon^{i,j}, e \notin p^{i,k}} (1 - R_e^{i,j}) \Big), \quad \forall p^{i,k} \in \overline{\mathcal{P}}^i, i \in \mathcal{I}, \tag{16}$$

where $\rho^{i,k,j}$ and $\tilde{\rho}^{i,j}$ denote j-th ES in paths $p^{i,k}$ and $\tilde{p}^i$, respectively.

Based on the above analysis, we need to prune the reliability enhanced by paths that violate constraints after considering service latency requirements. The reliability of each service i should be no less than its reliability requirements. This can be formulated as

$$\Re^i = R^i - \sum_{p^{i,k} \in \overline{\mathcal{P}}^i} \tilde{R}^{i,k} \geq \mathbb{R}^i, \forall i \in \mathcal{I}, \tag{17}$$

It should be noted that once all paths violate the latency constraint, that is, $|\overline{\mathcal{P}}^i| = |\mathcal{P}^i|$, then $\Re^i = 0$.

C. Problem Formulation

In summary, the offline version of the reliable and low-latency service provisioning (RLSP) problem in dynamic environments can be formulated as

$$\min_{x_e^{i,j}} \sum_{i \in \mathcal{I}} \mathcal{C}^i$$
$$s.t.\ (1) - (17),$$
$$x_e^{i,j} \in \{0,1\}, y_e^{i,j} \in \{0,1\}, z_{e,e'}^{i,j} \in \{0,1\}.$$

D. Problem Complexity

1) Hardness Result: We first analyze the hardness of finding a feasible solution for RLSP problem. Then, we analyze the hardness of solving RLSP problem in general and realistic situations in which all service requests are accepted. It should be noted that in edge computing environments, although each ES has limited resources, massive distributed ESs provide powerful computing capabilities. Therefore, it is generally realistic to have sufficient infrastructure resources to accept all service requests.

Theorem 1: It is strongly NP-hard to determine whether there is a feasible solution for RLSP problem.

Proof: See Appendix A (see the supplementary material). □

Theorem 2: Assuming all service requests are accepted, RLSP problem is NP-hard.

Proof: See Appendix B (available online). □

Since it is NP-hard to determine whether a feasible solution exists for RLSP problem, we only need to handle RLSP problem in a best-effort manner. Moreover, the above theorems rule out polynomial time algorithms for this problem, even when all service requests are accepted, unless $P = NP$.

2) Challenges: Solving RLSP problem while satisfying its constraints is non-trivial for the following reasons. First, service requests arrive one by one, and current decisions may have an impact on future decisions, resulting in delayed rewards, while future service requests are unknown. Second, there are many inherent features in RLSP problem, such that making deployment, backup and primary selection decisions with optimal or near-optimal resource costs is complicated. These features include heterogeneous resource capacity, resource cost and hardware reliability of the underlying infrastructure. Each service consists of different SFs, each of which has different software reliability and resource requirements. Third, RLSP problem is accompanied by high-dimensional constraints, including latency, reliability, and capacity constraints, that hinder the search for feasible solutions. Fourth, as shown in Eq. (9), providing any number of instances per SF may cause an exponential increase in the number of routing paths. As a result, reliability cannot be verified in polynomial time. For example, in NFV-ECS, there is a service with 7 SFs and 20 ESs. If each SF is deployed with one instance at each ES, there exist $20^7 = 1.28 * 10^9$ routing paths, which is unacceptable.

IV. SafeDRL Algorithm

In this section, we design a safe DRL (SafeDRL) algorithm to address the challenges mentioned above. Specifically, we first clarify the algorithm framework and the motivation behind it, then elaborate on the algorithm details, and finally analyze its theoretical properties.

A. Algorithm Framework

1) Motivation for Algorithm Framework: As discussed in the previous section, there are three challenges to solving RLSP problem. The first challenge is that in dynamic environments,

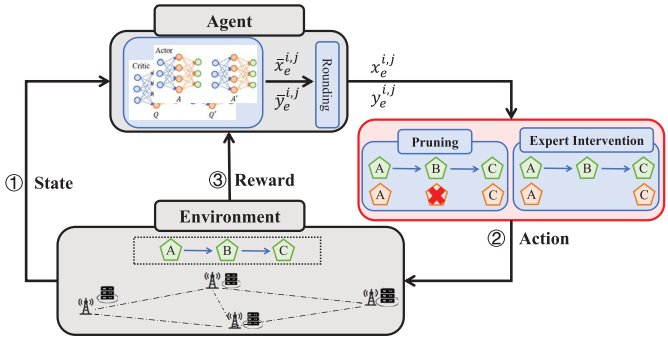


Fig. 4. Algorithm framework.

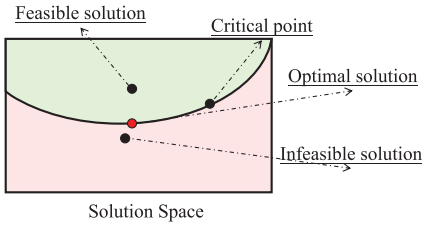


Fig. 5. Solution space, where the red area denotes the feasible solution space, and the green area denotes the infeasible solution space.

service requests arrive dynamically, which brings delayed rewards. To address this challenge, DRL is adopted to solve multi-stage decision problems in dynamic environments, capturing delayed rewards. As shown in Fig. 4, DRL contains two entities, the environment and the agent. First, the agent discovers the state from the environment, takes action based on the state, and acts on the environment. Then, the environment determines a reward based on the quality of the action. Finally, the agent updates the policy based on the feedback reward, aiming to maximize the long-term reward.

The second challenge is how to deal with optimality. In our RLSP problem, the solution space is huge and grows exponentially. For example, when $|\mathcal{E}| = 30$, $\mathcal{M}^i = 5$, the size of the solution space for a service is $2^{150} \approx 1.4 * 10^{45}$. This means that DRL needs to randomly explore such a huge solution space to search for the optimal solution, which may lead to suboptimal solutions and long training times. This is because the solution space is filled with decisions with many redundant instances, which are low-quality decisions. For example, in the worst case, DRL might provide 30 instances for an SF when there are 30 ESs which is unnecessary for reliability. Moreover, as shown in Fig. 5, the optimal solution is a critical point. The critical point refers to a decision in the solution space that satisfies all constraints, where adding an instance to it will lead to unnecessary resource cost, and reducing an instance to it will lead to reliability violation. Therefore, for the sake of efficiency, we design a pruning algorithm, which prunes unnecessary instances in the feasible solution output by DRL to save resource costs without violating any constraints.

The third challenge is how to make decisions to satisfy the high-dimensional constraints in RLSP problem. In DRL, the agent conducts trial-and-error exploration in the solution space to learn a policy that maximizes long-term reward. However, in RLSP problem, high-dimensional constraints lead to massive infeasible solutions, which makes random exploration in DRL frequently violate constraints, which is unacceptable for mission-critical applications. To address this challenge, we employ safe reinforcement learning, which aims to learn policies to maximize long-term rewards while respecting constraints. Inspired by human intervention-based safe reinforcement learning algorithms, we design an expert intervention algorithm to prevent constraint violations. The algorithm is triggered when the action output by the DRL violates the constraints, and it tries to replace the action that violates the constraints with a high-quality feasible solution.

The final challenge is that when providing an arbitrary number of instances for each SF, the routing paths grow exponentially, which is unacceptable. As mentioned in the previous section, when $|\mathcal{M}^i| = 7$ and $|\mathcal{E}| = 20$, in the worst case, there are $20^7 = 1.28 * 10^9$ routing paths. When determining whether the decision is feasible, we need to check all these paths to determine its reliability. Therefore, based on our insight, we limit the number of instances of each SF to no more than a constant ϵ, which is achieved by using top-ϵ rounding for the solution output by the neural network.

2) Workflow in Algorithm Framework: As shown in Fig. 4, our algorithm consists of three components, including DRL in the agent, pruning algorithm and expert intervention algorithm. First, the agent discovers the state from the environment and takes action based on that state. If the action output by the agent is a feasible solution, the pruning algorithm is called to prune unnecessary instances in the action, so that the modified action becomes a critical point in the solution space, which satisfies the constraints and has no unnecessary SF instances. On the contrary, if the output action is an infeasible solution, the expert intervention algorithm is used to replace the infeasible solution with a high-quality feasible solution. Then, the action is conducted in the environment and generates corresponding rewards. Finally, the agent updates the neural network based on the feedback reward to maximize the long-term reward.

B. Algorithm Design

1) RL Model: RL can be modeled as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$, which indicate the state, action, and reward, respectively. These three elements can be modeled as follows.

State. As shown in Fig. 4, the state in the environment includes information about that newly arrived service request and ESs. That is, $\mathcal{S} = \langle \mathbb{R}^i, \mathbb{L}^i, r^{i,j}, c^{i,j}, b^{i,j}, \tau_e, \kappa_{e,e'}, r_e, C_e, \mathcal{B}_{e,e'}, l_{e,e'}, l_e^i, \forall j \in \mathcal{M}^i, e \in \mathcal{E}, (e, e') \in \mathcal{L} \rangle$. In this case, the size of the state is $3|\mathcal{M}^i| + 4|\mathcal{E}| + 3|\mathcal{L}| + 2$. Obviously, the state size will vary with the number of SFs in the service, which does not match neural networks with fixed input layers. To overcome this issue, we express information about the service in a redundant manner. This is reasonable because the number of SFs in typical services does not exceed 7 [38]. Assume that the maximum service length is $|\mathcal{M}|$. Then, we model the service information with length $|\mathcal{M}|$. If the service length is less than $|\mathcal{M}|$, we fill it with zero. Then,

Algorithm 1 DRL Algorithm

Input: Parameters in RLSP.
Output: $x_e^{i,j}$, $y_e^{i,j}$, $z_{e,e'}^{i,j}$: decision variables.
$\overline{x}_e^{i,j}, \overline{y}_e^{i,j} \leftarrow$ call DDPG;
/*Top-ϵ Rounding*/
$x_e^{i,j} \leftarrow 0, \forall j \in \mathcal{M}^i, e \in \mathcal{E}$;
for $j \in \mathcal{M}^i$ **do**
  for $iteration \in \{1, 2, ..., \epsilon\}$ **do**
    $e \leftarrow \arg\max_e \{\overline{x}_e^{i,j} | x_e^{i,j} = 0, \forall e \in \mathcal{E}\}$;
    $x_e^{i,j} \leftarrow$ perform rounding on $\overline{x}_e^{i,j}$;
  end for
  $e \leftarrow \arg\max_e \{\overline{y}_e^{i,j} | x_e^{i,j} = 1, \forall e \in \mathcal{E}\}$;
  $y_e^{i,j} \leftarrow 1$;
  Caculate $z_{e,e'}^{i,j}$ based on Eq. (5);
end for

Algorithm 2 Pruning Algorithm

Input: $x_e^{i,j}$: action output by DRL agent.
Output: $\mathcal{A}$: binary variable indicates whether to accept the request; $x_e^{i,j}$, $y_e^{i,j}$, $z_{e,e'}^{i,j}$: decision variables.
$\mathcal{A} \leftarrow 1$;
while $True$ **do**
  Calculate $\Psi_e^{i,j}, \forall j \in \mathcal{M}^i, e \in \mathcal{E}$ by Eq. (18);
  $\mathcal{H} = \{(j,e), \forall j \in \mathcal{M}^i, e \in \mathcal{E} | x_e^{i,j} = 1, \Delta_e^{i,j} = 0\}$;
  if $\mathcal{H}$ is empty **then**
    $\mathcal{A} \leftarrow 0$, Break;
  end if
  $j, e = \arg\max_{j,e} \{\Psi_e^{i,j}, \forall (j,e) \in \mathcal{H}\}$;
  $x_e^{i,j} \leftarrow 0$;
  Calculate $\Re^i$ by Eq. (17);
end while

$\mathcal{S} = \langle \mathbb{R}^i, \mathbb{L}^i, r^{i,j}, c^{i,j}, b^{i,j}, \tau_e, \kappa_{e,e'}, r_e, C_e, \mathcal{B}_{e,e'}, l_{e,e'}, l_e^i, \forall j \in \{1, ..., |\mathcal{M}|\}, e \in \mathcal{E}, \ (e, e') \in \mathcal{L}\rangle$. Then, the state size is $3|\mathcal{M}| + 4|\mathcal{E}| + 3|\mathcal{L}| + 2$.

Action. The action is needed to make backup, deployment, and primary instance selection decisions (e.g., $x_e^{i,j}$, $y_e^{i,j}$, $z_{e,e'}^{i,j}$). Since $z_{e,e'}^{i,j}$ can be obtained by Eq. (5), after $y_e^{i,j}$ is determined. Then, the action can be denoted as $\mathcal{A} = \langle x_e^{i,j}, y_e^{i,j}, \forall j \in \mathcal{M}^i, e \in \mathcal{E}, e' \in \mathcal{E}\rangle$. The action size is $2|\mathcal{M}|^i|\mathcal{E}|$. The action size varies with the number of SFs in the service. Similar to the state model, we also model the agent's action in a redundant manner. Then, $\mathcal{A} = \langle x_e^{i,j}, y_e^{i,j}, \forall j \in \{1, ..., |\mathcal{M}|\}, e \in \mathcal{E}\rangle$, and the action size is $2|\mathcal{M}||\mathcal{E}|$.

Reward. The reward is the feedback from the environment that indicates the quality of the action. In RLSP problem, we aim to minimize the resource cost. The higher the resource cost, the smaller the reward. Besides, once the action output by the agent violates the constraint, it may lead to service request rejection and revenue loss, which should be punished. Therefore, we can define the reward as $\mathcal{R} = -\mathcal{C}^i - \rho o^i$, where o^i indicates whether the output action violates the constraint, o^i is 0 if it is satisfied, and 1 otherwise. ρ is a constant indicating the penalty factor.

2) DRL Algorithm: DRL is designed for the agent. In the RL model above, the state is continuous, and the action is large-scale and discrete. Therefore, we adopt a typical DRL algorithm, called DDPG [19], which can effectively deal with continuous state and large-scale discrete action. After defining the action, state, and reward above, the DDPG is specified. However, as mentioned above, for the fourth challenge, in the worst case, the neural network may deploy instances for each SF on all ESs, resulting in the exponential time required to verify the feasibility of the decision. We address this challenge based on the following insights.

Lemma 1: When the latency requirement is negligible, providing $\epsilon = 4$ instances for each SF satisfies almost all service reliability requirements in the general case, where $r^{i,j} \geq 0.995$, $r_e \geq 0.99$, $|\mathcal{M}| \leq 7$.

Proof: See Appendix C (available online). □

The above insight inspired us to limit the number of instances provided for each SF, which does not sacrifice feasible solutions while avoiding the exponential time to verify feasible solutions.[6] Thus, we redefine the output of DDPG as $\overline{\mathcal{A}} = \langle \overline{x}_e^{i,j}, \overline{y}_e^{i,j}, \forall j \in \{1, ..., |\mathcal{M}|\}, e \in \mathcal{E}\rangle$, where $\overline{x}_e^{i,j}, \overline{y}_e^{i,j} \in [0, 1]$. Then, we perform top-$\epsilon$ rounding for $\overline{x}_e^{i,j}$ to obtain $x_e^{i,j}$, as shown in Algorithm 1. This is used to ensure that the number of instances of each SF in the action output by the DRL does not exceed ϵ, further ensuring that each action can be verified as a feasible solution in polynomial time. Then, we select the instance on the ES with the maximum value ($\overline{y}_e^{i,j}$) as the primary instance of SF $m^{i,j}$. After that, we can determine which links the service's traffic traverses via Eq. (5). The details of the DDPG algorithm are omitted for brevity, and please refer to [19] for details.

3) Pruning Algorithm: As mentioned above, when a DRL agent outputs an action that does not violate constraints, it may contain unnecessary instances. For optimality, we need to design a pruning algorithm that removes unnecessary instances of the action without sacrificing constraints. Naturally, we need to decide which instances to prune preferentially. For this purpose, we define a priority

$$\Psi_e^{i,j} = \tau_e c^{i,j} \tag{18}$$

The motivation behind the definition is that we prefer to prune instances with high resource cost, which leads to more resource cost reduction.[7]

Then, the pruning algorithm is designed as shown in Algorithm 2. In the algorithm, we iteratively prune unnecessary instances for the action. Specifically, we first calculate the priority for each instance. Then, we further record those instances that can be pruned without violating constraints ($\Delta_e^{i,j}$ is a binary variable indicating whether the constraint will be violated after the instance of SF $m^{i,j}$ on ES e is pruned.). If no instances can be pruned (otherwise, the constraint is violated), the prune operation is terminated. Otherwise, we prune the instance with the highest priority.

[6]For example, when $|\mathcal{E}| = 20, |\mathcal{M}^i| = 7$, $\epsilon = 4$. In the worst case, there are $4^7 = 16384$ possible routing paths. This is acceptable.

[7]Reliability is ignored in the priority definition, due to the slight loss of reliability caused by pruning an SF instance. Moreover, once pruning an SF results in a reliability violation, it is excluded.

4) Expert Intervention Algorithm: The expert intervention algorithm is designed to output a high-quality solution to replace the constraint violation action output by DRL. Therefore, in this algorithm, we need to make resource-efficient backup, deployment, and primary instance selection decisions to meet the reliability and latency requirements of the service. Next, we give the following three insights to guide the above three decisions.

Backup. Although more backups may bring higher reliability, it also incurs higher resource costs. Therefore, we need to determine how to provide backups cost-effectively to meet the service reliability requirements.

Lemma 2: Service reliability is less than or equal to the reliability of the SF with the lowest reliability, that is, $\Re^i \leq \min\{R^{i,j}, \forall j \in \mathcal{M}^i\}$.

Proof: Based on Eq. (17), we have

$$\Re^i = R^i - \sum_{p^{i,k} \in \overline{\mathcal{P}}^i} \tilde{R}^{i,k} \leq R^i = \prod_{j \in \mathcal{M}^i} R^{i,j}, \tag{19}$$

where the inequality holds because the path reliability is not less than 0, that is, $\tilde{R}^{i,k} \geq 0$. The second equality is based on Eq. (12).

Since $1 \geq R^{i,j} \geq 0$, based on Eq. (19), we have

$$\Re^i \leq R^{i,j}, \forall j \in \mathcal{M}^i. \tag{20}$$

This implies,

$$\Re^i \leq \min\{R^{i,j}, \forall j \in \mathcal{M}^i\}. \tag{21}$$

□

The above lemma inspires us to provide backup for the SF with the lowest reliability to efficiently improve the service reliability to meet its reliability requirements.

Deployment. Deploying SF instances to reliable ESs may bring higher resource costs, but may face higher charges. Therefore, we define a priority metric to measure the reliability improvement per unit of resource cost as

$$\phi_e^{i,j} = \frac{\Theta_{j,e} - \Re^i}{\tau_e c^{i,j}}, \tag{22}$$

where $\Re^i$ is the reliability obtained by the current decision, and $\Theta_{j,e}$ denotes the reliability obtained after further deploying an instance on the ES e for SF $m^{i,j}$. The motivation behind this priority definition is that we prefer to deploy SF instances to the ES that brings higher reliability improvements per unit resource cost.

Primary instance selection. As mentioned earlier, each SF is deployed with no more than ϵ ($\epsilon = 4$) instances, and generally, each service consists of no more than 7 SFs ($|\mathcal{M}| \leq 7$) [39], resulting in a small solution space ($\leq 4^7 = 16384$) after the backup and deployment decision ($x_e^{i,j}$) is determined. Thus, we can explore all possible primary instance selection decisions to obtain the most cost-effective one, while traffic is forwarded on the primary instances without exceeding link capacity.

Based on the above insight, we designed the expert intervention algorithm as shown in Algorithm 3. In this algorithm, we first initialize the decisions and service reliability. We then iteratively add instances to the service to meet its reliability constraints. Specifically, we first calculate the reliability of each SF and obtain the one that has the lowest reliability and has no more than ϵ instances. Subsequently, we record those ESs with sufficient capacity to accommodate the selected SF. Then, we calculate the priority of each ES and extract the ES with the highest priority. Further, we add instances on the ES with the highest priority for the SF with the lowest reliability. The above operations are terminated until service reliability is satisfied, no ESs are available to accommodate SF instances, or all SFs have ϵ instances. If the reliability is satisfied, we calculate the set of paths ($\mathcal{P}^i - \overline{\mathcal{P}}^i$) that do not violate the latency constraint based on Eq. (9) and Eq. (11), and select the one with the minimum resource cost and meeting the capacity constraint from them. Based on this, we obtain the primary instance selection decision and update the remaining link capacity (the capacity of links and ESs remains unchanged when the service is rejected).

Algorithm 3 Expert Intervention Algorithm

Input: Parameters in RLSP.

Output: $\mathcal{A}$: binary variable indicates whether to accept the request; $x_e^{i,j}$, $y_e^{i,j}$, $z_{e,e'}^{i,j}$: decision variables.

$\mathcal{A} \leftarrow 1$;
$\Re^i \leftarrow 0, x_e^{i,j} \leftarrow 0, y_e^{i,j} \leftarrow 0, z_{e,e'}^{i,j} \leftarrow 0, \forall j \in \mathcal{M}^i, e \in \mathcal{E}$;
while $\Re^i < \mathbb{R}^i$ **do**
  Calculate $R^{i,j}, \forall j \in \mathcal{M}^i$ by Eq. (16);
  $j \leftarrow \arg\min_{j \in \mathcal{M}^i}\{R^{i,j} | \sum_{e \in \mathcal{E}} x_e^{i,j} \leq \epsilon\}$;
  $\mathcal{E}' \leftarrow \{e \in \mathcal{E} | x_e^{i,j} \leftarrow 0, C_e \geq c^{i,j}\}$;
  if $\mathcal{E}'$ is empty or $j = None$ **then**
    $\mathcal{A} \leftarrow 0$, Break;
  end if
  Calculate $\phi_e^{i,j}, \forall e \in \mathcal{E}'$ by Eq. (22);
  $e \leftarrow \arg\max_{e \in \mathcal{E}'} \phi_e^{i,j}$;
  $x_e^{i,j} \leftarrow 1$;
  $C_e \leftarrow C_e - c^{i,j}$;
  Update $\Re^i$ by Eq. (17);
end while
if $\mathcal{A} = 1$ **then**
  Calculate $\mathcal{P}^i, \overline{\mathcal{P}}^i$ based on Eq. (9) and Eq. (11);
  $\tilde{p}^i \leftarrow \arg\min_{p^{i,k} \in \mathcal{P}^i - \overline{\mathcal{P}}^i}\{\sum_{j \in \mathcal{M}^i} \sum_{e,e' \in \mathcal{L}} b^{i,j} \kappa_{e,e'} z_{e,e'}^{i,j} | \sum_{j \in \mathcal{M}^i} b^{i,j} z_{e,e'}^{i,j} \leq \mathcal{B}_{e,e'}, (e,e') \in p^{i,k}\}$;
  if $\tilde{p}^i = None$ **then**
    $\mathcal{A} \leftarrow 0$, Break;
  end if
  $z_{e,e'}^{i,j} \leftarrow 1, \forall (e,e') \in \tilde{p}^i$;
  Caculate $y_e^{i,j}$ based on Eq. (5);
  $\mathcal{B}_{e,e'} \leftarrow \mathcal{B}_{e,e'} - \sum_{j \in \mathcal{M}^i} b^{i,j} z_{e,e'}^{i,j}, \forall (e,e') \in \tilde{p}^i$;
end if

5) SafeDRL: The SafeDRL algorithm workflow is shown in Algorithm 4. The algorithm is triggered when a new service request arrives. The algorithm first calls DRL algorithm to make an action. If this action is a feasible solution, then the pruning algorithm is invoked to prune unnecessary instances for it to save resource costs without violating constraints. Conversely, if the decision is an infeasible solution, then an expert intervention

Algorithm 4 SafeDRL Algorithm

Input: Parameters in RLSP.

Output: $\mathcal{A}$: binary variable indicates whether to accept the request; $x_e^{i,j}$, $y_e^{i,j}$, $z_{e,e'}^{i,j}$: decision variables.

1: $x_e^{i,j}, y_e^{i,j}, z_{e,e'}^{i,j} \leftarrow$ **Algorithm 1**;
2: **if** $x_e^{i,j}, y_e^{i,j}, z_{e,e'}^{i,j}$ is feasible **then**
3: $x_e^{i,j}, y_e^{i,j}, z_{e,e'}^{i,j}, \mathcal{A} \leftarrow$ **Algorithm 2**;
4: **else**
5: $x_e^{i,j}, y_e^{i,j}, z_{e,e'}^{i,j}, \mathcal{A} \leftarrow$ **Algorithm 3**;
6: **end if**

algorithm is invoked to replace the decision with a high-quality solution.

C. Algorithm Analysis

Next, we analyze the theoretical properties of our algorithm, including the algorithm complexity and its approximation ratio.

1) Algorithm Complexity: In SafeDRL algorithm, the computational complexity in the offline training process is proportional to the size of the training data and the training period. After offline training, we can leverage the trained model to perform online inference, i.e., make backup, deployment and primary instance selection decisions for newly arrived services. Since the training process is run offline, we mainly focus on the computational complexity in the online running process [15].

Theorem 3: In SafeDRL algorithm, the online running process runs in $O(\mathbb{N}\mathbb{M}^2 + \mathbb{M}|\mathcal{E}| + |\mathcal{L}| + |\mathcal{E}|^2)$.

Proof: As shown in Algorithm 4, SafeDRL algorithm includes three algorithms, called DRL algorithm, expert intervention algorithm and pruning algorithm. We first analyze the complexity of DRL algorithm. Assume that there are $\mathbb{N}$ hidden layers in the neural network of DRL algorithm, each layer contains $\mathbb{M}$ neurons. As the RL model, the state size is $3|\mathcal{M}| + 4|\mathcal{E}| + 3|\mathcal{L}| + 2$ and the action size is $2|\mathcal{M}||\mathcal{E}|$. That is, the input and output layers of DRL's neural network contain $3|\mathcal{M}| + 4|\mathcal{E}| + 3|\mathcal{L}| + 2$ and $2|\mathcal{M}||\mathcal{E}|$ neurons. Then, the DRL algorithm runs in $O(\mathbb{N}\mathbb{M}^2 + \mathbb{M}|\mathcal{M}||\mathcal{E}| + \mathbb{M}|\mathcal{L}|)$. Second, we analyze the complexity of the pruning algorithm. In the worst case, with $|\mathcal{M}||\mathcal{E}|$ instances, pruning one instance takes $O(|\mathcal{M}||\mathcal{E}|)$. Checking whether the number of instances of each SF exceeds a threshold needs $O(|\mathcal{M}||\mathcal{E}|)$. Therefore, the pruning algorithm runs in $O(|\mathcal{M}|^2|\mathcal{E}|^2)$. Finally, we analyze the complexity of the expert intervention algorithm. In the worst case, the algorithm adds $\epsilon|\mathcal{M}|$ instances to a service, adding one instance runs in $O(|\mathcal{M}||\mathcal{E}|)$. Besides, in the worst case, there are $\epsilon^{|\mathcal{M}|}$ paths that satisfy the latency constraint, and calculate the bandwidth cost of each one and check its capacity constraint consumption $O(|\mathcal{L}| + |\mathcal{M}|^2)$. Note that, $|\mathcal{M}|$ and ϵ are no more than a small constant. Thus, the expert intervention algorithm runs in $O(|\mathcal{E}| + |\mathcal{L}|)$. Obviously, based on the above analysis, the SafeDRL algorithm runs in $O(\mathbb{N}\mathbb{M}^2 + \mathbb{M}|\mathcal{E}| + |\mathcal{L}| + |\mathcal{E}|^2)$. □

2) Algorithm Approximation: To facilitate the following theoretical analysis, we define two variables

$$\begin{aligned}\tau_{min} &= \min\{\tau_e, \forall e \in \mathcal{E}\},\\ \tau_{max} &= \max\{\tau_e, \forall e \in \mathcal{E}\},\\ \kappa_{min} &= \min\{\kappa_{e,e'}, \forall (e,e') \in \mathcal{L}\},\\ \kappa_{max} &= \max\{\kappa_{e,e'}, \forall (e,e') \in \mathcal{L}\}.\end{aligned} \tag{23}$$

Theorem 4: When all service requests are accepted, SafeDRL algorithm approximates the optimal solution by factor $\max\{\frac{\epsilon\tau_{max}}{\tau_{min}}, \frac{\kappa_{max}}{\kappa_{min}}\}$.

Proof: See Appendix D (available online). □

V. Evaluation

In this section, we first present the evaluation settings, then discuss the simulation results.

A. Settings

Evaluation environment. We implement a Python-based simulator and use PyTorch to build the machine learning framework. All simulations are conducted on a computer with 8-Core Intel(R) Xeon(R) CPU @ 3.19GHz, 16G RAM.

Microservice. In our evaluation, Each service consists of 1-7 microservices, which are typical microservice chain lengths [39]. Besides, the computational and bandwidth resource required by each microservice is derived from the Alibaba *cluster-trace-microarchitecture-v2022* trace data [40]. Besides, the software reliability of microservices is distributed in $[0.995, 0.99999]$ [41]. Moreover, the reliability requirements of services are distributed in [0.99, 0.999999], and the latency requirements of services are distributed in [5-10] ms [5].

Edge Sites. In our evaluation, there are 30 ESs, which are interconnected by overlay links. Each link has a latency in the range of [1-10] ms [42]. The computing resource on each ES is also derived from the Alibaba *cluster-trace-microarchitecture-v2022* trace data. The hardware reliability is distributed in $[0.99 - 0.99999]$, which is obtained from typical infrastructure providers [43]. The cost per unit of computing and bandwidth resources on each ES is distributed between 1-10 units [27], [28], which is normalized.

Parameters. Each neural network has three layers, and each hidden layer contains 1024 neurons [44]. The learning rates for Actor and Critic are 0.001 and 0.003 [14], respectively. The discount factor is 0.99. The penalty parameter ρ is set to 100, which is set based on the resource cost. Besides, we set $\epsilon = 4$, $|\mathcal{I}| = 20$. The requests arrive one by one, in which future requests are unknown. The above parameters are adopted as default settings unless otherwise specified. All the data points are collected from 20 runs.

Algorithms Comparison. We evaluate SafeDRL with the following algorithms.

- DRL: This algorithm is a simplified version of our algorithm, where the action output by DRL [19] is directly applied to the environment.
- EI: The expert intervention (EI) algorithm is a simplified version of our algorithm and is designed based on our insights without DRL assistance.

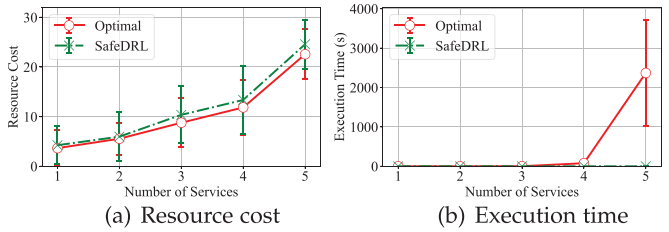


Fig. 6. Compared with the optimal solution in resource cost and execution time.

- RABAI: This is an improved version of RABA [6], where both hardware and software reliability are considered.
- RDSSAI: This is an improved version of RDSSA [9], where DRL is used to make deployment decisions.
- Optimal solution: The optimal solution is obtained by exploring the solution space by the branch and bound method.

B. Simulation Result

1) Optimality: We evaluate the optimality of our algorithm by comparing it with the optimal solution in terms of resource cost and execution time. Since solving the optimal solution is time-consuming, we set $|\mathcal{E}| = 5$, $|\mathcal{M}| = 1$, which is a small-scale special case.[8] The evaluation results are shown in Fig. 6.

As shown in Fig. 6(a), our algorithm is always close to the optimal solution in terms of resource cost. In the worst case, when the number of service requests is 3, the resource cost of our algorithm and the optimal solution is 10.37 and 8.78, respectively. This means that the gap between our algorithm and the optimal solution is no more than 15.34%. Besides, the standard deviation of our algorithm and the optimal solution is no more than 6.71. Therefore, our algorithm has good optimality and is close to the optimal solution, especially in our algorithm, future requests are unknown.[9]

The execution time results of our algorithm and the optimal solution are shown in Fig. 6(b). The execution time of our algorithm is significantly faster than that of the optimal solution. When there are 5 service requests, the execution time of our algorithm is 0.19 s, while it takes 2369 s to obtain the optimal solution. That is, our algorithm is 12468 times faster than the optimal solution. This is because the optimal solution algorithm needs to explore the optimal solution in a huge solution space, which is time-consuming, even after branch and bound. Moreover, the execution time of the optimal algorithm increases exponentially, which is unacceptable. Therefore, we conclude that our algorithm has good optimality and can be well close to the optimal solution while running significantly faster than solving the optimal solution.

2) Superiority: We evaluate the superiority of our algorithm by comparing it with other algorithms in resource cost and service acceptance ratio. The resource cost includes bandwidth resource cost and computing resource cost. The results are shown in Fig. 7 and Fig. 8.

[8]The reason for this setting is to obtain the optimal solution in a suitable time, where the optimal solution is obtained by exploring a huge solution space, which grows exponentially as $|\mathcal{E}|$ and $|\mathcal{M}|$ increases. For example, when $|\mathcal{E}| = 5, |\mathcal{M}| = 2, |\mathcal{I}| = 5$, the optimal solution was not obtained after 24 hours of running, let alone 20 runs for averaging.

[9]In the optimal solution algorithm, all service requests are known.

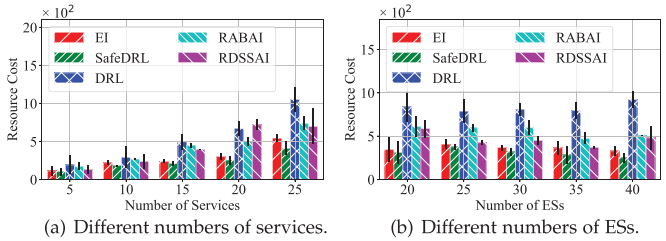


Fig. 7. Compared with other solutions in resource cost.

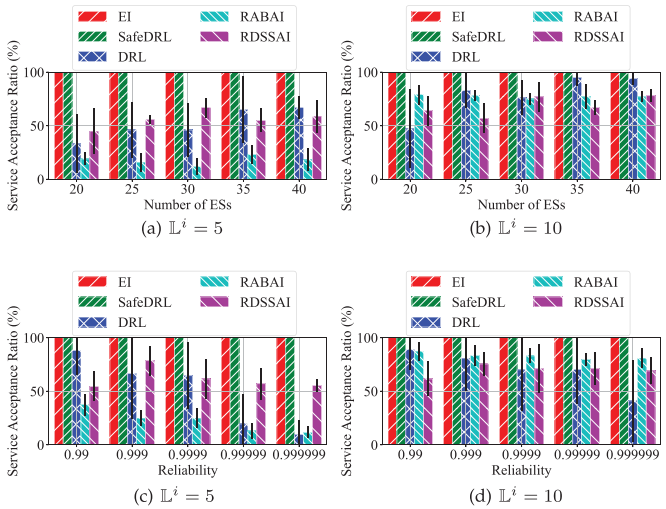


Fig. 8. Compared with other solutions in service acceptance ratio.

As shown in Fig. 7(a), the resource cost of our algorithm is always lower than other algorithms under different numbers of services. When there are 20 ESs, the average resource costs of EI, SafeDRL, DRL, RABAI, and RDSSAI are 300.87, 256.77, 674.38, 506.65, and 719.26, respectively. That is, SafeDRL algorithm can reduce the resource cost by 14.66%, 61.93%, 49.32% and 64.3% compared to EI, DRL, RABAI and RDSSAI. This also means that our algorithm can reduce the resource cost by up to 49.32% compared to existing algorithms (RABAI, RDSSAI). This is because, compared with rule-based algorithms (EI, RABAI), SafeDRL can effectively capture delayed rewards and can more effectively extract the rules behind complex environments by neural networks. Compared with DRL-based algorithms (DRL, RABAI), SafeDRL algorithm can use a pruning algorithm to prune unnecessary instances, which is helpful for exploring the solution space more efficiently and avoiding unnecessary action exploration. Besides, as the number of service requests increases, the resource cost of the four algorithms increases, as more services consume more resource cost. Similar results can be found in Fig. 7(b), the resource cost of our algorithm is always lower than other algorithms under different numbers of ESs. When there are 40 ESs, SafeDRL algorithm can reduce the resource cost by 24.37%, 72.95%, 50.32%, and 48.03% compared to EI, DRL,

RABAI, and RDSSAI. This means that our algorithm can reduce the resource cost by up to 48.03% compared to existing algorithms (RABAI, RDSSAI).

Fig. 8 shows the results of our algorithm and other algorithms on service acceptance ratio under different number of ESs, service latency and reliability requirements. As shown in Fig. 8(a), our SafeDRL and EI algorithms always accept all service requests, while DRL, RDSSAI, and RABAI algorithms suffer from frequent service request rejections. For example, when there are 20 service requests, the service acceptance ratios of algorithms DRL, RABAI, and RDSSAI are 33%, 19%, and 45%. This means that our algorithm can improve the service acceptance ratio of existing algorithms (RABAI and RDSSAI) by 81% and 55%. This is due to the fact that existing reliable service provisioning algorithms ignore latency, which can lead to potential reliability violations. Moreover, DRL and RDSSAI algorithms need to explore the solution space with massive infeasible solutions in a trial-and-error manner, which results in frequent violations of constraints for output actions. In addition, as the number of ES increases, the service acceptance ratio of DRL gradually increases. This is because in DRL algorithm, more ESs lead to more ESs being deployed with instances for each SF. Similar results can be found in Fig. 8(b), SafeDRL and EI can still accept all service requests. In addition, compared to $\mathbb{L}^i = 5$, in DRL, RDSSAI and RABAI algorithms, the service acceptance ratio is higher when $\mathbb{L}^i = 10$. This is because the looser the latency requirements of each service, the fewer paths violate the latency constraints, and the more reliable they are.

As shown in Fig. 8(c), SafeDRL and EI can still accept all service requests, while service requests are frequently rejected in DRL, RDSSAI, and RABAI algorithms under different service reliability requirements. When $\mathbb{R} = 0.99$, the reliability acceptance ratio of DRL, RABAI, and RDSSAI are 88.0%, 37.5%, and 54.5%. Besides, the service acceptance ratio of DRL algorithm gradually decreases with the increase in reliability requirements. This is because higher reliability requirements lead to more infeasible solutions in the solution space, and random exploration will lead to more frequent constraint violations. The service acceptance ratio of RDSSAI algorithm does not decrease significantly with the increase in reliability requirements because it provides more instances for a service when it has higher reliability requirements. Similar results can be found in Fig. 8(d). In conclusion, the above simulation results verify the superiority of our algorithm over other algorithms in terms of resource cost and service acceptance ratio.

Fig. 9 shows the results of our algorithm and other algorithms on the average number of SF backups under different reliability requirements. As the reliability requirement increases, the number of SF backups in all algorithms increases gradually, because higher reliability requires more backup instances. Besides, our algorithm consumes fewer backup instances compared to DRL, since redundant backup instances are pruned. Moreover, our algorithm consumes almost or less backup instances to meet service reliability, which means that our algorithm can meet service requirements more resource-efficiently. Note that

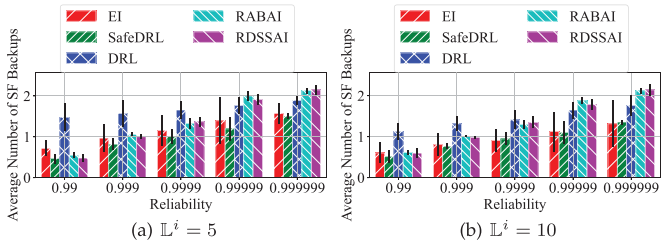


Fig. 9. Compared to other solutions in average number of SF backups.

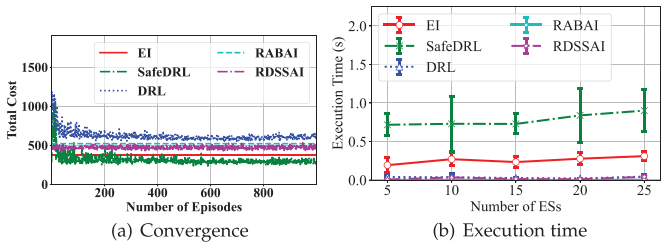


Fig. 10. Compared to other solutions in convergence and execution time.

another reason our algorithm consumes less cost is that it is more cost-efficient. Finally, the number of backups is slightly less as the latency requirements are looser. This is because lower latency requirements result in more paths meeting the latency requirements, which in turn leads to higher reliability and fewer backups.

3) Practicality: We evaluate the practicality of our algorithm by comparing it with other algorithms in terms of convergence and execution time. The results are shown in Fig. 10. As shown in Fig. 10(a), the total cost of RABAI and our EI algorithms is 522.58 and 372.76, respectively. The total cost includes the resource cost and the penalty cost due to constraint violation. This shows that the EI algorithm we designed has good performance. More importantly, both our SafeDRL and DRL gradually converge to a lower resource cost with increasing training episodes, while the total cost of RABAI is jittery. This may be due to the fact that RDSSAI algorithm may be frequently penalized for constraint violations, resulting in non-convergence. Besides, SafeDRL algorithm converges to the lowest value (262.44) among these algorithms. This demonstrates that our SafeDRL has good optimality and convergence. As shown in Fig. 10(b), the execution time of our algorithm is longer than other algorithms. This is due to our algorithm combining three algorithms, which involve more complex processing. Nonetheless, we can find that the average execution time of SafeDRL is always less than 1 s, which is acceptable for service provisioning. Besides, the algorithm can also be accelerated by hardware acceleration, such as GPU.

VI. Conclusion

Achieving high-reliable and low-latency service resources cost-effectively is challenging in dynamic environments, due to the delayed rewards brought by future service requests, high-dimensional resource constraints, and heterogeneous infrastructure and service requests. To capture these challenges, we

formulate the reliable and low-latency service provision problem as a nonlinear integer programming problem, prove its NP-hardness, and analyze its challenges. To address the problem and tackle its challenges, we design a SafeDRL algorithm for learning a policy that maximizes long-term rewards to capture delayed rewards while respecting high-dimensional resource constraints. Specifically, for optimality, we design a pruning algorithm to prune unnecessary instances in the action that do not violate constraints. To cope with high-dimensional constraints, we design an expert intervention algorithm based on our insights to generate high-quality feasible solutions to replace constraint-violating actions. This algorithm is rigorously proved to output solutions with bounded approximation guarantees in general cases. Finally, extensive trace-driven evaluation results verify that our algorithm significantly outperforms the state-of-the-art solution in terms of total resource cost and service acceptance ratio.

Machine learning-driven service provision is an emerging trend. SafeDRL brings DRL and expert knowledge-based heuristics into service provisioning to achieve resource cost efficiency while complying with constraints. Our future work is to further integrate more novel machine learning techniques, such as attention mechanism and contrast learning, to achieve more efficient, flexible, and automated service delivery.

REFERENCES

[1] W. Bao, D. Yuan, B. B. Zhou, and A. Y. Zomaya, "Prune and plant: Efficient placement and parallelism of virtual network functions," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 800–811, Jun. 2020.

[2] Z. Xu et al., "Affinity-aware VNF placement in mobile edge clouds via leveraging GPUs," *IEEE Trans. Comput.*, vol. 70, no. 12, pp. 2234–2248, Dec. 2021.

[3] J. Zheng, Z. Zhang, Q. Ma, X. Gao, C. Tian, and G. Chen, "Multi-resource VNF deployment in a heterogeneous cloud," *IEEE Trans. Comput.*, vol. 71, no. 1, pp. 81–91, Jan. 2022.

[4] H. Rahimi, Y. Picaud, K. D. Singh, G. Madhusudan, S. Costanzo, and O. Boissier, "Design and simulation of a hybrid architecture for edge computing in 5G and beyond," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1213–1224, Aug. 2021.

[5] 3GPP TS 22.261, "Service requirements for the 5G system (Release 18)," Tech. Specification Group Serv. Syst. Aspects, Tech. Rep., 2021.

[6] J. Zhang, Z. Wang, C. Peng, L. Zhang, T. Huang, and Y. Liu, "RABA: Resource-aware backup allocation for a chain of virtual network functions," in *IEEE Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 1918–1926.

[7] J. Covitz, "Resilience at edge computing sites is resilience for the whole IT environment," 2019. [Online]. Available: https://www.networkworld.com/article/3356439/resilience-at-edge-computing-sites-is-resilience-for-the-whole-it-environment.html

[8] J. Fan, C. Guan, Y. Zhao, and C. Qiao, "Availability-aware mapping of service function chains," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2017, pp. 1–9.

[9] J. Jia, L. Yang, and J. Cao, "Reliability-aware dynamic service chain scheduling in 5G networks based on reinforcement learning," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2021, pp. 1–10.

[10] M. Bennis, M. Debbah, and H. V. Poor, "Ultrareliable and low-latency wireless communication: Tail, risk, and scale," *Proc. IEEE*, vol. 106, no. 10, pp. 1834–1853, Oct. 2018.

[11] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.

[12] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin, "Intelligent VNF orchestration and flow scheduling via model-assisted deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 279–291, Feb. 2020.

[13] J. S. P. Roig, D. M. Gutierrez-Estevez, and D. Gündüz, "Management and orchestration of virtual network functions via deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 304–317, Feb. 2020.

[14] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, "Virtual network function placement optimization with deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 292–303, Feb. 2020.

[15] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 263–278, Feb. 2020.

[16] W. Saunders, G. Sastry, A. Stuhlmüller, and O. Evans, "Trial without error: Towards safe reinforcement learning via human intervention," in *Proc. AAMAS*, 2018, pp. 2067–2069.

[17] G. Warnell, N. Waytowich, V. Lawhern, and P. Stone, "Deep TAMER: Interactive agent shaping in high-dimensional state spaces," in *Proc. AAAI*, vol. 32, no. 1, 2018.

[18] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich, "Efficiently combining human demonstrations and interventions for safe training of autonomous systems in real-time," in *Proc. AAAI*, vol. 33, no. 01, 2019, pp. 2462–2470.

[19] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," in *Proc. ICML*, 2016.

[20] 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.

[21] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, "Latency-aware VNF chain deployment with efficient resource reuse at network edge," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2020, pp. 267–276.

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

[23] X. Shang, Y. Huang, Z. Liu, and Y. Yang, "Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2020, pp. 2096–2105.

[24] E. Altman, "Constrained Markov decision processes," Ph.D. dissertation, INRIA, 1995.

[25] C. Tessler, D. J. Mankowitz, and S. Mannor, "Reward constrained policy optimization," in *Proc. ICLR*, 2019.

[26] NFV, GS and others, "Network functions virtualisation (NFV); architectural framework," *NFV ISG*, vol. 2, no. 2, p. V1, 2013.

[27] "Oracle compute pricing," 2022. [Online]. Available: https://www.oracle.com/cloud/compute/pricing.html

[28] "Bandwidth pricing," 2022. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/bandwidth/

[29] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker, "SCL: Simplifying distributed SDN control planes," in *Proc. USENIX NSDI*, 2017, pp. 329–345.

[30] T. ETSI, "ETSI TS 123 502 V16.7.0," *Procedures for the 5G System (5GS)* (3GPP TS 23.502 version 16.7.0 Release 16), 2021.

[31] T. ETSI, "ETSI TS 123 501 V17.5.0," *System Architecture for the 5G System (5GS)* (3GPP TS 23.501 version 17.5.0 Release 17), 2022.

[32] W. Fisher, M. Suchara, and J. Rexford, "Greening backbone networks: Reducing energy consumption by shutting off cables in bundled links," in *Proc. 1st ACM SIGCOMM Workshop Green Network.*, 2010, pp. 29–34.

[33] "End-to-end latency," 2022. [Online]. Available: https://en.wikipedia.org/wiki/End-to-end_delay

[34] M. Ghaznavi, E. Jalalpour, B. Wong, R. Boutaba, and A. J. Mashtizadeh, "Fault tolerant service function chaining," in *Proc. ACM SIGCOMM*, 2020, pp. 198–210.

[35] S. Sahhaf, W. Tavernier, D. Colle, and M. Pickavet, "Adaptive and reliable multipath provisioning for media transfer in SDN-based overlay networks," *Comput. Commun.*, vol. 106, pp. 107–116, 2017.

[36] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proc. ACM SIGCOMM*, 2011, pp. 350–361.

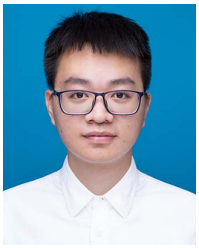
[37] W. Liang, Y. Ma, W. Xu, Z. Xu, X. Jia, and W. Zhou, "Request reliability augmentation with service function chain requirements in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4541–4554, Dec. 2022, doi: 10.1109/TMC.2021.3081681.

[38] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service function chaining use cases in mobile networks," IETF, Tech. Rep., 2019.

[39] S. Luo et al., "Characterizing microservice dependency and performance: Alibaba trace analysis," in *Proc. ACM SOCC*, 2021, pp. 412–426.

[40] "Cluster-trace-microarchitecture-v2022," 2022. [Online]. Available: https://github.com/alibaba/clusterdata

[41] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: A field study of middlebox failures in datacenters," in *Proc. ACM IMC*, 2013, pp. 9–22.
[42] Y. Chen, C. Li, M. Lv, X. Shao, Y. Li, and Y. Xu, "Explicit data correlations-directed metadata prefetching method in distributed file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2692–2705, Dec. 2019.
[43] C. Cérin et al., "Downtime statistics of current cloud solutions," Int. Work. Group Cloud Comput. Resiliency, Tech. Rep., 2014.
[44] H. Zhu, V. Gupta, S. S. Ahuja, Y. Tian, Y. Zhang, and X. Jin, "Network planning with deep reinforcement learning," in *Proc. ACM SIGCOMM*, 2021, pp. 258–271.
[45] D. P. Williamson and D. B. Shmoys, *The Design of Approximation Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2011.



Yue Zeng received the M.S. degree from the Department of Electronic Information Engineering at Southwest University, Chongqing, China, in 2019. He is currently working toward the Ph.D. degree with the Department of Computer Science and Technology at Nanjing University, China. His research interests include network functions virtualization, software defined networking, machine learning for networking, distributed computing, and edge computing.

Zhihao Qu (Member, IEEE) received the B.S. and Ph.D. degrees in computer science from Nanjing University, Nanjing, China, in 2009 and 2018, respectively. He is currently an Associate Professor with the College of Computer Science and Software Engineering at Hohai University. His research interests include the areas of wireless networks, edge computing, and distributed machine learning.

Song Guo (Fellow, IEEE) is currently a full professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology. He is also the Changjiang Chair Professor awarded by the Ministry of Education of China. His research interests include edge AI, mobile computing, and distributed systems. He has been recognized as a highly cited researcher (Web of Science). He was the recipient of more than 14 best paper awards from the IEEE/ACM conferences, journals, and technical committees. He is the Editor-in-Chief of the IEEE OPEN JOURNAL OF THE COMPUTER SOCIETY. He was on the IEEE Communications Society Board of Governors, IEEE Computer Society Fellow Evaluation Committee, and editorial board for a number of prestigious international journals, including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, and the IEEE INTERNET OF THINGS JOURNAL.

Baoliu Ye (Member, IEEE) received the Ph.D. degree in computer science from Nanjing University, China, in 2004. He is a Full Professor with the Department of Computer Science and Technology, Nanjing University. He served as a Visiting Researcher with the University of Aizu, Japan, from March 2005 to July 2006, and the Director of the Division of Informatics, Hohai University currently. His current research interests mainly include distributed systems, cloud computing, and wireless networks with over 70 papers published in major conferences and journals.

Jie Zhang (Member, IEEE) is currently working toward the Ph.D. degree with the Department of Computing, The Hong Kong Polytechnic University. Her current research interests include edge computing, federated learning, and deep reinforcement learning.

Jing Li (Member, IEEE) received the B.Sc. and Ph.D. degrees with the first class Honors from The Australian National University, in 2018 and 2022, respectively. He is currently a Postdoctoral Fellow with The Hong Kong Polytechnic University. His research interests include mobile edge computing, Internet of Things, network function virtualization, and combinatorial optimization. He has published papers in top journals and conferences such as the IEEE TRANSACTIONS ON MOBILE COMPUTING, the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, ACM TOSN, and the IEEE ICDCS.

Bin Tang (Member, IEEE) received the B.S. and Ph.D. degrees in computer science from Nanjing University, Nanjing, China, in 2007 and 2014, respectively. He was an Assistant Researcher at Nanjing University from 2014 to 2020, and also a Research Fellow with The Hong Kong Polytechnic University in 2019. He is currently a Professor with Hohai University. His research interests include the area of communications, network coding, and distributed computing.