FIRE: A Failure-Adaptive RL Framework for Edge Computing Migrations

Marie Siew, Shikhar Sharma, Zekai Li, Kun Guo, *Member, IEEE*, Chao Xu, *Member, IEEE*,
Tania Lorido-Botran, Tony Q.S. Quek, *Fellow, IEEE*, Carlee Joe-Wong, *Senior Member, IEEE*

✦

Abstract—In edge computing, users' service profiles are migrated between edge servers due to user mobility. Reinforcement Learning (RL) frameworks have been proposed to do so, often trained on simulated data. However, existing RL frameworks overlook occasional server failures, which although rare, impact latency-sensitive applications like AR/VR and real- time obstacle detection. These rare failures, being not adequately represented in historical training data, pose a challenge for data-driven RL algorithms. We introduce FIRE, a framework that adapts to rare events by training a RL policy in an edge computing digital twin environment. We propose FIRE-ImRE, an importance sampling-based Q-learning algorithm, which samples rare events proportionally to their impact on the value function. FIRE considers delay, migration, failure, and backup placement costs across individual and shared service profiles. We prove FIRE-ImRE's boundedness and convergence to optimality. Next, we introduce novel deep Q-learning (FIRE-ImDQL) and actor critic (FIRE-ImACRE) versions of our algorithm to enhance scalability. We extend our framework to accommodate users with varying risk tolerances of rare failure events. Through trace-driven experiments, we show that FIRE reduces edge computing costs compared to vanilla RL and the greedy baseline in the event of failures.

Index Terms—Edge Computing, Service migration, Resilient Resource Allocation, Reinforcement Learning

1 INTRODUCTION

Mobile (or multi-access) edge computing (MEC) is a key technology in 5G and 6G telecommunication systems. It enables computationally intensive and latency sensitive mobile applications such as real time image processing, augmented reality (AR), interactive gaming, etc. In MEC, computing resources such as server clusters are positioned close to end-users, typically at cellular base stations or WiFi Access Points (AP) at the edge of the radio access network [1], [2]. This proximity reduces response latency by avoiding the wide-area network delays encountered in cloud computing [2].

Marie Siew and Tony Q.S Quek are with the Information Systems and Technology and Design Pillar, Singapore University of Technology and Design, Singapore 487372.
Shikhar Sharma, Zekai Li, and Carlee Joe-Wong are with the Electrical and Computer Engineering department, Carnegie Mellon University, 15213 Pittsburgh, PA, USA.
Kun Guo is with the Shanghai Key Laboratory of Multidimensional Information Processing, School of Communications and Electronics Engineering, East China Normal University, Shanghai 200241, China.
Chao Xu is with the School of Information Engineering, Northwest A&F University, Yangling 712100, China.
Tania Lorido-Botran is with Roblox, USA.

Researchers have been actively investigating how to jointly allocate computing and radio resources to computing tasks offloaded to edge servers [1], [3]. Nevertheless, user mobility across geographical areas presents a challenge towards task offloading [4]–[6]. Service migration, or moving service profiles to APs nearer the user as they move, has been proposed to maintain service continuity [5], [6]. Yet, frequent service migration causes additional expenditure, due to the required network usage. Therefore, balancing the delay-migration cost tradeoff has been widely studied [4], [5], [7]–[9]. Due to the large decision space when the numbers of APs and time-slots increase, and the lack of information on costs and mobility patterns, Reinforcement Learning (RL) is a common solution method [4], [7], [10]. Such methods often rely on digital twins of the edge computing network to train RL models, as digital twins allow the exploration of different resource allocation policies, without generating real-world consequences.

1.1 Challenges: Edge Computing and Resilience

Few migration works, however, account for another challenge in edge computing: the resilience of edge computing systems to *rare but serious events like server failures*. They occur due to reasons such as system overload, hardware failures, or malicious attacks, and they can be costly: the average cost of an outage at a cloud data center, for example, has increased to $740000 in 2016 [11]. Edge server failures may be even more probable than those in the cloud due to their distributed geographical locations, which complicates management and maintenance [12], though they will likely remain rare overall. Even worse, edge computing failures can have a sizeable impact as many edge computing applications are latency sensitive, such as the Internet of vehicles, AR, and video processing applications [1]. Managing resources without considering rare but severe failure increases latency and reduces reliability. For instance, if the user's service profile is migrated to a server that then fails, its job is not able to be completed, jeopardizing the smooth and safe functioning of applications, such as autonomous driving and real time obstacle detection.

We propose to use *backups*, which can take over when the primary service's edge server fails, to handle failures. Nevertheless, managing the migration of both the primary and backup services is challenging, as the number of APs

and timeslots increases. As mentioned above, due to the large decision space, and the network operator's lack of information, RL has traditionally been proposed to solve the service migration problem [4], [7], [10], [13]. Nevertheless, rare events and failures occur at a low probability, making it difficult to jointly plan or learn an optimal resource allocation policy for both the usual and rare event scenarios. This is because *RL's reliance on past reward data may overlook the significance of rare events, impacting the training process [14].* Therefore, we introduce **FIRE**: Failure-adaptive Importance sampling for Rare Events, a *resilience* framework for edge computing service migration.

1.2 FIRE: Failure-adaptive Importance sampling for Rare Events

Our FIRE framework employs a digital twin - inspired setup, employing RL to optimize service migration and the *backup migration* in edge computing systems, amidst potential rare service failures. A digital twin edge computing system virtually replicates a physical edge computing system and mirrors its network conditions, akin to similar setups used for data center management and remote monitoring [15]. Our framework FIRE further models limited edge server capacity and communication delays, and our optimization balances the trade-offs amongst the following costs: *communication and computing delay costs* of the multiple users, the *migration cost* of service profiles, the *backup storage and migration costs*, along with the *failure costs*.

Because the actual rare event probability is low and unable to help us learn an optimal policy to prepare for rare events, we use **importance sampling of rare events** in our digital twin setup to increase the sampling of the rare events in the RL environment. Error correction is done through the use of importance sampling weights, to enable us to learn an optimal policy with respect to the true rare events probability. Unlike online training, training of RL in the edge computing digital twin also enables the system to learn policies while *avoiding the large failure costs* of experiencing actual server failures. The converged policy trained by the simulator is then applied to online scenarios, in which rare events happen at their natural probabilities. While [14], [16] also use importance sampling to sample rare events, these two works involve estimating the value function given a fixed policy, and their algorithm is not applied in the edge computing scenario. In contrast, FIRE learns, not just evaluates, the optimal policy π^*. We further propose novel boundedness and convergence proofs. *To the best of our knowledge, we are the first to propose importance sampling of rare events to prepare for failures in edge computing.*

We model two decision making scenarios. Firstly, we suppose that every user has an individual service profile for job computation. Secondly, we consider multiple users sharing service profiles, for example common game environments when users are playing the same mobile or AR game together, or common neural networks for multi-modal learning [17]. The service profile replicas can also serve as backups in this second scenario [18], while in the first scenario we must create an individual backup for each user. We further consider that edge computing users may exhibit varying risk tolerances due to differences in their computing applications and latency requirements, or different willingness to incur the backup placement or migration costs.

In summary, our **contributions** are as follows:

- We present a **model** of service migration and backup placements given user mobility that introduces server failures as rare events. We model both the individual and shared service profile settings.
- We introduce **FIRE**, a *resilience* framework designed to handle rare events like server failures. As a digital twin, our importance sampling based Q-learning algorithm **FIRE-ImRE** simulates rare events frequently without real-world consequences, allowing us to learn a failure-aware migration policy. We prove that this algorithm is bounded (Theorem 2), and that it converges to the optimal policy (Theorem 4).
- We propose deep Q-learning (**FIRE-ImDQL**) and actor critic (**FIRE-ImACRE**) versions of FIRE, to handle large and combinatorial state and action spaces in real-world networks. These algorithms differ from traditional deep Q-Learning and actor critic as we incorporate importance sampling and error correcting weights. Additionally, we present the **RiTA** algorithm that adjusts service migration and backup placement based on individual risk tolerance, without separate RL training for different risk tolerances.
- Finally, we provide **trace-driven simulation results** showing the convergence of our algorithms to optimality. We show that unlike vanilla Q-learning, FIRE's variants are resilient towards server failures, resulting in sizeable cost reductions. Our importance sampling framework can be extended to solve other resource allocation and decision making problems in light of rare events in networking scenarios.

The paper is structured as follows. Section 2 discusses the related work. Sections 3 and 4 present the system models, on the individual and shared service profile settings respectively. Section 5 presents the problem formulation. Following which, we present our solution and its proofs in Section 6, and the deep Q-learning and Actor Critic algorithms in Section 7. Next, Section 8 addresses the heterogeneous risk tolerance scenario. Finally, we present simulation results in Section 9 and conclude in Section 10.

2 BACKGROUND AND RELATED WORK

Building on the literature on service and virtual machine (VM) migration in follow-me-cloud, **service or VM placement and migration** in light of user mobility in edge computing has been a widely studied problem [4]–[10], [19]. These are often NP-hard knapsack-like problems. In particular, [4], [7] used dynamic programming and RL to make migration decisions given unknown mobility information. [5] optimized the long term performance-cost trade-off via Lyapunov optimization. Nevertheless, these service migration works have not addressed resilience, including preparation for and adaptation to server failures.

Other works have realized that **cloud and edge computing presents resilience challenges** to resource allocation, including ultra-reliable offloading mechanisms using two-tier optimization [20], proactive failure mitigation for edge-based NFV [21], failure aware workflow scheduling for

cloud computing [22], RL for replica placement in edge computing [23], and graphical models to learn spatio-temporal dependencies between edge server and link failures [11]. The above works did not consider resilience in light of user mobility. The migration of users at overlapping coverage areas has been considered in [24], but here, users at the center of coverage areas still experience a higher latency as they are connected to the cloud. Nevertheless, the above works did not consider unknown user mobility and unknown costs. Our prior work [25] addressed rare events in single-user service migration using an actor-critic importance sampling algorithm. Here, we extend this to multi-user scenarios with both individual and shared service profiles, introduce tabular and deep Q-learning algorithms, and provide theoretical guarantees for the tabular algorithm.

We do not assume that failures can be predicted, unlike most other failure mitigation works. Instead, we aim to learn a policy that builds in resiliency to rare events that are not predicted in advance. Note that our algorithm can complement strategies like checkpointing [26] or dynamically adjusting the number of service backups. It addresses the unique challenge of resiliency in reinforcement learning-based policies, which fundamentally suffer from a lack of rare events in their training datasets.

Other edge computing challenges have been explored from the **risk-awareness** perspective, catering to heterogeneous users and applications, for example risk-aware non-cooperative MEC offloading [27], and energy budget risk-aware application placement [28]. However, they do not consider the effect of rare events on learned RL policies.

(Deep) RL has been used to optimize migration and offloading in edge computing [4], [7], [10], [13], [27] with the goal of optimizing across trade-offs such as migration cost, latency and energy costs. Importance sampling has been used in RL for policy evaluation (i.e., finding the expected future reward from following a given policy π) [14], [16]. However, these two works involve estimating the value function given a fixed policy, instead of learning the optimal policy π^* over time. Their proof techniques do not generalize to the Q-learning [29] that forms the core of FIRE. Recent work has proposed using importance sampling to enhance RL's data efficiency [30] in general, but they do not consider rare events explicitly and require knowledge of the full system dynamics.

3 MIGRATION OF INDIVIDUAL SERVICE PROFILES

In this section, we introduce our failure-aware service migration model, with individual service profiles. In the next section, we will consider shared service profiles. Our system models, which capture rare events, can be integrated into a "digital twin" for offline algorithm learning in light of rare events such as server failures. The purpose of digital twins is to allow the system to learn a policy in a simulated and virtual environment, without actually experiencing the cost of rare events. For example, a simulated edge computing environment may serve as a digital twin [31], [32].

3.1 Service Placement Model

Users and edge servers. We consider an MEC system with K users and N APs (APs), such as a base station or wifi-access point. Each AP is equipped with a server to which

users can offload latency-sensitive computing jobs, and we associate each AP with a disjoint physical region; users in this region then connect to this AP. Users are mobile and move from region to region at different times of the day. For example, office workers may move to the city for work during morning rush hours, changing their nearest AP. User k's location at time t is represented by $l_u^k(t)$. We consider that each user will be associated with its nearest AP, via local area network (LAN). The users' mobility pattern follows a probability distribution $m(l_u'^k | l_u^k)$, where $m(l_u'^k | l_u^k)$ represents the transition probability the user travels to location $l_u'^k$, after being at location l_u^k.

Each user maintains a *service profile* at the edge servers, e.g., in VMs or containers [5], that consists of the state needed to run its edge computing application. Our goal is then to decide where the service profile and any backups should be located. To maintain low latencies for time-sensitive service applications, in light of user mobility, services are pre-migrated - likely to nearby APs. Service migration is stateful, involving the migration of the set of parameters and intermediate results associated with the user's computing job. At each time-slot t, the user sends a service request to the network operator. The location of the user's service at time t is represented by $l_s^k(t)$, and due to pre-migrations the network operator makes this placement decision before the user location at time t is known.

Failure model. In reality, rare events (system anomalies) such as server failures or shutdowns occur. Such events will impact the user's quality of service, e.g., if the user is not able to have his or her job computed on time, impacting the safe and smooth functioning of edge applications and jeopardizing the low latencies of edge computing. We define a failure as any event that prevents a user's service job from being completed, which may stem from either a hardware or a software failure.

We consider two server types in the network. The first type models larger edge datacenters, which tend to have more safety mechanisms like fully duplicated electrical lines with transfer switches that help servers come back online fast after a failure, causing just a slight delay to users. The second type models a smaller edge data center that is less equipped with mechanisms to help deal with failure. These servers take longer to come back online, resulting in a longer delay and higher probability of jobs not being served at all, hence a higher cost. We let $f_{ind}^k(t)$ represent the failure indicator, which indicates whether $l_s^k(t)$, the location of user k's service profile, experiences failure at time t. Failures are Markovian, and under Server Type two, there is a higher probability of failures continuing into the next time-slot.

To take into account the potential failures and reduce the costs experienced by the user, a **backup** of the user's service profile can be placed in the system. This way, if the server $l_s^k(t)$ at which user k's service is placed at experiences a failure, the user will still be able to offload its job. As seen in Fig. 1, unlike the green user which does not have a backup service profile, the brown user is still able to offload its job computation even though the server at AP C failed, because there is a backup at location E.

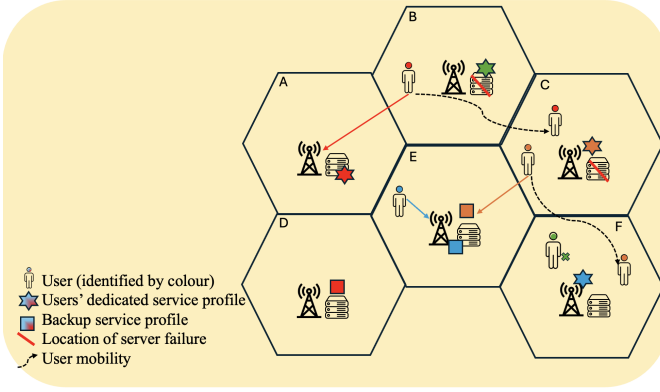We formulate a **Markov Decision Process** (MDP) with

Fig. 1. Users move across the network. Each user has its individual service profile, which has a backup, in case the server where their service profile is at fails. To evaluate system resilience and optimize failure-handling algorithms and strategies without incurring real-world costs, the setup is modeled within a digital twin.

system state $s(t) \in S$ is $[s_1(t), ..s_k(t), ...s_K(t)]$, where

$$s_k(t) = (l_u^k(t), l_s^k(t), f_{ind}^k(t), b_{ind}^k(t)). \quad (1)$$

l_u^k is user k's current location, l_s^k is the location of user k's service, f_{ind}^k is an indicator on whether the user's service is placed at a location with a server failure, and b_{ind}^k indicates whether there currently is a backup for user k's service in the system. At each timeslot t, the network operator takes action $a(t) = [a_1(t), ...a_k(t), ...a_K(t)]$, where

$$a_k(t) = (l_s^k(t+1), b_u^k(t+1)) \quad (2)$$

pre-emptively, before the users moves to their next locations $[l_u^k(t+1)]$. This involves determining the placement location of the service $l_s^k(t+1)$, and the position of the backup in the system $b_u^k(t+1) \in \{0, 1, ...N, \text{no backup}\}$. These decisions will directly impact the system state s at the next timeslot.

Transitions between states are governed by users' movements between APs ($l_u^k(t)$ and $l_u^k(t+1)$) following their mobility pattern $m(l_u'|l_u)$. To quantify them, we let

$$h(\tilde{s}|s, a) = Pr(a) \prod^{K} m(l_u'|l_u), \quad (3)$$

the product of the user's mobility distribution and the probability that the network operator takes a particular action, where $\tilde{s}(t) = [\tilde{s}_1(t), ..\tilde{s}_k(t), ...\tilde{s}_K(t)]$ and

$$\tilde{s}_k(t+1) = (l_u^k(t+1), l_s^k(t+1), b_{ind}^k(t+1)) \in \tilde{S} \quad (4)$$

refers to the state record without the failure indicator f_{ind}. We let the set of these records be $\tilde{S}$. At each state s, there is a small probability $\epsilon(s)$ (e.g., $\epsilon(s) = 0.01$), that in the next timeslot, a server failure occurs in at least one location. The failure indicator f_{ind} is then set to 1. These states belong to the set of "Rare event states", $T \subset S$, defined as follows:

Definition 1. *A subset of states $T \subset S$ is called the Rare Event State Set if the following properties hold:*

1. There exists $s \in S, s' \in T$ for which $p(s'|s, a) > 0$.

2. Let $T^\pi(s)$ denote the contribution of the rare states towards the value of state s, according to Eq. (22). For a given policy π, there exists $s \in S$ for which

$$|T^\pi(s)| \gg 0 \quad (5)$$

3. For every $s' \in T$, $f_{ind} = 1$, indicating that the location of at least one user's service profile l_s^k is experiencing a failure.

Property 1 means that transition to the rare event state set T is possible. Property 2 means that the rare event states (set T) collectively have a sizeable (non-negligible) impact on the value function and hence on the system's cost.

With a larger probability of $1 - \epsilon(s)$, no server failure occurs, i.e. $f_{ind} = 0$. The non rare event states "*Normal States*" are defined as the set $S \backslash T$. Hence we have

$$\begin{aligned} s_k(t+1) &= \tilde{s}_k(t+1) \cup f_{ind}^k(t+1) \\ &= (l_u^k(t+1), l_s^k(t+1), b_{ind}^k(t+1), f_{ind}^k(t+1)) \end{aligned}$$
$$(6)$$

Therefore, the overall state transition probability of the system can be expressed as

$$p(s'|s, a) = \begin{cases} (1 - \epsilon(s))h(\tilde{s}|s, a), & \text{if } s' \notin T \\ \epsilon(s)h(\tilde{s}|s, a), & \text{if } s' \in T, \end{cases} \quad (7)$$

3.2 Reward Function

We next specify the reward function of our MDP. We aim to learn a policy that minimizes the total cost of migration, including operational, failure and backup costs.

3.2.1 Operational Costs

The network operator migrates the users' profiles across APs, to deliver a better QoS (lower job delay) for the user. We first model the costs incurred in normal (non-failure) states, which include communication and computing delays, both of which affect user QoS, as well as the cost of migrating service profiles. We let $d_{l_u,j}^{\text{comm},k}$ denote the communication delay the user faces when it offloads its job to the edge server, given that its current location is l_u^k and its service is placed at AP j. $d_{l_u,j}^{\text{comm},k}$ consists of the access latency of uploading the job to the user's associated AP, and the transfer latency of forwarding the job to the edge server's location if the service is placed at another AP (i.e. $l_u^k \neq j$). This transferring latency via LAN depends on the hop count of the shortest communication path [33]. Therefore, the delay is a function of the distance between l_u^k and j. Therefore, the **total communication delay cost of K users** at time t is

$$D(t) = \sum_{k=1}^{K} \sum_{j} d_{l_u^k,j}^{\text{comm}} \mathbb{1}_{\{l_s^k(t)=j\}}, \quad (8)$$

where $\mathbb{1}_{\{l_s^k=j\}}$ is the indicator function, which will take the value of 1 if the user's service profile is placed at AP j, and will take the value of 0 otherwise.

As multiple users share the resources at an edge node, users will experience a compute delay. Modelling the process at each AP as an M/M/1 queue, the **total computing delay experienced across all APs** at time t is [8]:

$$C(t) = \sum_{j \in A} \sum_{k=1}^{K} \mathbb{1}_{\{l_s^k(t)=j\}} \frac{1}{x_j - \sum_{k=1}^{K} z_k \mathbb{1}_{\{l_s^k(t)=j\}}} \quad (9)$$

where x_j refers to the capacity at AP j, z_k is user k's task size, and $\sum_{k=1}^{K} z_k(t) \mathbb{1}_{\{l_s^k(t)=j\}}$ is the total load at AP j at time t. The total computing delay is the sum over all

locations j and users k, of the average queuing delay each user experiences. As users share computing resources at an AP, the presence of other users' jobs will impact the queuing (i.e. computing) delay experienced by a user.

The migration cost $m_{i,j}$, which includes the operational and energy costs on network devices like routers and switches [5], is a function of the distance across two APs i and j. The **total migration cost of K users** at time t is

$$M(t) = \sum_{k=1}^{K} \sum_{i \in A} \sum_{j \in A} m_{ij} \mathbb{1}_{\{l_s^k(t-1)=i\}} \mathbb{1}_{\{l_s^k(t)=j\}}, \quad (10)$$

where $\mathbb{1}_{\{l_s^k(t-1)=i\}}$ is the indicator function that equals 1 if the service profile was placed at location (AP) i at time $t-1$.

3.2.2 Failure and Backup Costs

We next specify the failure costs and the costs incurred by provisioning backups in case of failures. As mentioned earlier, the system can be modeled within a digital twin setup, allowing algorithm testing and policy learning in a virtual and simulated setting - to avoid experiencing the real-world cost. There is a **backup cost** $B(t)$ incurred by *storing* the backup at a server: a sum of the backup migration cost and the backup storage cost, as it takes up space and prevents other content from being stored.

$$
\begin{aligned}
B(t) = &\sum_{k=1}^{K} \sum_{j \in A} \rho_j \mathbb{1}_{\{b_u^k(t)=j\}} \\
&+ \sum_{k=1}^{K} \sum_{i \in A} \sum_{j \in A} m_{ij} \mathbb{1}_{\{b_u^k(t-1)=i\}} \mathbb{1}_{\{b_u^k(t)=j\}},
\end{aligned} \quad (11)
$$

where ρ_j is the storage cost at AP j. There is also a **failure cost** $F(t)$ incurred

$$F(t) = \sum_{k=1}^{K} F \mathbb{1}_{\{f_{ind}^k(t)=1\}} \left(\mathbb{1}_{\{b_u^k(t)=\text{no backup}\}} \vee \mathbb{1}_{\{b_u^k(t)=l_s^k(t)\}} \right) \quad (12)$$

This cost is experienced at time t when i) location $l_s^k(t)$ has a failure at time t ($f_{ind}^k(t) = 1$), and ii) there is no backup in the system ($b_u^k(t) = $ no backup), or when the placement of the backup $b_u^k(t) = l_s^k(t)$, the location of the main service. Otherwise, if there is a backup placed, there will be no failure cost. The presence of failures themselves, i.e., $f_{ind}(t)$, is however independent of the user actions.

4 MIGRATION OF SHARED SERVICE PROFILES

In this section, we introduce the edge computing service migration scenario where, differing from Section 3, multiple users share a common service profile (SP) for the same task. This could be a common neural network for multi-modal learning, or a common game environment for their AR or mobile gaming applications such as when multiple users are playing a game together [17]. Therefore, the network operator is concerned with the placement of shared SPs across the network, in light of the geographical demand distributions of the different SPs. In the event of a server failure, the impact will be multiplied when SPs are shared, as more users cannot have their job served.
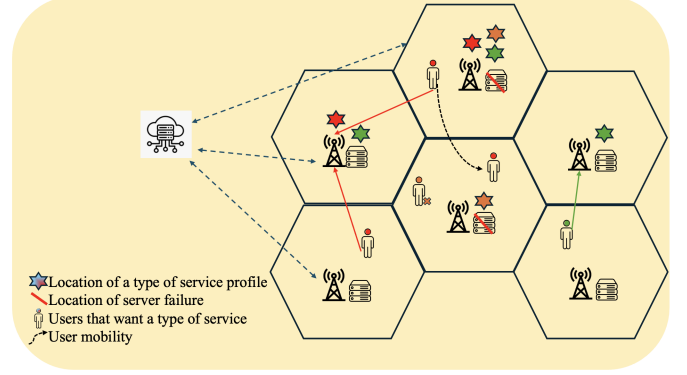

Fig. 2. Multiple users (who are mobile across the network) share a service profile (SP) for their task, e.g. a common game environment or neural network. For instance, the users in red share the SP in red, which has multiple copies in the network.

As in the case of individual service profiles, we formulate a **Markov Decision Process** (MDP) in which the system state is $[s_{1,1}(t), ..s_{i,j}(t), ...s_{I,J}(t)]$, where for each location, SP pair (i, j), their component of the state space at time t is

$$s_{i,j}(t) = (u_{i,j}(t), d_{i,j}(t-1), f_i(t), t_i(t)), \quad (13)$$

where $u_{i,j}(t)$ is the number of users who want to use SP j at location i, $d_{i,j}(t-1)$ indicates whether SP j has been previously deployed at location i, and $f_i(t)$ and $t_i(t)$ denote the failure status and the server type at location i respectively. Here, there are multiple identical copies of profiles being placed, as many users share them (see Fig. 2). Note that this state space (concatenation over all location - SP pairs) is different from that in the individual service profile scenario in Section 3 (concatenation over individual user information). Here the placement decisions depend on the location and mobility distributions of the users who *share* the profile, whereas in Section 3 the placement of every profile is dependent on the location of the corresponding individual user. The corresponding action space will be

$$[(l_{\text{loc1}}^{SP1}, l_{\text{loc2}}^{SP1}, ..., l_{\text{locN}}^{SP1}), ...(l_{\text{loc1}}^{SPj}, l_{\text{loc2}}^{SPj}, ...), ...], \quad (14)$$

where $l_{\text{loc } i}^{SPj} \in \{0, 1\}$ is a binary variable indicating whether we place or migrate SP j at location i.

As in the case of individual service profiles, we aim to learn a policy that minimizes the total costs, which include both operational and backup and failure-related costs.

The **total communication delay** cost across all SP types $j \in SP$ and locations $i \in A$, at time t is

$$D^{sh}(t) = \sum_{j \in SP} \sum_{i \in A} u_{ij}(t) \min_{\hat{i} \in \{\hat{i}|l_{\text{loc } \hat{i}}^{SPj}=1\}} d_{i,\hat{i}}^{comm}, \quad (15)$$

where $d_{i,\hat{i}}^{comm}$ is the communication delay from location i to $\hat{i}$, same as Section 3. The users at location i access the nearest AP which contains the SP type they require ($\min_{\hat{i} \in \{\hat{i}|l_{\text{loc } \hat{i}}^{SPj}=1\}} d_{i,\hat{i}}^{comm}$). QoS is also affected by the **total computing delay experienced across all APs** at time t [8]:

$$C^{sh}(t) = \sum_{i \in A} \frac{1}{x_i - \sum_{j \in SP} \sum_{\hat{i} \in A} load_{j,\hat{i}}(t)} \quad (16)$$

where x_i is the capacity of the servers at AP i. The load at each location i is a sum over all SP types j, all other

locations $\hat{i}$ (inclusive of i). The load $load_{j,\hat{i}}(t) = z_j u_{i,j}(t) \wedge \mathbb{1}_{\{i=\text{argmin}_{\{m \mid l^{SPj}_{loc\ m}=1\}} d^{comm}_{m,\hat{i}}\}}$, the product of task j's size z_j, the number of users at location $\hat{i}$ who need SP type j, $u_{i,j}(t)$, and the indicator function indicating that location i is the nearest location containing SP j, to the users at $\hat{i}$. As in the case of individual service profiles, service profiles at the same AP must share resources, leading to increased delays.

The **total migration cost** at time t is

$$M^{sh}(t) = \sum_{j \in SP} \mathbb{1}_{\{d_{i,j}(t-1)=0\ \&\ d_{i,j}(t)=1\}} \times \big[g_{cloud,i}$$

$$\mathbb{1}_{\{d_{\hat{i},j}(t-1)=0,\ \forall \hat{i}\}} + \min_{\{\hat{i} \mid d_{\hat{i},j}(t-1)=1\}} m_{\hat{i},i} \mathbb{1}_{\{\exists \hat{i}\ s.t.\ d_{\hat{i},j}(t)=1\}} \big].$$

The migration cost for SP j kicks in if it is to be placed at location i at time t and was not placed there at $t-1$ (i.e. $d_{i,j}(t-1) = 0\ \&\ d_{i,j}(t) = 1$). It consists of either downloading SP j from the cloud server for a new deployment when $d_{\hat{i},j}(t-1) = 0,\ \forall \hat{i}$, or migrating SP j from the nearest location $\hat{i}$ when $\exists \hat{i}\ s.t.\ d_{\hat{i},j}(t) = 1$. The **total storage cost** of storing all the deployed SPs is

$$S^{sh}(t) = \sum_{j \in SP} \sum_{i \in A} h_j \rho_i \mathbb{1}_{\{l^{SPj}_{\text{loc}\ i}(t)=1\}}, \tag{17}$$

where h_j is the size of service profile type i and ρ_i is the per unit storage cost at AP i. The **failure cost** is incurred for the users who need to use service profile type j when either a) SP type j is not placed anywhere ($l^{SPj}_{\text{loc}\ i} = 0\ \forall i$), b) the delay exceeds del^j, the delay threshold for this application ($\min_{\hat{i} \in \{\hat{i} \mid l^{SPj}_{\text{loc}\ \hat{i}}=1\}} d^{comm}_{i,\hat{i}} > del^j$), or c) there is a failure at a location $\hat{i}$, and there is no available SP j placed elsewhere, or the minimum delay elsewhere exceeds the delay threshold del^j for this application.

$$F^{sh}(t) = \sum_{j \in SP, i \in A} u_{ij} \big[\mathbb{1}_{\{l^{SPj}_{\text{loc}\ i}=0\ \forall i\}}$$

$$\vee\ \mathbb{1}_{\{\min_{\hat{i} \in \{\hat{i} \mid l^{SPj}_{\text{loc}\ \hat{i}}=1\}} d^{comm}_{i,\hat{i}} > del^j\}} \vee \Pi_{\hat{i}} \big(\mathbb{1}_{\{f_i(t)=1\}}$$

$$\wedge\ \big(\mathbb{1}_{\{l^{SPj}_{\text{loc}\ i}=0\ \forall i \neq i\}} \vee \mathbb{1}_{\{\min_{\hat{i} \in \{\hat{i} \neq i \mid l^{SPj}_{\text{loc}\ \hat{i}}=1\}} d^{comm}_{i,\hat{i}} > del^j\}} \big) \big) \big]$$

5 COST MINIMIZATION IN LIGHT OF RARE EVENTS

In this section, we introduce the service migration optimization problems, for both the individually used (Section 3) and shared (Section 4) service profile scenarios. Our focus is on long term service placements and backup storage decisions, taking into account potential rare events such as server shutdowns ($f^k_{ind}(t) = 1$) which have an impact on the user experience. For both problems, we aim to minimize the expected sum of the **weighted sum of the communication and computing delay, migration, storage and failure costs**. Actions are taken in a digital twin framework, as they allow for algorithm training while avoiding the cost of failures in real-world applications. Firstly, we introduce the optimization problem for the scenario in Section 3 where all users have their individual service profiles.

$$\min_{\pi(s)} \mathbb{E}_\pi \big[\sum_t (w_D D(t) + w_C C(t) + w_M M(t) + w_B B(t)$$

$$+ w_F F(t)) \big] \tag{18}$$

s.t. $[s_1(t),..s_k(t),...s_K(t)] \sim p(s'|s,a)$.
Eqs.$(8),(10),(9),(11),(12)$, $\qquad$ (19)

where w_D, w_C, etc. are the corresponding weights, which can be adjusted based on the priority of different costs. The decision variable $\pi(s,a) = \{P_r(a_t = a|s_t = s)\}$ is the network operator's policy, the probability it will take each action $a(t)$ (service placement and backup storage decision for all users) given the state $s(t)$. The first constraint indicates that the state dynamics follows the transition probability of the system in Eq. (7), incorporating rare event transitions, while the other constraints indicate the delay, migration, backup and failure costs respectively.

Next, we introduce the optimization problem for the scenario in Section 4 where users share service profiles; note that we aim to minimize the same types of costs as in Eq. (18) with individual service profiles.

$$\min_{\pi(s)} \mathbb{E}_\pi \big[\sum_t (w_D D^{sh}(t) + w_C C^{sh}(t) + w_M M^{sh}(t)$$

$$+ w_S S^{sh}(t) + F^{sh}(t)) \big] \tag{20}$$

s.t. $[s_{1,1}(t),..s_{i,j}(t),...s_{I,J}(t)] \sim p(s'|s,a)$.
Eqs.$(15),(17),(16),(17),(18)$. $\qquad$ (21)

The decision variable $\pi(s,a)$ is the network operator's policy, the probability it will take each action $a(t)$ (placement decisions of the service profiles) given the current state $s(t)$. The first constraint indicates that the state dynamics follows the system's transition probability, which incorporates rare event transitions, while the other constraints indicate the delay, migration, storage and failure costs.

Proposition 1. *The number of migration path possibilities grows at least fast as $O(N^T)$.*

Proof. For each timeslot t, there are N possible locations to migrate a service profile. Since the service profile must be placed at a server each timeslot, the result follows. $\qquad \square$

Deciding a migration path for the user across T timeslots is similar to the shortest path problem. However, the network operator may not have information on mobility distributions and the corresponding latency costs, complicating look-ahead solutions. Furthermore, the costs (e.g. computing delays, Eqs. (9) and (16)) of different users are coupled through service profiles' sharing of AP resources. Thus, Markov Decision Processes and RL have been used to derive service migration policies [7], [10], [13]. The difficulty in our problem further arises as we model occasional server failures. As these failures occur at a low probability, and because RL relies on past reward data, the learner might overlook rare events and fail to learn an optimal failure-adaptive policy. We introduce our solution in the next section.

6 RL IN THE PRESENCE OF SERVER FAILURES

In this section, we present FIRE: Failure-adaptive Importance sampling for Rare Events, a RL based framework in an edge computing digital twin. Our algorithm is applied to solve both optimization problems introduced in Section 5. While [16] and [14] also use importance sampling to sample rare events, these two works involve estimating the

value function given a fixed policy. Their proof techniques also do not generalize to the Q-learning. In contrast, our algorithms involve learning the optimal policy π^*, which involves policy changes. Based on our algorithm, we propose novel boundedness and convergence proofs. We present our algorithm (FIRE-ImRE) and its proof in Subsections 6.2 and 6.3 respectively, and present an eligibility traces version (ETAA) for faster convergence in the Appendix. We propose a Deep Q learning and Actor Critic version of our algorithm in Section 7 to accommodate large state and action spaces. FIRE is compatible with any RL formulation, and our algorithms are examples of this.

6.1 Rare Events and the Value Function

We define rare event states as occasions when service fails: $f_{ind}^k(t) = 1$ in Section 3 or $f_i(t) = 1$ in Section 4. We are concerned with rare event states if they have a sizeable impact on the user experienced costs, i.e. if these states collectively have an impact on the value function. The value function $V^\pi(s) : S \to \mathbb{R}$ for the policy π is the expected-return of state s, i.e. it indicates how "good" (cost wise) a state is (based on the discounted sum of future rewards), when π is used. The discount factor γ indicates how much the network operator cares about future vs current rewards.

$$V^\pi(s) = \mathbb{E}_{a_t, p(s'|s,a)}\left[\sum_{t=1}^\infty \gamma^k r(s(t), a(t), a(t+1))|s_0 = s\right]$$

where the reward (cost) is the per-timeslot cost in (18) or (20). The value function is the solution to the recursive Bellman equation [29]:

$$V^\pi(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')].$$

Defining $T^\pi(s)$ as the collective contribution of the rare states towards $V^\pi(s)$ [14] given a fixed policy π, where the rare states T are as defined in Section 3.1, we have

$$T^\pi(s) = \epsilon(s) \sum_{a \in A} \sum_{\tilde{s}' \in \tilde{S}} h(\tilde{s}'|s,a)\pi(a|s)[r(s,a,s') + \gamma V^\pi(s')] \tag{22}$$

Likewise, defining $U^\pi(s)$ as the collective contribution of the non-rare states (states in $S\backslash T$) towards the value of state s, given a fixed policy π, we have :

$$U^\pi(s) = (1 - \epsilon(s)) \times$$
$$\sum_{a \in A} \sum_{\tilde{s}' \in \tilde{S}} \pi(a|s) h(\tilde{s}'|s,a)[r(s,a,s') + \gamma V^\pi(s')]. \tag{23}$$

The Q-value of a state action pair (s,a) is the expected-return of taking action a at state s, following π.

$$Q^\pi(s,a) = \mathbb{E}_\pi[\sum_{k=1}^\infty \gamma^k r(s(t), a(t), s(t+1))|s_t = s, a_t = a]$$

$$= \sum_{s' \in S} p(s'|s,a)[r(s,a,s') + \gamma \sum_{a' \in A} \pi(a'|s')Q^\pi(s',a')]$$

The optimal policy will achieve the minimum cost and minimum value function:

$$Q^*(s,a) = \min_\pi Q^\pi(s,a). \tag{24}$$

6.2 FIRE-ImRE: Q-Learning in Light of Rare Events

In this subsection, we present our importance sampling based failure-adaptive algorithm. We use a digital twin-based simulator in order to avoid the large failure costs experienced during server failures in online scenarios, and the converged policy trained in the digital twin is then applied to online scenarios, in which rare events happen at their natural probabilities. Our algorithm performs importance sampling while increasing the rare event probabilities from $\epsilon(s)$ to $\hat{\epsilon}(s)$. The transition probabilities $h(\tilde{s}|s,a)$, which mostly depend on the user mobility probabilities (Eq. (3)), are obtained from historical data. We use importance sampling because if rare events such as server failures are sampled at their natural probabilities, the RL algorithm is not able to converge to the optimal policy; in particular, it may decide that the storage cost isn't worth having backups. This can be catastrophic in the event of failures. In our numerical simulations in Section 9, we show that unlike our algorithm, a traditional RL algorithm is unable to train an optimal policy which avoids experiencing high cost in the event of failures (Figs. 5 - 7).

Idea behind algorithm: Firstly, we introduce importance sampling: when we want to calculate the expectation of a function $f(x)$, and the true distribution p is difficult to sample from, importance sampling involves sampling from another distribution q, to help compute $E[f(x)]$ [34]. Here, the expectation of $f(x)$ can be estimated as

$$E[f(x)] = \int f(x)p(x)dx \tag{25}$$

$$= \int f(x)\frac{p(x)}{q(x)}q(x)dx \tag{26}$$

$$\approx \frac{1}{n}\sum_i f(x_i)\frac{p(x_i)}{q(x_i)}. \tag{27}$$

When q is well crafted, variance in the estimates will be reduced. For our service migration and backup placement problem, we will sample the rare events at probability $\hat{\epsilon}(s)$ (analogous to $q(x)$), a higher rate than $\epsilon(s)$ (analogous to $p(x)$), which is the probability of visiting a rare event state from state s. Therefore, to obtain $\hat{\epsilon}(s)$, we define $T(s,a)$ as the contribution of the rare states $s' \in T$ towards the Q-value of the state action pair (s,a), and $U(s,a)$ as the contribution of the normal states $s' \in S\backslash T$ towards the Q-value of the state action pair (s,a):

$$T(s,a) = \epsilon(s) \sum_{\tilde{s}' \in \tilde{S}} h(\tilde{s}'|s,a)[r(s,a,s') + \gamma \max_b Q(s',b)] \tag{28}$$

$$U(s,a) = (1-\epsilon(s)) \sum_{\tilde{s}' \in \tilde{S}} h(\tilde{s}'|s,a)[r(s,a,s')+\gamma \max_b Q(s',b)] \tag{29}$$

where $s' = \tilde{s}' \cup f_{ind}$. With this, we have the relationship $T(s,a) + U(s,a) = Q(s,a)$. A potential update equation would be $T(s,a) \leftarrow (1 - \alpha_T)T(s,a) + \alpha_T\epsilon(s)(r(s,a,s') + \gamma \max_b Q(s',b))$, where α_T is the learning rate.

Updating at the state action (s,a) level may lead to slower convergence, especially with an increasing number of APs. For example, with 9 APs, there will be 360 states and 90 actions, leading to 32400 state-action combinations. We can reduce this complexity by exploiting our earlier

observation that *the probability of rare events (i.e., server failures) happening is independent of the actions chosen*. Therefore, we propose updating T and U (the contributions of the rare events and non-rare events respectively towards the Q-value) at the state level:

$$T(s) = \epsilon(s) \sum_{a \in A} \sum_{\tilde{s}' \in S} \pi(a|s) h(\tilde{s}'|s,a) [r(s,a,s') + \gamma \max_b Q(s',b)]$$

$$U(s) = (1 - \epsilon(s)) \times \sum_{a \in A} \sum_{\tilde{s}' \in S} \pi(a|s) h(\tilde{s}'|s,a) \big[r(s,a,s') + \gamma \max_b Q(s',b) \big]$$

These equations contain a sum over the actions. The relationship $U(s) + T(s) = \sum_a Q(s,a)\pi(a|s)$ holds. To help us obtain the importance sampling rare event rate $\hat{\epsilon}(s)$, We will use the following respective update equations for $T(s)$ and $U(s)$ in our algorithm:

$$T(s) \leftarrow (1 - \alpha_T)T(s) + \alpha_T \epsilon(s)(r(s,a,s') + \gamma \max_b \hat{Q}(s',b))$$

$$U(s) \leftarrow (1 - \alpha_U)U(s) + \alpha_U(1 - \epsilon(s))(r(s,a,s') + \gamma \max_b \hat{Q}(s',b)).$$

Importance sampling and correction: Based on the above, we will calculate the rare event importance sampling rate $\hat{\epsilon}(s)$ as follows:

$$\hat{\epsilon}(s) \leftarrow \min(\max(\delta, \frac{|T(s)|}{|T(s)| + |U(s)|}), 1 - \delta). \qquad (30)$$

The bounds $(\delta, 1 - \delta)$ ensure sufficient rare event sampling. As importance sampling of rare events takes place according to $\hat{\epsilon}(s)$, $\hat{p}(s'|s,a)$ is the transition probability in our algorithm which incorporates importance sampling. It follows Eq. (4), with $\epsilon(s)$ being replaced by $\hat{\epsilon}(s)$.

Because the rare events are sampled at the probability $\hat{\epsilon}$ instead of their actual probability ϵ, we need a method of correction, in order to learn the optimal policy for the original system with transition probability $p(s'|s,a)$ (Eq. (7)). Importance sampling correction weights $w(s,a,s')$ will be used, when obtaining the temporal-difference (TD) error in Q-learning. They are obtained through:

$$w(s,a,s') \leftarrow \begin{cases} \epsilon(s)/\hat{\epsilon}(s), & \text{if } s' \in T, \\ (1 - \epsilon(s))/(1 - \hat{\epsilon}(s)), & \text{if } s' \notin T, \end{cases} \qquad (31)$$

where $1 - \epsilon(s)$ is the probability that s' is a non-rare event state. The TD error will be

$$w_t(r(s,a,s') + \gamma \max_b \hat{Q}(s',b)) - \hat{Q}(s,a), \qquad (32)$$

instead of the traditional Q-learning TD error $r(s,a,s') + \gamma \max_b \hat{Q}(s',b) - \hat{Q}(s,a)$.

Algorithm description: The algorithm is presented in Algorithm 1 (**FIRE-ImRE**). Firstly, we initialise $\hat{Q}(s,a)$, $\hat{T}(s)$ and $\hat{U}(s) = 0$, and $\hat{\epsilon}(s) = \frac{1}{2}$, for all states, and initialise the learning rates $\alpha^t, \alpha_T^t, \alpha_U^t$. For every timeslot, the importance sampling rare event probability $\hat{\epsilon}(s)$ will determine whether or not a rare-event (failure) occurs. Thereafter, the rest of the state transition will occur according to the probability distribution $h(\tilde{s}|s,a)$, where $\tilde{s}(t) = (l_u(t), l_s(t), b_{ind}(t))$.

This would result in a new state s^{t+1}, and a reward value $r^{t+1} = r(s^t, a^t, s^{t+1})$ (line 4). Based on the next state s^{t+1}, a new action is selected according to the β-greedy policy (lines 5-6), which is the same as the traditional ϵ-greedy policy in Q-learning, where with probability β the greedy action is selected and with probability $1 - \beta$ a random action is selected, for exploration. We call it β-greedy to avoid confusion with our rare event probability ϵ.

Next, we obtain the importance weight w_t for this timeslot. The importance weight is the actual probability divided by the importance sampling - based probability (line 7). The temporal-difference (TD) error $\triangle_t$ will be updated (line 8), while involving error correction using the importance sampling weight w_t. With the TD-error, we update $\hat{Q}(s^t, a^t)$, the Q-value for state s^t and action a^t (line 9). The size of update is determined by the learning rate α^t. We then update either $T(s^t)$ or $U(s^t)$, depending on whether the next state s^{t+1} is a rare event state or not (lines 10-14). Finally, the importance sampling rare events probability $\hat{\epsilon}(s^t)$ is updated in line 15, according to $\frac{|T(s^t)|}{|T(s^t)| + |U(s^t)|}$, and bounded by δ and $1 - \delta$. The process iterates for every timeslot, until we have achieved convergence. The converged policy will be applied online.

Algorithm 1 FIRE-ImRE: Importance Sampling Q-Learning for Rare Events

---

1: **Initialise:** $\hat{Q}(s,a)$ randomly, $\hat{T}(s), \hat{U}(s) \leftarrow 0$, $\hat{\epsilon}(s) \leftarrow \frac{1}{2}$, $\alpha^t, \alpha_T^t, \alpha_U^t$.
2: Select the initial state s^0 and action a^0.
3: **for** all timeslots t **do**
4:     $\hat{\epsilon}(s^t)$ determines if an rare event happens. Thereafter, sample according to $h(\tilde{s}|s,a)$. The new state s^{t+1} and a reward value $r^{t+1} = r(s^t, a^t, s^{t+1})$ is observed.
5:     $a^{t+1} \leftarrow \beta - \text{greedy}(s^{t+1})$.
6:     $a^* \leftarrow \text{argmax}_b \hat{Q}(s^{t+1}, b)$.
7:     $w_t \leftarrow \begin{cases} \epsilon(s^t)/\hat{\epsilon}(s^t), & \text{if } s^{t+1} \in T. \\ (1 - \epsilon(s^t))/(1 - \hat{\epsilon}(s^t)), & \text{if } s^{t+1} \notin T. \end{cases}$
8:     $\triangle_t \leftarrow w_t(r^{t+1} + \gamma \hat{Q}(s^{t+1}, a^*)) - \hat{Q}(s^t, a^t)$
9:     $\hat{Q}(s^t, a^t) \leftarrow \hat{Q}(s,a) + \alpha^t \triangle_t$.
10:     **if** $s^{t+1} \in T$ **then**
11:         $T(s^t) \leftarrow (1 - \alpha_T^t)T(s^t) + \alpha_T^t \epsilon(s^t)(r^{t+1} + \gamma \hat{Q}(s^{t+1}, a^*))$
12:     **else**
13:         $U(s^t) \leftarrow (1 - \alpha_U^t)U(s^t) + \alpha_U^t(1 - \epsilon(s^t))(r^{t+1} + \gamma \hat{Q}(s^{t+1}, a^*))$
14:     **end if**
15:     $\hat{\epsilon}(s^t) \leftarrow min(max(\delta, \frac{|T(s^t)|}{|T(s^t)| + |U(s^t)|}), 1 - \delta)$
16: **end for**

Our algorithm works for the scenario where $\exists s, \text{s.t.} |T(s)| \gg 0$. See Statement 2 in the definition of rare events in Definition 1. This condition means that the rare events have a sizeable contribution to the Q-value of at least one other state action pair (s,a), and hence collectively have a sufficient impact on the reward/cost of the system. When this condition is not met, the cost of rare events to the users and network operator is not sufficiently high enough, resulting in less of a need to deal with rare events and use importance sampling to capture their impact on the cost.

6.3 Boundedness and Convergence Properties

In this subsection, we prove the convergence of **FIRE-ImRE**, our importance sampling based Q-learning algorithm (Algorithm 1).

Firstly, we show that the sequence of updates in Algorithm 1 is bounded, in the following theorem.

Theorem 2. *For stepsizes that satisfy $\sum_t \alpha(t) = \infty$ and $\sum_t \alpha^2(t) \leq \infty$, and discount factor $\gamma \in (0, 1)$, the sequence of the $\hat{Q}$ updates in Algorithm 1 is bounded, with probability 1.*

Proof. See our Technical Report [35]. □

The main idea of the proof is as follows: We rewrite Algorithm 1's update equation, through adding and subtracting terms. We rewrite the equation in the form of $\hat{Q}^{t+1}(s, a) = \hat{Q}^t(s, a) + \alpha^t[F_{s,a}(\hat{Q}^t) - \hat{Q}^t(s^t, a^t) + M_s(t)]$. Invoking Theorem 1 of [36] on stochastic approximation algorithms, we finally prove convergence by showing that the conditions for the sequence $F_{s,a}(\hat{Q})$ to be bounded are met. We do this by showing that $||M(t)||$ is bounded, and that the property $||F(\hat{Q})||_\infty \leq b||\hat{Q}||_\infty + d$ holds, where $||\ ||_\infty$ is the supremum norm.

Next, to help us prove that our algorithm FIRE-ImRE converges to optimality, we invoke the following corollary from [37]:

Corollary 3. *The random process $\{\triangle_t\}$ taking values in $\mathbb{R}^n$ and defined as $\triangle_{t+1}(x) = (1 - \alpha_t(x))\triangle_t(x) + \alpha_t F_t(x)$ converges to zero with probability 1 when the following assumptions hold:*

$$0 \leq \alpha_t \leq 1, \sum_t \alpha_t(x) = \infty \ and \ \sum_t \alpha_t^2(x) < \infty.$$

$$||\mathbb{E}[F_t(x)]|| \leq \gamma||\triangle_t||, where \ \gamma < 1.$$

$$var[F_t(x)] \leq C(1 + ||\triangle||^2), \ for \ C > 0.$$

In the following theorem, we show that our algorithm converges, and that it converges to the optimal policy.

Theorem 4. *If the MDP underlying the RL environment is unichain for $\epsilon \in (\delta, 1-\delta)$,[1] for stepsizes satisfying $0 \leq \alpha_t \leq 1$, $\sum_t \alpha_i(t) = \infty$ and $\sum_t \alpha_i^2(t) \leq \infty$, and discount factor $\gamma \in (0,1)$, Algorithm 1 converges to the optimal policy Q^*.*

Proof. See our Technical Report [35]. □

The main idea of the proof is as follows. Firstly, we show that the optimal policy of the true system is a fixed point of the following equation with our simulator's transition probability $\hat{p}(s'|s, a)$ and the importance sampling correction weight $w_t(s)$: $Q^*(s, a) = \sum_{s' \in S} \hat{p}(s'|s, a)w_t(s)[r(s, a, s') + \gamma \max_b Q^*(s', b)]$. We define the operator $\mathbf{H}\hat{Q} = \sum_{s' \in S} \hat{p}(s'|s, a)w_t(s)[r(s, a, s') + \gamma \max_b \hat{Q}(s', b)]$, and having shown that that $\mathbf{H}Q^* = Q^*$, we next show that $\mathbf{H}\hat{Q}$ is a contraction mapping, through showing that $||\mathbf{H}\hat{Q}_1 - \mathbf{H}\hat{Q}_2||_\infty \leq \gamma||\hat{Q}_1 - \hat{Q}_2||_\infty$. Next, we invoke the result of Corollary 3 on the convergence of random processes. We rewrite the update equation of Algorithm 1, such

---

1. $(\delta, 1 - \delta)$ is the range of values which $\hat{\epsilon}$ takes, as defined in Algorithm 1. If the MDP defined by transition probability $p(s'|s, a)$ is unichain for one value in $(\delta, 1 - \delta)$, it is unichain for all values [14].

Algorithm 2 FIRE-ImDQL: Importance Sampling for Rare Events using Deep Q Learning

1: **Initialise:** $\hat{Q}(s, a, \boldsymbol{\theta}^-)$ and $Q(s, a, \boldsymbol{\theta})$ randomly, $\hat{T}(s), \hat{U}(s) \leftarrow 0$, $\hat{\epsilon}(s) \leftarrow \frac{1}{2}$.
2: Select the initial state s^0 and action a^0.
3: **for** all timeslots t **do**
4: $\quad \hat{\epsilon}(s^t)$ determines if an anomaly happens. Thereafter, sample according to $h(\tilde{s}|s, a)$. The new state s^{t+1} and a reward value $r^{t+1} = r(s^t, a^t, s^{t+1})$ is observed.
5: $\quad a^{t+1} \leftarrow \beta - \text{greedy}(s^{t+1})$.
6: $\quad a^* \leftarrow \text{argmax}_b Q(s^{t+1}, b, \boldsymbol{\theta})$.
7: $\quad$ **if** $Q(s^{t+1}, a^*, \boldsymbol{\theta}) = Q(s^{t+1}, a^{t+1}, \boldsymbol{\theta})$ **then**
8: $\quad\quad a^* \leftarrow a^{t+1}$
9: $\quad$ **end if**
10: $\quad w_t \leftarrow \begin{cases} \epsilon(s^t)/\hat{\epsilon}(s^t), & \text{if } s^{t+1} \in T. \\ (1 - \epsilon(s^t))/(1 - \hat{\epsilon}(s^t)), & \text{if } s^{t+1} \notin T. \end{cases}$
11: $\quad$ target $y \leftarrow w_t(r^{t+1} + \gamma\hat{Q}(s^{t+1}, a^*, \boldsymbol{\theta}^-))$
12: $\quad \boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\nabla_{\boldsymbol{\theta}}\mathbb{E}[y - Q(s, a, \boldsymbol{\theta})]^2$
13: $\quad T(s^t) \leftarrow (1 - \alpha_T)T(s^t) + \alpha_T\epsilon(s^t)(r^{t+1} + \gamma Q(s^{t+1}, a^*, \boldsymbol{\theta}_{t+1}))$
14: $\quad U(s^t) \leftarrow (1 - \alpha_U)U(s^t) + \alpha_U(1 - \epsilon(s^t))(r^{t+1} + \gamma Q(s^{t+1}, a^*, \boldsymbol{\theta}_{t+1}))$
15: $\quad \hat{\epsilon}(s^t) \leftarrow min(max(\delta, \frac{|T(s^t)|}{|T(s^t)| + |U(s^t)|}), 1 - \delta)$
16: $\quad$ Every K steps, reset $\hat{Q} = Q$.
17: **end for**

---

that it fits the form of the equations in Corollary 3, and show that the conditions of Corollary 3 hold. We do this by letting $G_t(s, a) = w^t[r(s, a, X(s, a)) + \gamma \max_b \hat{Q}^t(X(s, a), b)] - Q^*(s, a)$, and by using the fact that $\mathbf{H}\hat{Q}$ is a contraction mapping, we show that $||\mathbb{E}[G_t(s, a)]||_\infty \leq \gamma||\hat{Q}^t(s, a) - Q^*||_\infty = \gamma||\triangle_t||_\infty$, and $\mathbf{var}[G_t(s, a)] \leq C(1 + ||\triangle_t||_\infty^2)$.

Rare Event Adaptive Eligibility Traces Algorithm (FIRE-ETAA): To speed up convergence for the tabular Q learning algorithm, we propose another variant of Algorithm 1 using eligibility traces. This algorithm (**ETAA**) is a combination of our proposed importance sampling based Q-learning (Algorithm 1) and Watkin's $Q(\lambda)$ [38], which is an algorithm which combines Q-learning with eligibility traces. We present this algorithm in our Technical Report [35].

7 SCALING TO LARGE STATE OR ACTION SPACES

To handle large state spaces, as might be found in real-world edge computing networks that span one or more cities and surrounding suburbs, we propose two variants of Algorithm 1 using deep Q-learning (Algorithm FIRE-ImDQL) and actor critic (Algorithm FIRE-ImACRE) respectively. The codes are available online at [39].

Rare Event Adaptive Deep Q-learning algorithm (FIRE-ImDQL, Algorithm 2): We modify and combine the classic deep RL algorithm in [40] with our importance sampling based rare events adaptive algorithm (Algorithm 1) by proposing an importance sampling based loss function. We parameterize the $\hat{Q}(s, a)$ value function using neural networks. The loss function is

$$L(\boldsymbol{\theta}) = \mathbb{E}_{s,a,s'}[w_t(r^{t+1} + \gamma \max_a \hat{Q}(s, a, \boldsymbol{\theta}^-)) - Q(s, a, \boldsymbol{\theta})]^2,$$

Algorithm 3 FIRE-ImACRE: Importance Sampling based Advantage Actor Critic for Rare Events

---

1: **Initialise:** parameters $\boldsymbol{\theta}, \boldsymbol{c}$ randomly, $\hat{T}(s), \hat{U}(s) \leftarrow 0$, $\hat{\epsilon}(s) \leftarrow \frac{1}{2}$, learning rates $\alpha_{\boldsymbol{\theta}}, \alpha_{\boldsymbol{c}}, \alpha_T, \alpha_U$.
2: Select the initial state s^0 and action a^0.
3: **for** all timeslots t **do**
4:      $\hat{\epsilon}(s^t)$ determines if an anomaly happens. Thereafter, sample according to $h(\tilde{s}|s, a)$. The new state s^{t+1} and a reward value $r^{t+1} = r(s^t, a^t, s^{t+1})$ is observed.
5:      Sample next action $a^{t+1} \sim \pi_{\boldsymbol{\theta}}(a|s^{t+1})$
6:      $w^t \leftarrow \begin{cases} \epsilon(s^t)/\hat{\epsilon}(s^t), & \text{if } s^{t+1} \in T. \\ (1-\epsilon(s^t))/(1-\hat{\epsilon}(s^t)), & \text{if } s^{t+1} \notin T. \end{cases}$
7:      $V(s^t) \leftarrow \boldsymbol{c}(s^t)$
8:      Advantage function $A(s^t, a^t) \leftarrow r^{t+1} + \gamma V(s^{t+1}) - w^t(s^t, a^t, s^{t+1})V(s^t)$
9:      Store $A(s^t, a^t)$ in buffer.
10:     $T(s^t) \leftarrow (1-\alpha_T)T(s^t) + \alpha_T \epsilon(s^t)(r^{t+1} + \gamma V(s^{t+1}))$
11:     $U(s^t) \leftarrow (1-\alpha_U)U(s^t) + \alpha_U(1-\epsilon(s^t))(r^{t+1} + \gamma V(s^{t+1}))$
12:     $\hat{\epsilon}(s^t) \leftarrow min(max(\delta, \frac{|T(s^t)|}{|T(s^t)|+|U(s^t)|}), 1-\delta)$
13:     **for** every K steps **do**
14:         Update policy parameters using data in buffer: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha_{\boldsymbol{\theta}}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{K}[\frac{1}{K}log\pi_{\boldsymbol{\theta}}(a^i|s^i)A(s^i, a^i)]$
15:         Update critic $\boldsymbol{c} \leftarrow \boldsymbol{c} - \alpha_{\boldsymbol{c}}\nabla_{\boldsymbol{c}}\sum_{i=1}^{K}\frac{1}{K}[A(s^i, a^i)]^2$
16:         Empty buffer.
17:     **end for**
18: **end for**

---

where $\boldsymbol{\theta}^-$ are the weights of the target neural network, $\boldsymbol{\theta}$ are the weights of the predicted neural network, which are updated every iteration, and w_t is the importance sampling weight. The loss function is optimized via gradient descent in line 13. We are trying to minimize the difference between the target $w_t(r^{t+1} + \gamma \max_a \hat{Q}(s, a, \boldsymbol{\theta}^-))$ and the prediction $Q(s, a, \boldsymbol{\theta})$. The target network is updated only every K iterations (line 16), while the prediction network is updated every iteration. Having the target fixed for awhile helps in having stable training. The key idea which differentiates our algorithm from traditional deep RL algorithms is that our target $w_t(r^{t+1} + \gamma \max_a \hat{Q}(s, a, \boldsymbol{\theta}^-))$ is "error corrected" by the importance sampling weight w_t, and the rare event distribution is updated via T, U and $\hat{\epsilon}$. The variables T, U, $\hat{\epsilon}$ and w_t are updated in the same manner as the tabular algorithm (Algorithm 1), through using the output of the prediction net $Q(s, a, \boldsymbol{\theta})$ where the input is state s^{t+1}. The actions are also selected in the same manner as Algorithm 1, through using the output of the prediction net $Q(s, a, \boldsymbol{\theta})$ where the input is state s^t.

Rare Event Adaptive Actor Critic Algorithm (FIRE-ImACRE, Algorithm 3): Neural networks are used to approximate the "actor" and "critic" functions [41]. The critic function estimates the value (network c), based on which the actor function updates the policy (network θ). The advantage function $A(s^t, a^t)$ (line 8) measures how good taking action a^t is, compared to taking the average action, at state s^t. It is a function of the reward r^{t+1}, the values $V(s^t)$ and $V(s^{t+1})$ (obtained from the critic network $\boldsymbol{c}$, line 7). It is corrected by the importance sampling correction weight

$w^t(s^t, a^t, s^{t+1})$ (line 8), because we perform importance sampling of rare events. Each iteration, $A(s^t, a^t)$ will be stored in the buffer (line 9). The variables $T, U, \hat{\epsilon}$ and w_t are updated in the same manner as the tabular algorithm (Algorithm 1). At the end of an epoch (K timeslots), both the actor and critic networks are optimized, via gradient descent. The gradients for the actor and critic networks, respectively, are:

$$\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{K}[\frac{1}{K}log\pi_{\boldsymbol{\theta}}(a^i|s^i)A(s^i, a^i)]. \tag{33}$$

$$\nabla_{\boldsymbol{c}} \sum_{i=1}^{K} \frac{1}{K}[A(s^i, a^i)]^2. \tag{34}$$

Both gradients are functions of the advantage value, stored in the buffer every iteration (line 9).

Both algorithms differ from traditional Deep Q learning and Actor Critic algorithms, as importance sampling of server failures is integrated.

Application to other Settings with Rare Events: Our algorithm can also be applied for other networked and wireless settings, in which there are rare events. An example application is UAV-assisted wireless networks with potential node failures. The failures are: UAV nodes failing (e.g. due to physical damage). The state space could include UAV positions and channel conditions, and the number of users served by each UAV. Possible actions include the user-UAV allocation, the bandwidth allocation, and the decision whether or not to deploying backup UAVs. The environment, state space, action space, and rare events would be specific to the new setting, while our importance-sampling algorithm itself would remain the same.

8 HETEROGENEOUS RISK TOLERANCES

We present our overall algorithm framework in Fig 3. Our importance sampling based reinforcement learning algorithms learns offline in a digital twin setup, to avoid experiencing the real cost of rare events. The converged policy is applied to online scenarios where rare events occur at their true rate. Running the policy once trained is identical to deploying any RL-trained policy. If we use, e.g., tabular RL, this is a table lookup; deep RL may require some inference time through the neural network.

Edge computing users, application vendors, and network providers have different tolerances towards server failures and the resulting higher latency which occur. Some applications are highly latency sensitive, and any server failure has a significant impact on safety and smooth functioning of the application, e.g., obstacle detection in autonomous vehicles, or immersive outdoor sport applications. Other users with less urgent jobs may not priortize a failure-aware backup placement solution, due to the extra costs of storing and migrating backups. To deal with heterogeneous risk tolerances, one potential solution is to alter the failure cost when training the algorithm. Nevertheless, the same rare event with the same system cost may lead to varying risk tolerances and willingness to incur a preparation cost amongst different users. Therefore, we propose a mechanism that can cater to edge computing users of heterogeneous risk tolerances. It takes the users' risk tolerance
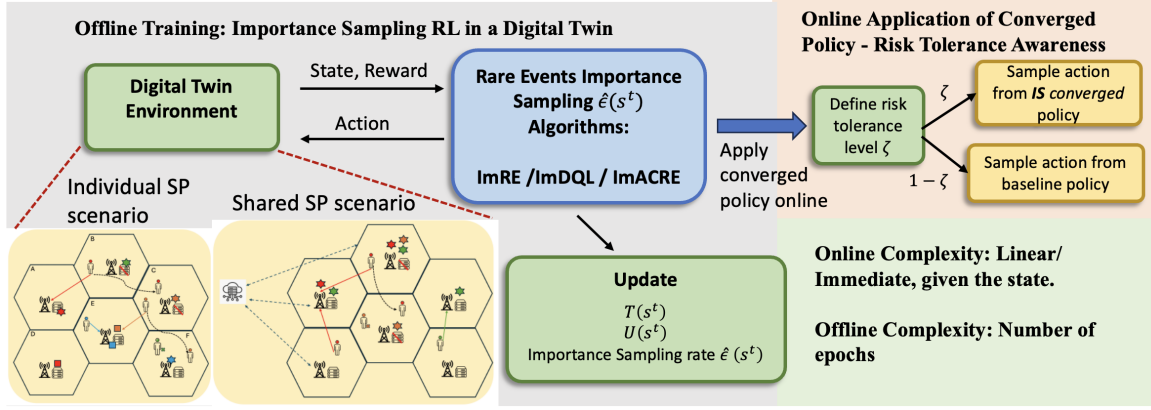
Fig. 3. Algorithm Framework: Our importance sampling based reinforcement learning algorithms learns offline in a digital twin setup, to avoid experiencing the real cost of rare events. The converged policy is applied to online scenarios where rare events occur at their true rate.

Algorithm 4 RiTA: Risk Tolerance-Adaptive Algorithm

1: **Input:** User risk tolerance level $\zeta \in [0, 1]$
2: $p_{riskAware} \leftarrow \text{softmax}(Q_{riskAware}(s, a))$
3: $p_{riskTaking} \leftarrow \text{softmax}(Q_{riskTaking}(s, a))$
4: **for** all timeslots t **do**
5:    State transition occurs according to $p(s'|s, a)$ and $\epsilon(s)$.
6:    **if** rand() $< \zeta$ **then**
7:       Sample an action according to $p_{riskTaking}$.
8:    **else**
9:       Sample an action according to $p_{riskAware}$
10:    **end if**
11: **end for**

---

towards server failures as an input, and derives a joint service placement and backup placement policy, based on the users' risk tolerance. With RiTA (Risk Tolerance-Adaptive Algorithm), we also do not need to re-train a new policy $Q_{riskAware}$ for every possible level of risk, which scales better to a wider variety of users.

We present this algorithm in Algorithm 8 (RiTA). Firstly, RiTA takes the user's risk tolerance level, ζ as input. The lower a user's risk tolerance, the more the user wants to avoid the cost arising from server failure. The higher a user's risk tolerance, the less the user is willing to prepare for failure. Next, this algorithm converts $Q_{riskAware}$ to a vector of probabilities $p_{riskAware}$, where $Q_{riskAware}$ is the converged policy trained from one of our importance-sampling failure-adaptive Q-learning simulator algorithms (Algorithms 1, 2, or 3). The algorithm also converts $Q_{riskTaking}$ to a vector of probabilities $p_{riskTaking}$, where $Q_{riskTaking}$ is the converged policy of a Q-learning algorithm that does not take into account rare events in modelling of their environment, and does not consider the possibility of backups. This represents the risk taking component. Algorithm 8 is an online algorithm for real-time scenarios. For each timeslot, the state transition occurs according to $p(s'|s, a)$, and rare events according to the true rare events probability $\epsilon(s)$. At each timeslot, the policy which is used will be in accordance with the risk tolerance level $\zeta \in [0, 1]$. The lower the risk tolerance ζ, the higher the probability that

the action selected will be sampled according to $p_{riskAware}$ (lines 6-10), obtained from our algorithm's converged policy. When $\zeta = 0$, the action selected will be according to our algorithm's converged policy. In the next theorem, we characterize the condition under which it would have been beneficial to prepare for rare events.

Theorem 5. *After T time-slots, at risk level ζ, assuming our algorithm is able to prevent the cost of failures with probability $1 - f$, it benefited the user to prepare for rare events, if*

$$F_c > \frac{N_c((\zeta - 1)f + \zeta)}{(\zeta - (1 - \zeta)f)} +$$
$$\frac{T[(1 - \zeta)(N_c + B_c) - \zeta N_c]\sum_{i=0}^{T}\binom{T}{i}\epsilon^i(1 - \epsilon)^{T-i}}{(\zeta - (1 - \zeta)f)\sum_{i=0}^{T}\binom{T}{i}\epsilon^i(1 - \epsilon)^{T-i}i},$$

where F_c, N_c, B_c respectively denote the average failure cost, normal state cost, and backup placement and migration cost.

Proof. The number of failures follow a binomial distribution with probability ϵ. We use this to model the expected cost of preparation, and cost of not preparing for rare events. $\square$

9 SIMULATION EXPERIMENTS

In this section, we provide numerical evaluation for our algorithms FIRE-ImRE, FIRE-ImDQL, and FIRE-ImACRE, validating the results in Section 6. In particular, we show that we have solved the main research challenges identified in Section 1. We show that *FIRE-ImRE, FIRE-ImDQL and FIRE-ImACRE converged to optimality*. We also show that they were able to *learn a policy which mitigates the high cost of server failures*, unlike the baselines in which importance sampling or backups were not used, given a tradeoff of incurring a higher cost during normal states due to backup storage and migration. Moreover, *we can adapt to varied risk tolerances without re-training*. We perform our experiments with the help of real world traces [42], to obtain latency (delay) costs across different user-server location pairs in the network.

We compare our algorithms with the following **baselines**, over 10 runs. Each of these baselines, besides the greedy baseline, may be trained with deep Q-learning or

(a) Convergence of tabular importance sampling Q-learning algorithm FIRE-ImRE.

(b) Convergence of actor critic importance sampling algorithm FIRE-ImACRE

(c) Convergence of deep Q-learning importance sampling algorithm FIRE-ImDQL, shared SP scenario.

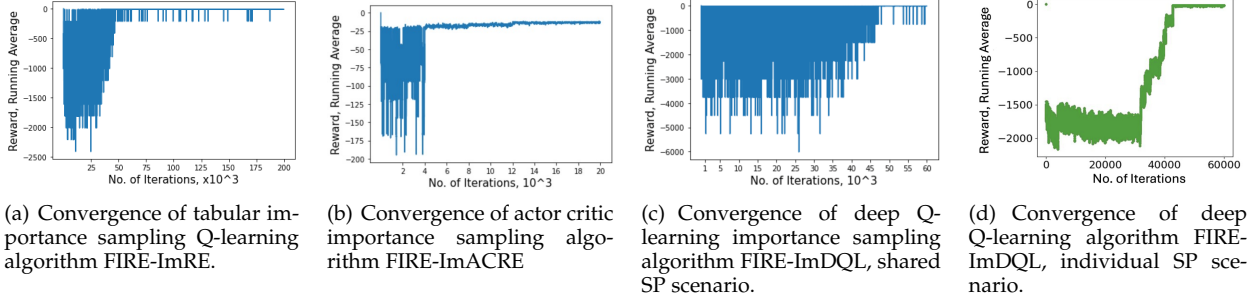(d) Convergence of deep Q-learning algorithm FIRE-ImDQL, individual SP scenario.

Fig. 4. **Convergence graphs:** Here, FIRE-ImRE and FIRE-ImACRE are applied to the a special case of the scenario (Section 3) where every user has their own service profile (the single user case), and FIRE-ImDQL is applied to the scenario where users share service profiles (Section 4). All three variations of our algorithm converge.

(a) Comparison of average cost.

(b) Comparison of average cost at rare state, when failure occurs.

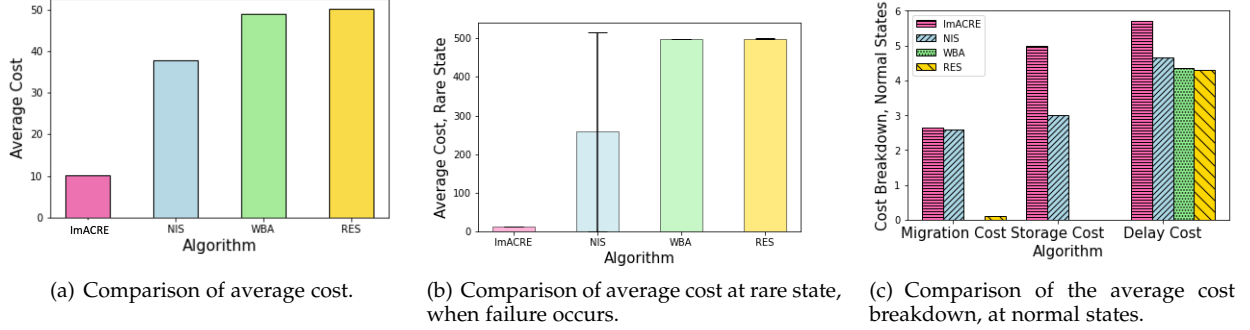(c) Comparison of the average cost breakdown, at normal states.

Fig. 5. **Single user service migration scenario:** Comparison of our actor critic algorithm importance sampling FIRE-ImACRE, with actor critic versions of the baselines NIS, WBA and RES, in an online scenario. FIRE-ImACRE leads to lower costs on average and in rare failure states, but higher storage and delay costs in normal states.

actor critic (to correspond to our algorithm). We will compare by applying the converged policy of each algorithm in an online scenario where rare events occur at their true rate.

RL with No Importance-Sampling (NIS): This simulator is a traditional Q-learning algorithm, without importance sampling of rare events. Rare events are simulated at their true rate, and backups are a possible action. Comparison to this baseline indicates the role of importance sampling in helping the learned policy adapt to rare events.

RL without Backups as an Action (WBA): This simulator is a Q-learning algorithm in which there is no importance sampling of rare events and backups are not used as possible actions. Rare events are simulated at their true rate. Comparison to this baseline indicates the importance of backups in mitigating failure costs.

RL without Rare Events Sampled (RES): This simulator is a Q-learning algorithm under the "normal scenario", where rare events and backups are not considered during training. Comparison to this baseline indicates the importance of including failures in the digital twin training environment.

Greedy Placement (GPM): Services and backups are placed at the top most likely locations of users, based on the current location of users, similar to reactive migration methods in the existing literature [43]. Comparison to this baseline indicates the importance of using learned policies instead of heuristics to optimize migration policies.

Single User Service Profile Migrations: In our experiments, we have 9 base stations (edge APs). The user is highly mobile across these coverage areas, and its mobility pattern follows a transition matrix. We use the same

network settings as in [19] to simulate dynamic latencies between user locations and different base stations as users move around the service area: In particular, the ns-3 network simulator is used [42] to simulate dynamic channel network conditions, to obtain these latencies. These latencies are in the range (2,10). As both the latency costs and migration costs are functions of the distance between user and APs, we set the migration cost $m_{ij} = d_{i,j}^{comm} + \varepsilon$, where $\varepsilon \in (-0.5, 0.5)$. We set the cost of failures to be -500, representing the high cost in terms of user experience or safety, and the true failure rate as 0.01 (1%) [44]. We choose this failure rate, because if the failure rate is of a higher order (e.g. 15 percent), then we may not need a special failure-aware importance sampling strategy as the 'failures' are no longer rare events - they appear often enough for normal RL methods to learn a policy which deals with them. Conversely, if the failure rate is lower (e.g. 0.001=0.1 percent), the failures would give a smaller expected cost on the system (Definition 1 in Section 3.1), so are less important to handle. The learning rate used is 0.05 and both the actor and critic networks, for both our algorithms and the baselines, have a structure of $[24, 48, 24]$ nodes per layer.

We apply our importance sampling based actor critic algorithm FIRE-ImACRE and the corresponding actor critic baselines NIS, WBA, RES, for this scenario. In Fig. 4a, we show the convergence of FIRE-ImACRE. In Fig. 5, we plot the overall average cost, average cost at rare states, and at normal states, for our importance sampling based actor critic algorithm FIRE-ImACRE and the corresponding actor critic baselines NIS, WBA, RES. All algorithms are run 10

(a) Comparison of average cost.

(b) Comparison of average cost at rare state, when failure occurs.

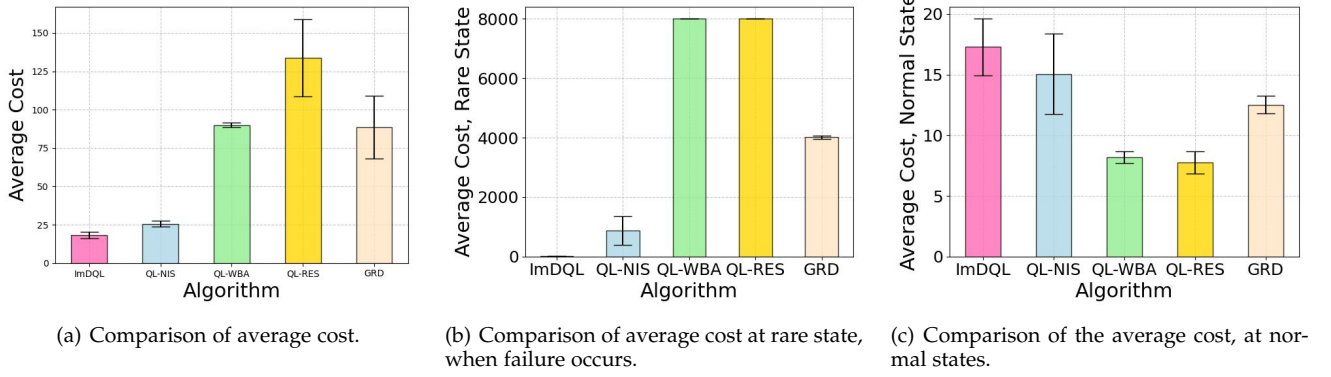(c) Comparison of the average cost, at normal states.

Fig. 6. **Multiple user service profile migrations:** Comparison of our deep Q-learning algorithm FIRE-ImDQL, with baselines QL-NIS, QL-WBA and QL-RES, and greedy algorithm, in an online scenario. FIRE-ImDQL yields much lower costs in rare states, yielding lower costs on average despite a slight increase in cost for normal states.

(a) Comparison of average cost.

(b) Comparison of average cost at rare state, when failure occurs.
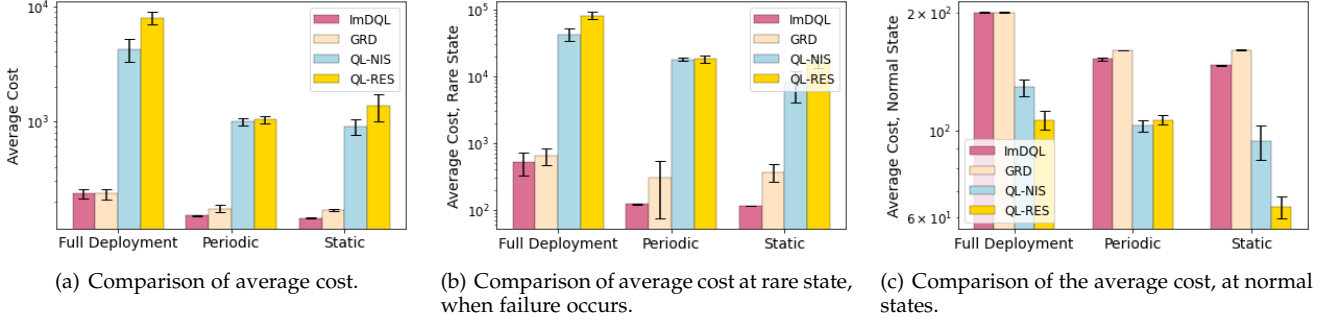
(c) Comparison of the average cost, at normal states.

Fig. 7. **Shared service profile migrations:** Comparison of our deep Q-learning algorithm FIRE-ImDQL, with baselines NIS, RES, and greedy algorithm, in three different settings. For all settings, our algorithm achieves a lower cost at rare states, and on average, than the baselines.

times, and the average is plot. Our algorithm achieves a lower average cost at rare states, because FIRE-ImACRE sufficiently samples rare events to learn a policy which prepares for them. Our algorithm incurs a trade-off with higher normal state costs, incurred through the migration and storage of backups. It can be seen in Fig. 5b that Algorithm NIS avoids the elevated costs associated with rare events in certain runs. However, its performance exhibits a greater variability. WBA and RES incur the highest costs in rare events due to the lack of backup usage and the exclusion of rare events from sampling during RES simulator training.

Multiple User Service Profile Migrations: Here, we consider service profile migration for **multiple users**, each with their own service profile. We model 13 users mobile across 9 APs. Parameter settings are similar to the setup above. The natural failure rate is set to be 0.01. User mobility patterns are sampled according to a transition probability matrix. We model *different server failure types*, corresponding to the differing amounts of maintenance resources at each AP, which affect the time taken for the servers to come back online. For APs 1,6, and 8, we let their service downtime be 2 time-slots per failure. For all other APs, we let their service downtime be 1 time-slot per failure. We apply algorithm FIRE-ImDQL, our importance sampling based deep Q learning algorithm in this scenario. In Fig. 6, we see that our FIRE-ImDQL policy outperforms the baselines by achieving a lower average cost during rare states, with a trade-off of a higher average cost in normal (non-rare) states. Overall, our algorithm still achieves a lower cost, when averaged over all

timeslots. QL-NIS and the greedy algorithm perform better than the other baselines, as backups are still a potential action for the algorithm in QL-NIS, and the greedy algorithm places one backup at the second most likely location every time-slot. Nevertheless these algorithms do not avoid the rare event cost as much as our algorithm does.

Comparison with PPO: In Table 1, we present results in which our algorithm is compared with PPO (proximal policy optimization), for the multiuser scenario, where users have individual service profiles. The results are averaged over 6 runs. It outperforms PPO when the converged policies of both algorithms are compared in an online setting where rare events occur at their true rate: PPO has a higher average cost over time, and higher rare event cost, as there is no importance sampling and hence lack of rare event data to train a policy which is adaptive to rare events.

Higher failure rates: We have further performed new experiments with higher failure rates, namely 5 and 10% (Tables 2 and 3). Our algorithm is able to handle higher failure rates. Note that if failures are fairly common, then we may not need to use importance sampling to amplify rare events in the training dataset, as the rare failure events would naturally occur at a fairly high rate. Thus, reinforcement learning algorithms without importance sampling should still be able to learn good policies that account for failure costs, as these costs would be present in the training dataset. Indeed, we observe this effect in our results. At a failure rate of 5 percent (Table 2), our algorithm still outperforms the baselines, achieving a much lower mean rare events reward.

(a) Individual SP scenario: Average reward across rare states.

(b) Individual SP scenario: Average reward across normal states.

(c) Shared SP scenario: Average Cost across all states.

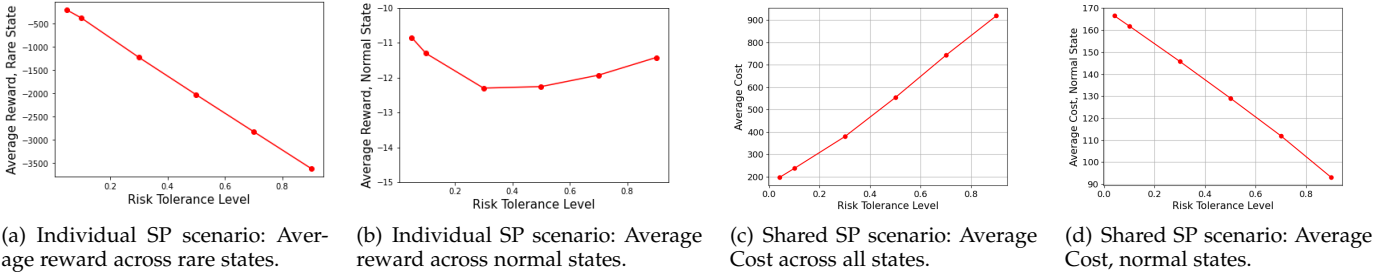(d) Shared SP scenario: Average Cost, normal states.

Fig. 8. **Heterogeneous Risk Tolerances:** Evaluating how the risk tolerance levels impact the cost, when Algorithm RiTA is applied. A higher risk tolerance leads to higher cost and lower reward in rare states, but may decrease the cost in normal states.

TABLE 1
Comparison of ImDQL with PPO, averaged over 6 runs. For both algorithms, the converged policy is applied in an online scenario where rare events occur at their true rate.

Reward across time, online setting	ImDQL	PPO
Avg reward across time	-18.210	-22.031
Mean RE reward	-18.689	-551.79
Mean RE migration cost	-6.688	-4.628
Mean RE delay	-5.176	-5.342
Mean RE storage cost	-5.000	-4.798
Mean RE computing cost	-0.9289	-0.889
Mean NS reward	-17.279	-14.584
Mean NS migration cost	-6.968	-4.656
Mean NS delay	-5.311	-5.171
Mean NS storage cost	-5.000	-4.757
Mean NS comp cost	-0.908	-0.895

QL-NIS outperforms QL-WBA and QL-RES in the online scenario because backups and rare events are considered in training. At a failure rate of 10 percent (Table 3), the performance of QL-NIS improves, as it is easier learn a policy in light of not-as-rare server failures. While they are of the same order, our algorithm still outperforms it.

TABLE 2
Comparison of Rewards (Costs) in Online Setting. Failure rate of 5%.

Reward across time	ImDQL	QL-NIS	QL-WBA	QL-RES
Avg reward across time	-19.586	-26.062	-416.322	-578.736
Mean RE reward	-20.679	-186.516	-8002.600	-8003.399
Mean RE migration cost	-7.663	-6.284	-2.600	-3.399
Mean RE delay	-5.189	-4.785	0.000	0.000
Mean RE storage cost	-4.999	-4.914	0.000	0.000
Mean RE computing cost	-0.957	-1.029	-0.929	-0.993
Mean NS reward	-18.394	-15.483	-8.061	-7.804
Mean NS migration cost	-8.053	-6.332	-2.609	-3.456
Mean NS delay	-5.344	-4.244	-5.452	-4.349
Mean NS storage cost	-4.997	-4.907	0.000	0.000
Mean NS computing cost	-0.924	-1.038	-0.932	-1.003

Shared service profiles: In this scenario (Fig. 7), we consider multiple users sharing the same service profile (SP). We model 100 users, with 70 users using SP Type 1, and 30 using SP Type 2. These users are spread over a 2-city map, with 2APs in one city and 3 in another. It has a failure cost

TABLE 3
Comparison of Rewards (Costs) in Online Setting. Failure rate of 10%.

Reward across time	ImDQL	QL-NIS	QL-WBA	QL-RES
Avg reward across time	-20.218	-18.665	-416.322	-578.736
Mean RE reward	-23.414	-42.013	-8002.600	-8003.399
Mean RE migration cost	-7.643	-5.568	-2.600	-3.399
Mean RE delay	-5.124	-4.652	0.000	0.000
Mean RE storage cost	-4.997	-4.984	0.000	0.000
Mean RE computing cost	-0.943	-1.027	-0.929	-0.993
Mean NS reward	-18.265	-14.698	-8.061	-7.804
Mean NS migration cost	-7.943	-5.633	-2.609	-3.456
Mean NS delay	-5.328	-4.082	-5.452	-4.349
Mean NS storage cost	-4.994	-4.983	0.000	0.000
Mean NS computing cost	-0.910	-1.043	-0.932	-1.003

of 1000 and failure rate of 1e-2 [44]. If a user's required AP is in another city, the delay cost between cities will be nearly as large as the failure cost, which is larger than the delay cost when accessing an SP within the same city. We apply FIRE-ImDQL to this scenario and compare its performance with NIS, RES and the greedy baseline. The greedy baseline deploys SPs to the N most probable places w.r.t distribution of user transitions, where we search through all N and plot the results with the lowest costs.

We look at 3 settings: a) *Full Deployment*, where users are spread across zones evenly. The greedy baseline's optimal solution is to fully deploy all SPs. b) *Periodic*, where users move across areas periodically, and c) *Static*, a corner case where users stay in their current zone. As seen in Fig. 7, in all settings, FIRE-ImDQL achieves a lower cost in rare event states, followed by GRD (greedy), as compared to QL-NIS and QL-RES, which place little or no backups. While there is a trade-off with a higher cost in normal states, on average over all states, FIRE-ImDQL and GRD still achieve a lower cost (Fig. 7 a) than the other baselines. FIRE-ImDQL outperforms GRD in the periodic and static settings, as the greedy baseline cannot adapt to the presence of rare failures that may occur in each city.

Catering to varying risk tolerances: We apply our algorithm RiTA (Algorithm 8), which caters to users having different risk tolerances. Given the risk tolerance level of a user ζ, with probability $1 - \zeta$, RiTA will select an action

15

according to the converged policy of our algorithm FIRE-ImRE (risk adverse component), and with probability ζ RiTA will select an action according to the converged policy of the baseline QL-RES (risk taking component). We apply RiTA to the single user (3 APs) service migration setting, which is a special case of the individual SP scenario, and to the multiple user shared SP scenario (5 APs across 2 cities).

In Fig. 8 we present how the risk tolerance level ζ impacts the reward levels. As seen in Figs. 8 a) and b), for the single user individual SP scenario, the higher the risk tolerance, the higher the cost experienced at rare states (as it is less likely backups were placed). For normal states, as the risk tolerance level ζ decreases from $\zeta = 1$, the cost first increases due to the increase in storage of backups (resulting from being increasingly risk adverse). After $\zeta = 0.3$, the cost decreases. This is because as ζ decreases, the weightage given to the risk adverse component (our algorithm FIRE-ImRE) increases. While FIRE-ImRE incurs a higher storage cost than RES, it manages to learn a policy in which frequent migrations do not occur, hence decreasing the normal state cost. For the shared service profile scenario, as the risk tolerance ζ increases, the cost at normal state decreases due to the placement of less services, while the average cost (over all states) increases. This is because multiple users share service profiles like game environments, multiplying the cost when failures occur.

Future work: Scalability to a large number of users: If users do not share service profiles, resource allocation is mostly de-coupled, except for the 'queuing delay at server' cost. Under this setting, the algorithm may not be as scalable for larger state spaces, because in our importance sampling algorithm, we update the functions $T(s)$ and $U(s)$ at state level, necessitating us to sample each state multiple times. Future areas of investigation can include modifying the importance sampling-RL algorithm (which is trained offline), to sample multiple trajectories in parallel while updating T and U, for faster convergence. Another future area of investigation is to do "intelligent" or dynamic decoupling of the state space over base stations, via splitting the network into smaller subsets of groups of APs based on user mobility patterns. This makes the state space smaller. The algorithm can then be run on each 'smaller state space'.

10 CONCLUSION

In edge computing, server failures may spontaneously occur, which complicates resource allocation and service migration. While failures are considered as rare events, they have an impact on the smooth and safe functioning of edge computing's latency sensitive applications. As these failures occur at a low probability, it is difficult to jointly plan or learn an optimal service migration solution for both the typical and rare event scenarios. Therefore, we introduce a rare events adaptive resilience framework named FIRE, using the placement of backups and importance sampling based RL, which alters the sampling rate of rare events in order to learn an optimal risk-averse policy. We prove the boundedness and convergence to optimality of our proposed tabular Q-learning algorithm FIRE-ImRE. To handle large and combinatorial state and action spaces common in

real-world networks, we propose deep Q-learning (FIRE-ImDQL) and actor critic (FIRE-ImACRE) versions of our algorithm. Furthermore, we propose a decision making algorithm (RiTA) that caters to heterogeneous risk tolerances across users. Finally, we use trace driven experiments to show that our algorithms converge to optimality, and are resilient towards server failures, unlike several baselines, subject to a trade-off in terms of a potential higher normal state cost. Our framework can also be modified for other resource allocation problems pertaining rare events with large consequences in communication and networking.

REFERENCES

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.
[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.
[3] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM ToN*, vol. 24, no. 5, pp. 2795–2808, 2015.
[4] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on markov decision process," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, 2019.
[5] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE JSAC*, vol. 36, no. 10, pp. 2333–2345, 2018.
[6] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23 511–23 528, 2018.
[7] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Transactions on Mobile Computing*, vol. 20, no. 3, pp. 939–951, 2019.
[8] B. Gao, Z. Zhou, F. Liu, F. Xu, and B. Li, "An online framework for joint network selection and service placement in mobile edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.
[9] M. Siew, K. Guo, D. Cai, L. Li, and T. Q. Quek, "Let's share vms: Optimal placement and pricing across base stations in mec systems," in *IEEE INFOCOM 2021*. IEEE, 2021, pp. 1–10.
[10] D. Zeng, L. Gu, S. Pan, J. Cai, and S. Guo, "Resource management at the network edge: A deep reinforcement learning approach," *IEEE Network*, vol. 33, no. 3, pp. 26–33, 2019.
[11] L. Ponemon, "Cost of data center outages," *Data Center Performance Benchmark Serie*, 2016.
[12] A. Aral and I. Brandić, "Learning spatiotemporal failure dependencies for resilient edge computing services," *IEEE TPDS*, vol. 32, no. 7, pp. 1578–1590, 2020.
[13] Z. Gao, Q. Jiao, K. Xiao, Q. Wang, Z. Mo, and Y. Yang, "Deep reinforcement learning based service migration strategy for edge computing," in *2019 IEEE SOSE*, 2019, pp. 116–1165.
[14] J. Frank, S. Mannor, and D. Precup, "Reinforcement learning in the presence of rare events," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 336–343.
[15] H. Hong, Q. Wu, F. Dong, W. Song, R. Sun, T. Han, C. Zhou, and H. Yang, "Netgraph: An intelligent operated digital twin platform for data center networks," in *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration*, 2021, pp. 26–32.
[16] D. Precup, "Eligibility traces for off-policy policy evaluation," *Computer Science Department Faculty Publication Series*, p. 80, 2000.
[17] Y. Zhang, L. Jiao, J. Yan, and X. Lin, "Dynamic service placement for virtual reality group gaming on mobile edge cloudlets," *IEEE JSAC*, vol. 37, no. 8, pp. 1881–1897, 2019.
[18] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 516–529, 2018.
[19] T. Kim, S. D. Sathyanarayana, S. Chen, Y. Im, X. Zhang, S. Ha, and C. Joe-Wong, "Modems: Optimizing edge computing migrations for user mobility," in *IEEE INFOCOM 2022*. IEEE, 2022, pp. 1159–1168.

[20] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Transactions on Communications*, vol. 67, no. 6, pp. 4132–4150, 2019.

[21] H. Huang and S. Guo, "Proactive failure recovery for nfv in distributed edge computing," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 131–137, 2019.

[22] G. Yao, X. Li, Q. Ren, and R. Ruiz, "Failure-aware elastic cloud workflow scheduling," *IEEE TSC*, pp. 1–14, 2022.

[23] Y. Liang, Z. Hu, and L. Yang, "A two-stage replica management mechanism for latency-aware applications in multi-access edge computing," in *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2021, pp. 453–459.

[24] W. Du, Q. He, Y. Ji, C. Cai, and X. Zhao, "Optimal user migration upon server failures in edge computing environment," in *2021 IEEE International Conference on Web Services (ICWS)*. IEEE, 2021, pp. 272–281.

[25] M. Siew, S. Sharma, and C. Joe-Wong, "Acre: Actor critic reinforcement learning for failure-aware edge computing migrations," in *2023 57th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2023, pp. 1–6.

[26] M. Mudassar, Y. Zhai, and L. Lejian, "Adaptive fault-tolerant strategy for latency-aware iot application executing in edge computing environment," *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 13 250–13 262, 2022.

[27] P. A. Apostolopoulos, E. E. Tsiropoulou, and S. Papavassiliou, "Risk-aware data offloading in multi-server multi-access edge computing environment," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1405–1418, 2020.

[28] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Risk-aware application placement in mobile edge computing systems: A learning-based optimization approach," in *2020 IEEE International Conference on Edge Computing (EDGE)*, 2020, pp. 83–90.

[29] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[30] U. Madhushani, B. Dey, N. E. Leonard, and A. Chakraborty, "Hamiltonian q-learning: Leveraging importance-sampling for data efficient rl," *arXiv preprint arXiv:2011.05927*, 2020.

[31] R. Dong, C. She, W. Hardjawana, Y. Li, and B. Vucetic, "Deep learning for hybrid 5g services in mobile edge computing systems: Learn from a digital twin," *IEEE Transactions on Wireless Communications*, vol. 18, no. 10, pp. 4692–4707, 2019.

[32] K. Zhang, J. Cao, and Y. Zhang, "Adaptive digital twin and multi-agent deep reinforcement learning for vehicular edge computing and networks," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 2, pp. 1405–1413, 2021.

[33] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *IEEE INFOCOM 2019*. IEEE, 2019, pp. 1468–1476.

[34] J. A. Bucklew and J. Bucklew, *Introduction to rare event simulation*. Springer, 2004, vol. 5.

[35] "This paper's technical report." [Online]. Available: https://tinyurl.com/RareEvTechReport

[36] J. N. Tsitsiklis, "Asynchronous stochastic approximation and q-learning," *Machine learning*, vol. 16, no. 3, pp. 185–202, 1994.

[37] T. Jaakkola, M. Jordan, and S. Singh, "Convergence of stochastic iterative dynamic programming algorithms," *Advances in neural information processing systems*, vol. 6, 1993.

[38] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[39] "Code for algorithm imdql." [Online]. Available: https://github.com/ShikhSh/MultiUser-FIRE

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[41] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[42] "ns3 network simulator." [Online]. Available: https://www.nsnam.org/

[43] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, "A survey on mobility-induced service migration in the fog, edge, and related computing paradigms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–33, 2019.

[44] J. Liu, A. Zhou, C. Liu, T. Zhang, L. Qi, S. Wang, and R. Buyya, "Reliability-enhanced task offloading in mobile edge computing environments," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 10 382–10 396, 2021.

Marie Siew is a Faculty Early Career Award (FECA) Fellow at the Singapore University of Technology and Design.

Shikhar Sharma was a Masters Student in Carnegie Mellon University.

Zekai Li was a Masters Student in ECE, Carnegie Mellon University.

Kun Guo is a Professor at East China Normal University, Shanghai.

Chao Xu is a Professor in Northwest A&F University, China.

Tania Lorido Botran is a Research Scientist at Roblox.

Tony Q.S. Quek is the Cheng Tsang Man Chair Professor with Singapore University of Technology and Design (SUTD) and ST Engineering Distinguished Professor.

Carlee Joe-Wong is the Robert E. Doherty Associate Professor of Electrical and Computer Engineering at Carnegie Mellon University.